Network compression and faster inference using spatial basis filters

Roy Miles
Department of Electrical and Electronic Engineering
Imperial College London
London, UK
r.miles18@imperial.ac.uk

Krystian Mikolajczyk
Department of Electrical and Electronic Engineering
Imperial College London
London, UK
k.mikolajczyk@imperial.ac.uk

***Abstract*—We present an efficient alternative to the convolutional layer through utilising spatial basis filters (SBF). SBF layers exploit the spatial redundancy in the convolutional filters across the depth to achieve overall model compression, while maintaining the top-end accuracy of their dense counter-parts. Training SBF-Nets is modelled as a simple pruning problem, but instead of zeroing out the pruned channels, they are replaced with inexpensive transformations from the set of non-pruned features. To enable an adoption of these SBF layers, we provide a flexible training pipeline and an efficient implementation in CUDA with low latency. To further demonstrate the effective capacity of these models, we apply semi-supervised knowledge distillation that leads to significant performance improvements over the baseline networks. Our experiments show that SBF-Nets are effective and achieve comparable or improved performance to state-of-the-art across CIFAR10, CIFAR100, Tiny-ImageNet, and ILSCRC-2012.**

I. Introduction

Convolutional neural networks (CNNs) have achieved state-of-the-art results across a range of computer vision tasks [8, 39]. Despite their success, these models are typically far too large and computationally expensive for their deployment on resource constrained devices, such as mobile phones or edge devices in IoT. Recent work on compressing CNNs [28, 29, 47, 54], have exploited the inherent weight redundancy as attributed to the training in a highly over-parameterised regime. Pruning [13, 26] attempts to rank the importance of the weights/filters and remove those below a given threshold. However, at high pruning rates, some of these filters can still contribute to the attainable top-end accuracy. To this end, we propose a low-rank decomposition of the convolutional filters, whereby a subset of these filters are reconstructed using computationally inexpensive spatial transformations of the non-pruned filters, which we call the basis filters. Our approach follows a simple and non-exhaustive pruning/fine-tuning pipeline for transferring models to a compressed spatial-basis-filter (SBF) network at a given target pruning rate. This methodology can be considered a natural extension of pruning, but instead of zeroing out the pruned filters, we are replacing them with a low-rank approximation. To demonstrate the effective capacity of these networks, we train a small student network based on SBF using the supervision of a larger baseline teacher network without any ground truth labels. In most of these experiments, the student is able to outperform the baseline teacher, which demonstrates the effectiveness of the proposed approach. We further provide an efficient CUDA implementation of our proposed SBF layers and demonstrate their improved performance over state-of-the-art methods on the CIFAR dataset and comparable results on ImageNet. Our contributions are as follows:

- We propose a novel approach based on cheap spatial transformations of CNN basis filters, that can compensate for the reduced image representation due to pruning.
- We introduce a grouped extension of spacial transformations, which enables the preservation of a small bottleneck ratio for very effective knowledge distillation.
- We further provide an efficient CUDA implementation of our proposed SBF layers.
- Our results show significant improvement over state-of-the-art on the CIFAR and Tiny-ImageNet datasets, while comparable results on ImageNet.

A. Related work

The most relevant work to our approach can be divided into pruning, low-rank decomposition, and efficient convolutions.

Pruning explicitly exploits the inherent parameter redundancy by removing individual weight entries or entire filters that have the least contribution to the performance on a given task. This was first introduced in [13, 26] using the Hessian of the loss to derive a saliency measure for the individual weights. SNIP [27] proposed to prune weights using the connection sensitivity between individual neurons. Subsequent work propose a sparse neuron skip layer [42] to achieve fast training convergence and a high connectivity between layers. Cheap heuristic measures have also been used, such as the magnitude [12, 28], geometric median [15], or average percentage of zeros [19]. Although some of these unstructured pruning methods are able to achieve significant model size compression, the theoretical reduction in floating-point operations (FLOPs) does not translate to the same practical improvements without the use of dedicated sparse hardware and software libraries. This has led to the more widespread adoption of structured pruning approaches, which focus on removing entire filters. [54] introduced additional loss terms to select the channels with the highest discriminative power, while [49] proposed to prune in accordance with a neural

importance score. DMCP [10] models the pruning operation as a differentiable markov chain, where compression is achieved through a sparsity inducing prior. Similarly, [1, 52] model pruning in the probabilistic setting using both hierarchical and sparsity inducing priors. We model the rank search of the proposed low-rank decomposition as a pruning problem. However, we propose to use a simple, linear iterative pruning/fine-tuning pipeline that only spans 5-10 epochs of training. We observe that the selection of basis filters is flexible on the choice of heuristic saliency. In fact, we demonstrate this claim through using both magnitude and first-order approximations, which both yield strong results.

Low-rank decomposition is concerned with compactly representing a high-dimensional tensor, such as the convolutional weights, as linear compositions of much smaller, lower-dimensional, tensors, called factors. Any linear operations that are then parameterised by these weights can be expressed using cheaper operations with these factors, which can lead to a reduction in the computational complexity. Depthwise separable convolutions split the standard convolution into two stages, the first extracts the local spatial features in the input, while the second aggregates these features across channels. They were originally proposed in Xception [5] but have since been adopted in the design of a range of efficient models [3, 9, 18, 44]. This has led to the development of optimized GPU kernels that bridge the gap between the theoretical FLOP improvements and the practical on-device latency. Both CP-decomposition [17] and Tucker decomposition [45] have also been used to construct or compress pre-trained models [21, 23, 35]. Another line of work has explored the use of tensor networks as a mathematical framework for generalising tensor decomposition in the context of deep learning [14, 46]. Our proposed decomposition is similar to the Ghost modules [11]. However, there are a few distinct differences, specifically the SBF-Net features are spatial transformations of the basis features. Spatial transformations are not possible in Ghost modules due to the intermediate contraction over the patch dimensions. In fact, GhostNet modules can be seen as specific cases of SBF layers. We also propose to learn a suitable rank for the decomposition through a pruning pipeline, as opposed to introducing this value as an additional hyper-parameter.

