Accelerating Deep Neural Networks via Semi-Structured Activation Sparsity

Matteo Grimaldi Darshan C. Ganji Ivan Lazarevich Sudhakar Sah
Deeplite
matteo.grimaldi@deeplite.ai

Abstract

The demand for efficient processing of deep neural networks (DNNs) on embedded devices is a significant challenge limiting their deployment. Exploiting sparsity in the network's feature maps is one of the ways to reduce its inference latency. It is known that unstructured sparsity results in lower accuracy degradation with respect to structured sparsity but the former needs extensive inference engine changes to get latency benefits. To tackle this challenge, we propose a solution to induce semi-structured activation sparsity exploitable through minor runtime modifications. To attain high speedup levels at inference time, we design a sparse training procedure with awareness of the final position of the activations while computing the General Matrix Multiplication (GEMM). We extensively evaluate the proposed solution across various models for image classification and object detection tasks. Remarkably, our approach yields a speed improvement of $1.25\times$ *with a minimal accuracy drop of* 1.1% *for the ResNet18 model on the ImageNet dataset. Furthermore, when combined with a state-of-the-art structured pruning method, the resulting models provide a good latency-accuracy trade-off, outperforming models that solely employ structured pruning techniques. The code is available at* https://github.com/Deeplite/activ-sparse.

1. Introduction

Deep neural networks (DNNs) have become the go-to state-of-the-art solution in most domains of machine learning in recent years, like computer vision [27], natural language understanding [47] and generative AI [26]. Oftentimes, the computational footprint of DNN models limits their usage on low-resource embedded processors. Compression and acceleration of such models is an active research area aimed at bridging this gap [3] and could be generally categorized into pruning [29, 34, 52], tensor decomposition [33], quantization [7, 41], development of lightweight neural networks [21, 22, 37], and runtime optimizations [2, 15].

Pruning remains a prominent compression method, particularly evidenced by recent strides in structured weight pruning, achieving state-of-the-art latency-accuracy trade-offs across diverse computer vision tasks [8]. However, existing research in pruning has predominantly focused on removing redundant model parameters, overlooking the potential inherent sparsity within feature maps, commonly referred to as activations. Activation sparsity is naturally intrinsic in DNNs with ReLU-like activation functions to a certain extent [31, 44]. Nevertheless, this sparsity, tied to the functional form of the ReLU non-linearity, retains an unstructured nature and lacks homogeneity across layers. Several methods have emerged to artificially augment activation sparsity during training, enhancing model generalization and robustness through regularization techniques [10, 51]. However, such methods selectively remove blocks of connected pixels solely during model training, maintaining denseness at inference time and consequently forfeiting opportunities for model inference acceleration. In contrast, to achieve faster model execution post-training, activation sparsity needs to extend to inference time as well. A variety of works explored *data-dependent* mechanisms to exploit activation sparsity at runtime, dynamically selecting the pixels according to the complexity of the input sample to process [5, 43, 46]. While these approaches efficiently reduce computations with minimal accuracy loss, effectively integrating them into low-power embedded devices can be