II. Method

In this section, we propose the low-rank decomposition of the convolutional weights used in SBF layers. We discuss the main components of the model and their training.

A. Spatial basis filters (SBF) layer

a) SBF.: Let $\mathcal{W} = \{\mathcal{W}_n \in \mathbb{R}^{K\times K\times C}\}_{n=1}^{N}$ describe the set of filters for a given convolutional layer with an input depth C, output depth N, and a receptive field size of $K \times K$. Our approach relies on an assumption that a large subset of these filters can be faithfully approximated as spatial transformations of a much smaller set of filters, which we call the basis filters $\mathcal{B} \subset \mathcal{W}$. We observe that the expressiveness of the family of transformations does not hinder the attainable accuracy of the model (see Supplementary). In light of this observation, we express the transformations as cheap element-wise product between the spatial entries of the basis filters. Thus, for a $K \times K$ basis filter, the transformations can then be parameterised using $K \times K$ learnable weights. Without loss in generality, consider the case for N basis filters and N output feature maps. The proposed decomposition is then given as follows:

$$\mathcal{Y}_{h,w,n} = \sum_{k_w,k_h}^{K}\sum_{i}^{C} \mathcal{X}_{h',w',i} \cdot \mathcal{W}_{k_h,k_w,c,n} \tag{1}$$

$$\approx \sum_{k_w,k_h}^{K}\sum_{i}^{C} \mathcal{X}_{h',w',i} \cdot \mathcal{B}_{k_h,k_w,i,n} \cdot \mathcal{T}_{k_h,k_w,n}, \tag{2}$$

$$h' = (h-1)s + k_h - p, \quad w' = (w-1)s + k_w - p$$

where s is the stride and p is zero-padding. This is illustrated in Figure 1 (left). Strictly speaking, model compression is only achieved when the number of basis filters M is less than the number of output feature maps N. This is realised through pruning, which is discussed in section II-B. In this case, the basis filters are then re-used to compute different feature maps using different transformations.

b) Basis filter allocation.: The choice of mapping from which basis filter to which output feature map is not critical, as long as it is fixed after the pruning stage to enable the fine-tuning. We set this mapping as $i = j \ mod \ M$, where the ith output feature map is allocated the jth basis filter, and where M is the number of basis filters. This ensures that all basis filters are uniformly used throughout. The spatial transformations are then jointly learned.

c) Two stage processing.: Computing the output features using the transformed filters and the original dense convolution would still not lead to any efficiency improvements. Thus we propose to decompose the convolutional layer into two stages, the first computes the basis features $\mathcal{Z}$ using filters $\mathcal{B}$, while the second projects these features to a different space through the spatial transformations $\mathcal{T}$. The final set of output features is then the union of the original basis features (identity transformations) and the transformed features.

$$\mathcal{Y}_{h,w,n} \approx \sum_{k_w,k_h}^{K}\sum_{i}^{C} \mathcal{X}_{h',w',i} \cdot \mathcal{B}_{k_h,k_w,i,n} \cdot \mathcal{T}_{k_h,k_w,n} \tag{3}$$

$$= \sum_{k_w,k_h}^{K} \mathcal{T}_{k_h,k_w,n} \underbrace{\left(\sum_{i}^{C} \mathcal{X}_{h',w',i} \cdot \mathcal{B}_{k_h,k_w,i,n}\right)}_{\mathcal{Z}} \tag{4}$$

Computing $\mathcal{Z}$ can be achieved using a pointwise convolution, which translate to an optimised General Matrix Multiply (GeMM) primitive. However, the second stage, which consists of projecting $\mathcal{Z}$ to the output space, reduces to a series of gather operations and multiplications. Although this can be implemented in existing deep learning frameworks, the practical efficiency gains will be severely limited. To this

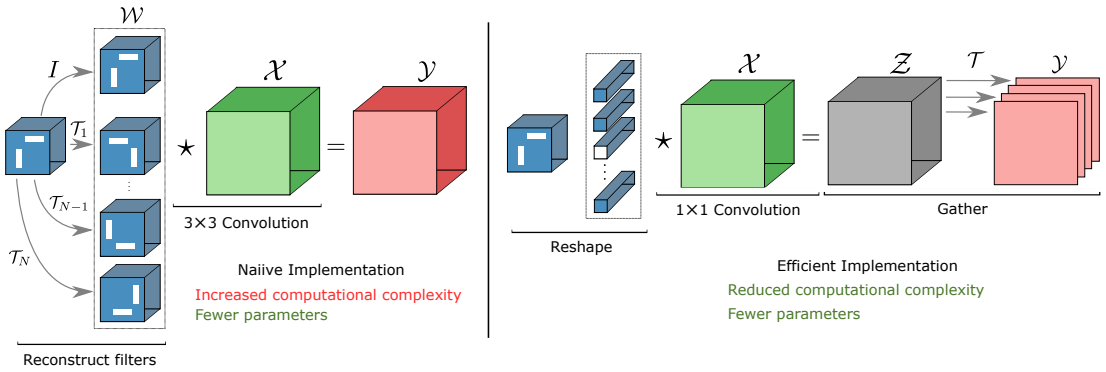


Fig. 1. Basis filters and their transformed filters in a naive implementation (see left) results in an increased computational cost. Filter transformations $\mathcal{T}$ are linear operations and shared across the input depth, therefore they can be moved outside the summation in Eq.2, resulting in two consecutive operations which are computationally efficient. These two cases are illustrated on the left and the right, respectively. Computing the basis features $\mathcal{Z}$ involves reshaping the 3×3 basis filters into 9 pointwise filters. Each transformation then acts on these groups of 9 feature maps.

end, we provide an efficient CUDA implementation[1] such that the theoretical reduction in FLOPs translates to practical improvements in terms of the computational latency.

B. Network pruning

The important parameter that will control the trade-off between the attainable accuracy and the overall model-size/computational complexity is the number of the basis filters for each layer throughout the network. We propose to approach this as a pruning problem. Our method starts with the original set of filters all assigned as basis filters with identity transformations. All filters are then evaluated using a heuristic saliency measure and a threshold is used to select a subset of filters for the basis. The choice of saliency measure is not critical as long as a reasonably uniform pruning rate across all of the layers is achieved. For our experiments we use magnitude based pruning [28] for the Residual based networks, and a first order Taylor expansion of saliency [26] for the VGG based networks. This threshold is increased over a series of epochs until a given global pruning rate is achieved. Filters below this global threshold are pruned away and replaced with transformations of an individual remaining basis filter.

C. Channel connectivity

By design, the proposed spatial basis layers preserve the same number of input and output channels as the original convolution. This choice is crucial for preserving the same dimensionality of intermediate features as the original network. In fact, we believe that this greatly reinforces the supervision signal provided through knowledge distillation during training as discussed in the next section. However, the consequence of this design is that at high pruning rates there will be a significant bottleneck in the latent space $\mathcal{Z}$ of the basis filters (see figure 2). To quantify this situation, we introduce the term 'bottleneck ratio', which we define as the ratio of the input size $\mathcal{X}$ to the size of latent space $\mathcal{Z}$. A high bottleneck ratio can result in losing too much information from the input, thus hinder the downstream performance. Ideally,

[1] http://github....

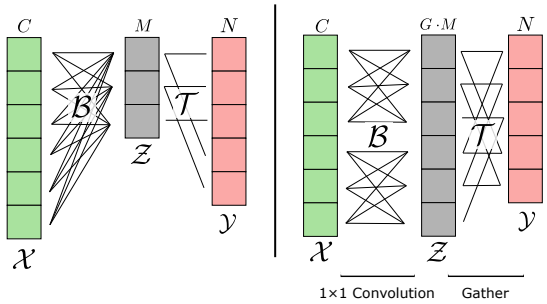


Fig. 2. (left) Significant bottleneck with number of groups $G = 1$ and basis filters $M = 3$. (right) Introducing the proposed grouped extension with e.g. $G = 2$, widens the size of the intermediate representations under the given compression rate, with a minimal additional computational cost. Note that transformations $\mathcal{T}$ are computationally much less expensive than convolutions with basis filters $\mathcal{B}$. The cost of $\mathcal{B}$ side remains the same while we increase the number of basis filters for reduced depth C/G.

we would like to minimize this bottleneck ratio, preserve the dimensionality of input and output feature maps, without incurring significant parameter or computational overheads. To do this, we propose to introduce a grouped extension of the SBF layers that can naturally scale the dimensionality of this latent space with a minimal overhead. To do this we replace the *pointwise convolution* in the two stage processing with a *grouped pointwise convolution* [25], which has an efficient implementation in most deep learning frameworks. The function computed by the custom CUDA op in the second stage is then modified to incorporate a group parameter. This transformation operation then translates to the sum of G transformations applied to feature maps from the G distinct groups. Doing so in this way enables cross-group information flow without the need for any channel shuffles [51]. Figure 2 graphically demonstrates this grouped extension. On the left is the original case, whereby $G = 1$. At this pruning rate, there is a very large bottleneck ratio. Increasing the groups to 2 (as shown on the right) provides a natural scheme for increasing the depth of $\mathcal{Z}$ without incurring any significant computational overhead.

The standard convolution has the computational cost of the

order of $H \cdot W \cdot K^2 \cdot C \cdot N$, whereas the cost of SBF layers is given by:

$$(H \cdot W \cdot K^2 \cdot \frac{C}{\not{G}} \cdot M \cdot \not{G}) + (H \cdot W \cdot K^2 \cdot G \cdot N) \quad (5)$$

Where M indicates the number of basis filters and G is the number of groups. The reduction in computation is subsequently given by:

$$\frac{(H \cdot W \cdot K^2 \cdot C \cdot M) + (H \cdot W \cdot K^2 \cdot G \cdot N)}{H \cdot W \cdot K^2 \cdot C \cdot N} = \frac{M}{N} + \frac{G}{C} \quad (6)$$

Note that we prune the basis filters such that $M \ll N$ and we set $G \ll C$. We improve the bottleneck problem by using $G \cdot M$ basis filters applied to C/G channels that are efficiently implemented with grouped convolutions. Finally, we ensure cross-group information flow by increasing the number of inexpensive transformations that are then applied cross group.

D. Knowledge distillation

Our SBF networks are trained using Knowledge Distillation (KD) [16], which has proved to be effective in training various compressed models [7, 22, 34, 37, 40]. Although knowledge distillation has also been extended to intermediate activations [34, 36, 38, 48], we focus our attention on KD applied at the output.

This methodology uses the soft predictions from a much larger teacher model as pseudo ground truth labels for the student. In doing so, the student is able to more easily learn the correlations between classes. The loss is then typically reformulated as the weighted sum of the original cross entropy, with the ground truth labels, and a KL divergence term between the softened student and teachers predictions:

$$\mathcal{L} = \omega H(y, \sigma(y_S)) + (1 - \omega) KL(\sigma(y_T/\tau), \sigma(y_S/\tau)) \quad (7)$$

Where y_T and y_S are the teacher and student predictions from the original and pruned SBF network respectively, while y is ground truth label, τ is a temperature term, and $\sigma(\cdot)$ is the softmax activation function. Note that setting $\omega = 0$ will make the training rely entirely on the output of the teacher and the ground truth will not be used.

III. Experiments

In this section we evaluate our approach on the CIFAR and ImageNet datasets. We give more insights into the benefits of using different pruning rates and numbers of groups. The performance is reported in terms of the number of parameters in the network, the number of multiplications and additions (MAdds), as well as the top-1 classification accuracy. All of these models are trained on a single NVIDIA RTX 2080Ti GPU using stochastic gradient descent (SGD). The initial learning rate for all experiments was set to 0.001. For CIFAR10, CIFAR100, and Tiny-ImageNet this learning rate was subsequently reduced by a factor of 4 every 40 epochs, whereas for ImageNet we used a linear decay schedule. Unless stated otherwise, we set the minimum number of basis filters for each layer and the number of groups to both be 4 throughout.

A. Comparison on CIFAR10 and CIFAR100

The CIFAR10 dataset [24] consist of 60K 32×32 RGB images across 10 classes and with a 5:1 training/testing split. The chosen VGG16 [39] architecture is modified for this dataset with batch normalisation layers after each convolution block and by reducing the number of classification layers to two; of depth 512 and 10 respectively. The CIFAR100 dataset is very similar to CIFAR10, except that it contains 100 classes with 600 images per class. During training, we augment the datasets using random horizontal flips, and random 32×32 crops. For CIFAR10, we also introduce random rotations. Finally, the baseline architectures are trained for 300 epochs with a step learning rate decay. We use a first-order approximation for the saliency [26] to rank the filters and select the basis filters. A simple linear pruning schedule is also adopted that spans the first 10 epochs. This is in contrast to most of the other pruning methods [33, 54], which use longer pruning stages, or introduce additional layer-by-layer stopping conditions.

Method	Top-1 Baseline Accuracy (%)	Params	MAdds	Top-1 Accuracy (%)
Original		14.98M	313M	71.51%
Principle filter analysis [41]	68.40%	4.96M	179M	$\uparrow$ **1.40%**
Correlation-based pruning [47]	72.59%	4.01M	178M	$\downarrow$ 0.82%
SBF-VGG16	71.51%	4.83M	196M	$\downarrow$ 0.02%
		3.32M	174M	$\downarrow$ 0.73%
global pruning rate		2.08M	144M	$\downarrow$ 2.51%
Pr=[0.6, 0.7, 0.8, 0.9]		1.19M	105M	$\downarrow$ 7.25%

TABLE I
COMPARISON TO OTHER PRUNING METHODS ON CIFAR100. THE SBF-NETWORKS ARE TRAINED FOR 300 EPOCHS USING TEACHER SUPERVISION ONLY I.E. $\omega = 0$ IN EQ. 7.

The results for CIFAR100 are shown in table I and the CIFAR10 are shown in figure 3. These results are compared to other state-of-the-art pruning methods, which can be seen as specific cases of SBF-Networks whereby $\mathcal{T} = 0$. This comparison shows that SBF-Nets achieve significantly improved accuracy v.s. performance trade-offs. Interestingly, we achieved the same results as shown in these tables with random weight initialisation as we did by initialising with the weights from the original model. Experimental results with magnitude-based pruning and first-order Taylor approximation are also shown in figure 6 (right) and demonstrate that the heuristic used to evaluate the basis filters is not significant.

We performed an ablation on the KD weighting used in equation 7 (see figure 4), where we observed that these SBF-Networks can be trained solely using the supervision of the teacher at various pruning rates. In light of this result, we use the loss weight $\omega = 0$ and temperature $\tau = 5$ in Equation 7. Despite this limitation of no ground truth labels, our models are still able to match and even outperform the original networks, which are used as the teachers. This is attributed to the effective capacity of our SBF-Networks and the label smoothing properties induced through the temperature term τ. Our results for $\omega = 0$ i.e. no ground truth used, suggest that the SBF-networks can provide a solution for compressing

larger teacher model in cases where no labelled data is available. To further isolate the benefits of the proposed spatial transformations, we train and prune the VGG16 network with and without the use of these transformations. The results are shown in figure 6 (left) across a range of complexity constraints, where cheap spatial transformations consistently improve the performance.

B. Comparison on Tiny-Imagenet and ILSVRC-12

For experiments on Tiny-Imagenet and ILSVRC-12 we use the ResNet-50 architecture and a magnitude based saliency measure [28] with the mean magnitude per layer subtracted. Magnitude pruning was empirically shown to provide a more uniform pruning rate across all of the residual blocks - see figure 6 (middle). A uniform pruning across all of the layers enables the total MAdds and parameter count to reduce at a similar rate. Similarly with the CIFAR experiments, we also set the minimum number of basis filters per layer and the number of groups per layer to be 4. The models were trained for 200 epochs with a linear learning rate decay. For Tiny-ImageNet, we used $\omega = 0$ and the results are shown in table II. The SBF networks show significant improvements over the baseline architecture, despite significant model size compression and reduction in computational complexity.

Method	Top-1	Top-5	Params	MAdds
ResNet50	56.02%	78.05%	24.5M	1.34B
SBF-ResNet50 Pr=[0.7, 0.8, 0.9, 0.95]	↑ 0.78%	↑ 1.06%	15.9M	1.00B
	↑ 0.64%	↑ 0.81%	14.8M	0.94B
	↑ 0.64%	↑ 0.48%	14.0M	0.88B
	↑ 0.30%	↑ 1.08%	13.8M	0.85B

TABLE II
EVALUATION ON THE TINY IMAGENET DATASET WITH THE RESNET-50 ARCHITECTURE. SBF NETWORKS IMPROVE UPON THE RESNET-50 ACCURACY WHILE REDUCING THE COMPUTATIONAL COST.

For the ImageNet experiments shown in Table III, we modified the original ResNet50 to have the $strides = 2$ inside the 3×3 convolutional layers as opposed to the pointwise projections. This modification has been employed in various other implementation of ResNet [54], which we compare our results to. Inception-style data augmentation [43] was also used in addition to modest saturation, brightness, and hue colour augmentations. We were not able to achieve comparable results using knowledge distillation alone. Thus, after 50 epochs of training, we increases ω from 0 to 0.2. This was to be expected, since most KD based methods do not provide positive results on the ImageNet dataset [4]. The ImageNet results show comparable performance to state-of-the-art pruning methods without the need for extensive pruning and fine-tuning pipelines. We expect that incorporating these state-of-the-art pruning methodologies into SBF-Networks would further improve upon on the attainable accuracy.

C. Ablation experiments

a) Group extension.: To demonstrate the benefit of our group extension, we train an SBF-VGG16 network at varying pruning rates and number of groups. The results in figure 4 (left) show that at high pruning rates, whereby the SBF layer incurs a large bottleneck ratio, increasing the number of groups can then provide a significant improvement in the accuracy. Although increasing the minimum number of basis filters per layer can also partially address this problem as shown in figure 4 (middle), it would come at a significant computational overhead. Increasing the number of groups, on the other hand, has a much lower cost, although there is some impact in terms of latency, as attributed to the non-contiguous memory access. Figure 4 (right) shows that training with KD leads to better performance.

b) Faster inference using an efficient implementation.: To encourage the adoption of our SBF layers, we provide an efficient implementation in CUDA. The results are shown in figure 5 and demonstrate that at even modest pruning rates, practical on-device performance is improved. The pointwise 1×1 convolutions contribute most significantly to the total on-device GPU compute time, whereas our custom CUDA operation (named TransformGather) has a minimal contribution.

IV. CONCLUSION

In this paper, we proposed the use of spatial basis filters which provides a natural extension of the standard pruning pipeline. Instead of zeroing out the pruned filters, they are replaced with cheap spatial transformations from the remaining non-pruned filters. These trained SBF networks are able to achieve comparable or improved results on the image classification task across a range of datasets and architectures. We also introduced grouped filters that allow to improve the bottleneck problem at minimal computational cost. Our approach applied to VGG16 and ResNet50 is able to significantly reduce the models size and computational cost while retaining the top recognition accuracy on CIFAR and ImageNet datasets.

We provide an efficient CUDA implementation that demonstrates improved latency, which will facilitate the use of our SBF layers in new architectures. We expect that this layer can be further optimised through fusing the two stages into a single CUDA kernel, as is done with depthwise-separable convolutions. The excellent results by using Knowledge Distillation with no ground truth labels suggest that these SBF-networks can be trained as an efficient substitute of large models in cases where no labelled data is available. With the recent works on data-free knowledge distillation [2, 32], this could even be further extended to the case where no data is available, which provides an interesting future direction.

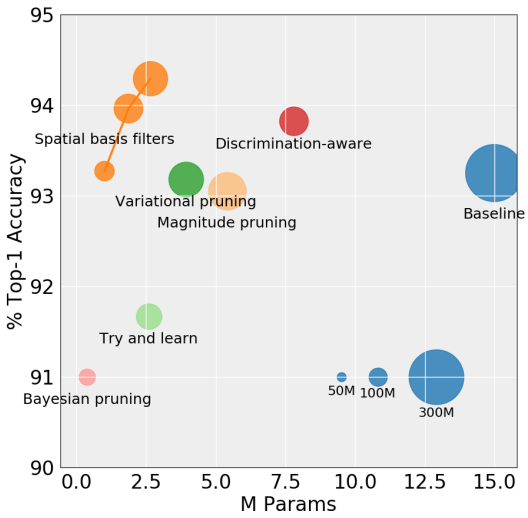
REFERENCES

[1] Mart Van Baalen et al. "Bayesian Bits : Unifying Quantization and Pruning". In: NeurIPS (2020).

[2] Hanting Chen et al. "Data-free learning of student networks". In: *ICCV* (2019).

[3] Hong Yen Chen and Chung Yen Su. "An Enhanced Hybrid MobileNet". In: *iCAST* (2018).

[4] Jang Hyun Cho and Bharath Hariharan. "On the efficacy of knowledge distillation". In: *ICCV* (2019).



Method	Params	MAdds	Top-1
Original	14.98M	313M	93.26%
Variational pruning [52]	3.92M	190M	↓ 0.07%
Geometric median [15]	–	237M	↓ 0.34%
Try-and-learn [20]	2.59M	140M	↓ 1.10%
performance drop bound	2.02M	111M	↓ 1.90%
b=[1, 2, 4]	1.08M	61M	↓ 3.40%
Magnitude pruning [28]	5.40M	206M	↓ 0.15%
Bayesian pruning [53]	**0.38M**	**89M**	↓ 0.60%
HRank [30]	2.64M	109M	↓ 1.62%
Discrimination-aware [54]	7.80M	157M	↓ 0.17%
DCP, DCP-Adapt	0.96M	109M	↓ 0.58%
SBF-VGG16	2.66M	188M	↑ **1.03**%
global pruning rate	1.87M	158M	↑ 0.70%
Pr=[0.7, 0.8, 0.9]	1.01M	107M	↑ 0.01%

Fig. 3. Performance v.s. accuracy of spatial basis filters at different pruning rates on the CIFAR10 with VGG16. Left compares the models on a scatter plot, where the size of the points is proportional to the computational cost, while right shows the quantitative results.

Method	Top-1 Baseline Accuracy (%)	Params	MAdds	Top-1 Accuracy (%)	Top-5 Accuracy (%)
Original		25.5M	3.86B	75.00%	92.11%
Discrimination-aware [54]	76.01	12.38M	1.72B	↓ 1.06%	↓ 0.61%
Bayesian Pruning [53]	76.10	-	1.68B	↓ 3.10%	↓ 2.90%
NISP [49]	-	18.58M	2.81B	↓ 0.21%	-
Filter Sketch [29]	76.13	14.53M	2.23B	↓ 1.45%	↓ 0.69%
ThiNet [33]	72.88	12.28M	1.71B	↓ 1.87%	↓ 1.12%
MetaPruning [31]	76.60	-	2.00B	↓ 1.20%	-
HRank [30]	76.15	16.15M	2.30B	↓ 1.17%	↓ 0.54%
SBF-ResNet50 MAG	75.00	14.60M	2.29B	↓ 1.40%	↓ 0.54%
SBF-ResNet50 FO		15.00M	2.95B	↓ 1.09%	↓ 0.27%

TABLE III
COMPARISON TO OTHER FILTER PRUNING METHODS ON THE IMAGENET DATASET USING THE RESNET-50 ARCHITECTURE. TIMINGS SHOW THE RELATIVE DIFFERENCE IN INFERENCE COMPUTE TIME FOR THE ORIGINAL MODEL AND THE PRUNED SBF-RESNET50.

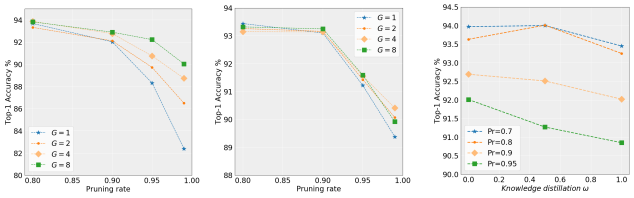


Fig. 4. (left) Using more groups enables a smaller bottleneck ratio, which is beneficial for preserving the top-end accuracy. In these sets of experiments the minimum number of basis filters is set to 1. (middle) Increasing the minimum number of basis filters (in this case to 8) incurs a larger computational overhead than increasing the number of groups. (right) The impact of knowledge distillation at varying pruning rates, with $\omega = 1$ showing results without KD.

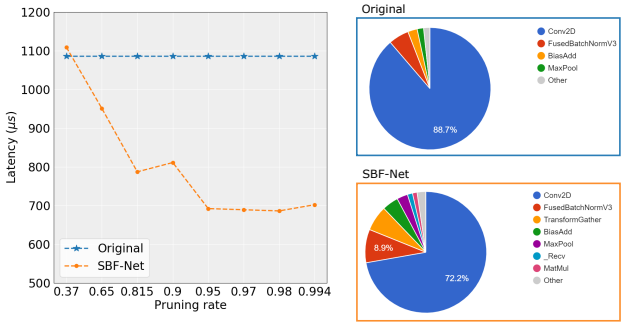


Fig. 5. (left) Compute time of the original VGG16 architecture and SBF-VGG16 at various pruning rates with the efficient CUDA implementation for the SBF layer. (right) Cumulative contribution of each operation on the total on-device compute time.

[5] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CVPR* (2017).

[6] Taco S. Cohen and Max Welling. "Group equivariant convolutional networks". In: *ICML* (2016).

[7] Elliot J. Crowley, Gavin Gray, and Amos Storkey. "Moonshine: Distilling with cheap convolutions". In: *NeurIPS* (2018).

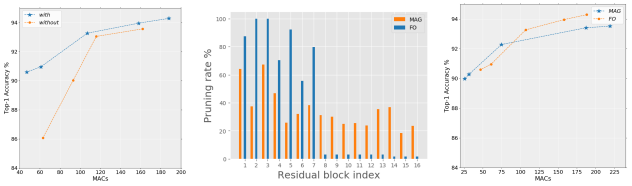


Fig. 6. (left) Performance of VGG16 networks trained with and without spatial transformations using the same training methodology and saliency measure. (middle) The allocation of basis filters for each layer under two different saliency measures, namely magnitude (MAG) and the first-order Taylor approximation (FO) for the ResNet50 architecture on ImageNet. (right) accuracy v.s. computational cost of the SBF-VGG16 at different pruning rates on the CIFAR10 dataset using both magnitude and FO saliency measures.

[8] Daniel Detone, Tomasz Malisiewicz, and Andrew Rabinovich. "SuperPoint: Self-supervised interest point detection and description". In: *ICPR* (2018).

[9] Michael H. Fox, Kyungmee Kim, and David Ehrenkrantz. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *CVPR* (2018).

[10] Shaopeng Guo et al. "DMCP: Differentiable Markov Channel Pruning for Neural Networks". In: *CVPR* (2020).

[11] Kai Han et al. "GhostNet: More Features from Cheap Operations". In: *CVPR* (2019).

[12] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: *ICLR* (2015).

[13] Babak Hassibi and David G Stork. "Second order derivatives for network pruning: Optimal Brain Surgeon". In: *NeurIPS* (1993).

[14] Kohei Hayashi et al. "Einconv: Exploring Unexplored Tensor Decompositions for Convolutional Neural Networks". In: *NeurIPS 2019* (2019).

[15] Yang He et al. "Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration". In: *CVPR* (2018).

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *NeurIPS* (2015).

[17] Frank L. Hitchcock. "The Expression of a Tensor or a Polyadic as a Sum of Products". In: *JMP* (2015).

[18] Andrew Howard et al. "Searching for MobileNetV3". In: *ICCV* (2019).

[19] Hengyuan Hu et al. "Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures". In: *arXiv* (2016).

[20] Qiangui Huang et al. "Learning to prune filters in convolutional neural networks". In: *WACV* (2018).

[21] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Speeding up convolutional neural networks with low rank expansions". In: *BMVC*. 2014.

[22] Jangho Kim, Seong Uk Park, and Nojun Kwak. "Paraphrasing complex network: Network compression via factor transfer". In: *NeurIPS* (2018).

[23] Yong-Deok Kim et al. "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications". In: *ICLR* (2015).

[24] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: (2009).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *NeurIPS* (2012).

[26] Yann Lecun. "Optimal Brain Damage". In: *NeurIPS* (1990).

[27] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H.S. Torr. "SnIP: Single-shot network pruning based on connection sensitivity". In: *ICLR*. 2019.

[28] Hao Li et al. "Pruning Filters For Efficient Convnets". In: *ICLR* (2017).

[29] Mingbao Lin et al. "Filter Sketch for Network Pruning". In: *arXiv* (2019).

[30] Mingbao Lin et al. "HRank: Filter Pruning using High-Rank Feature Map". In: *CVPR* (2020).

[31] Zechun Liu and Tim Kwang-ting Cheng. "MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning". In: ().

[32] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. "Data-Free Knowledge Distillation for Deep Neural Networks". In: *NeurIPS Workshop on Learning with Limited Data* (2017).

[33] Jian Hao Luo, Jianxin Wu, and Weiyao Lin. "ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression". In: *ICCV*. 2017.

[34] Roy Miles and Krystian Mikolajczyk. "Cascaded channel pruning using hierarchical self-distillation". In: *BMVC* (2020).

[35] Roy Miles and Krystian Mikolajczyk. "Compression of descriptor models for mobile applications". In: *ICASSP* (2021).

[36] Wonpyo Park et al. "Relational Knowledge Distillation". In: *CVPR* (2019).

[37] Qi Qian, Hao Li, and Juhua Hu. "Efficient Kernel Transfer in Knowledge Distillation". In: *arXiv* (2020).

[38] Adriana Romero et al. "FitNets: Hints For Thin Deep Nets". In: *ICLR* (2015).

[39] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks For Large-scale Image Recognition". In: *ICLR* (2015).

[40] Suraj Srinivas and François Fleuret. "Knowledge transfer with jacobian matching". In: *ICML* (2018).

[41] Xavier Suau, Luca Zappella, and Nicholas Apostoloff. "Filter distillation for network compression". In: *WACV* (2020).

[42] Arvind Subramaniam and Avinash Sharma. "N2NSkip: Learning Highly Sparse Networks using Neuron-to-Neuron Skip Connections". In: *BMVC* (2020).

[43] Christian Szegedy et al. "GoogLeNet/Inception - Going deeper with convolutions". In: *CVPR*. 2015.

[44] Mingxing Tan et al. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". In: *CVPR* (2018).

[45] Ledyard R Tucker. “Some mathematical notes on three-mode factor analysis”. In: *Psychometrika* (1966).
[46] Wenqi Wang, Brian Eriksson, and Wenlin Wang. “Wide Compression : Tensor Ring Nets”. In: *CVPR* (2018).
[47] Wenxiao Wang et al. “COP: Customized deep model compression via regularized correlation-based filter-level pruning”. In: *IJCAI* (2019).
[48] Junho Yim. “A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning”. In: *CVPR* (2017).
[49] Ruichi Yu et al. “NISP: Pruning Networks Using Neuron Importance Score Propagation”. In: *CVPR* (2018).
[50] Hongyi Zhang et al. “Mixup: Beyond empirical risk minimization”. In: *ICLR* (2017).
[51] Xiangyu Zhang, Xinyu Zhou, and Mengxiao Lin. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: *CVPR* (2018).
[52] Chenglong Zhao et al. “Variational Convolutional Neural Network Pruning”. In: *CVPR* (2019).
[53] Yuefu Zhou et al. “Accelerate CNN via Recursive Bayesian Pruning”. In: *ICCV* (2018).
[54] Zhuangwei Zhuang et al. “Discrimination-aware Channel Pruning for Deep Neural Networks”. In: *NeurIPS* (2018).

V. Appendix

A. Appendix

B. Using mixup regularisation

Knowledge distillation can also be used during training and can be used in conjunction with mixup regularisation [50], we coin this *mixup distillation*. Let $f_T(\cdot)$ and $f_S(\cdot)$ be the functions defined by the student and teacher networks respectively. We use the following definitions for building the mixup distillation loss.

$$\bar{x} = \lambda x_i + (1-\lambda)x_j \quad \text{where } x_i, x_j \text{ are the input images} \tag{8}$$

$$\bar{y} = \lambda y_i + (1-\lambda)y_j \quad \text{where } y_i, y_j \text{ are one-hot label encodings} \tag{9}$$

$$y_T = \lambda f_T(x_i) + (1-\lambda) f_T(x_j) \tag{10}$$

$$y_S = f_S(\bar{x}) \tag{11}$$

Where $\lambda \sim Beta(\alpha, \alpha)$. The final mixup distillation loss is then given as follows.

$$\mathcal{L} = \omega H(\bar{y}, y_S) + \underbrace{(1-\omega) KL(y_T, y_S)}_{\mathcal{L}_{KD}} \tag{12}$$

This loss in equation 12 is also shown concisely in figure 7a. We expect that the teacher models are not typically trained using *mixup*, which motivates the choice of mixing at the output. Through our experiments on the CIFAR datasets (see 7b), we observed little to no improvement when using this proposed *mixup* extension. We expect that this result is attributed to the fact that it requires a significant network capacity to behave linearly between the teacher predictions.

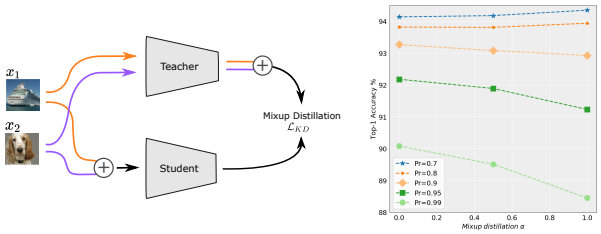


Fig. 7. (left) Graphical representation of the proposed *mixup* distillation. This training loss encourages the student to behave linearly between the soft teacher's predictions, as opposed to the ground truth labels. (right) shows the attainable accuracy's on the CIFAR10 dataset at different values of α. The student networks typically did not benefit much from *mixup*, which we expect is attributed to their lower capacity.

C. Family of transformations

We considered using more expressive families of spatial transformations for the basis filters and observed that this typically resulted in a poorer accuracy v.s. performance trade-off. Specifically, the improved accuracy came at a much more significant overhead in terms of both the computational cost and the model size.

We first looked at the case where these transformations are taken from the General Linear group $GL(K)$ and subsequently parameterised using a matrix $\mathcal{T}_i \in \mathbb{R}^{K^2 \times K^2}$, where the filters are of size $K \times K$. We also considered the transformations invoked using matrix multiplications G, and then finally the cheap element-wise products H which are used throughout the experiments in this paper. This gives rise to the following relation $H < G < GL(K)$. The results are shown in figure 8 and show that at even significant compression rates, the reduction in computational complexity is severely limited when using expressive transformations. However, introducing subgroups of $GL(K)$ [6], for example $SL(K)$, and methods for learning discrete entries, could enable these transformations to be implemented as simple memory movement i.e. zero FLOP transformations.

D. Pruning allocation

Figure 9 shows the number of basis filters chosen for each residual block on an SBF-ResNet50 at a pruning rate of 0.95. This model was trained on ImageNet and the basis filters were selected using a magnitude based saliency measure. To encourage a more uniform pruning rate across all of the residual blocks, we subtracted the mean magnitude per layer for filter salience.

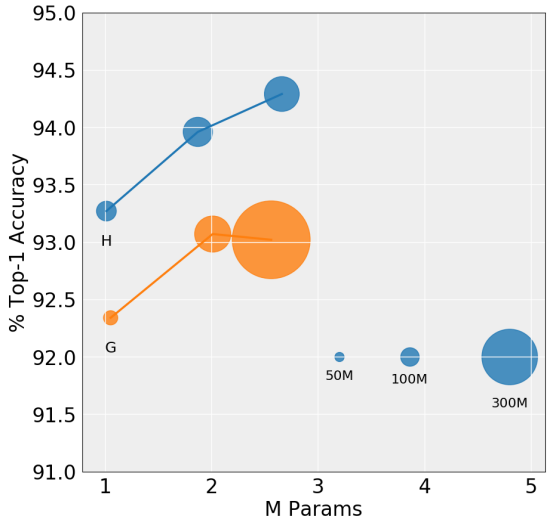


Fig. 8. Introducing more expressive transformations would hinder the attainable accuracy under the given training methodology and scale very poorly in terms of the computational cost at low pruning rates. These experiments are performed on the VGG16 architecture with the CIFAR-10 dataset. 'H' inidcates using the element-wise product between spatial entries of the basis filters, while 'G' uses a matrix multiplication. Using the most expressive set of spatial transformations given by $GL(3)$ performs much worse than shown here.

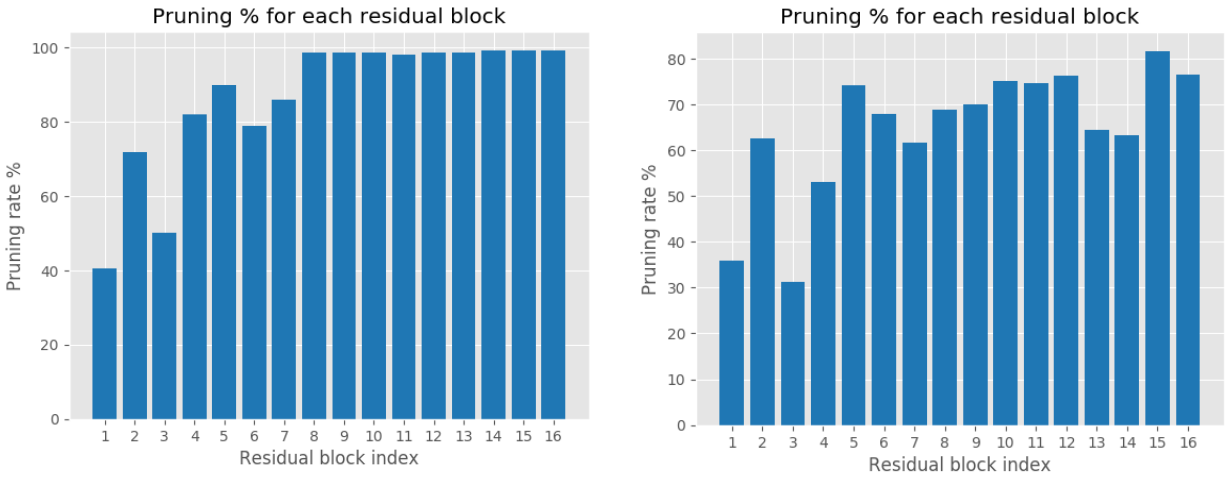


Fig. 9. (left) Shows the pruning allocation for a SBF-ResNet50 model at a pruning of 0.95, while (right) is for a pruning rate of 0.7. To avoid catastrophic over-pruning of layers, a minimum number of basis filters is enforced.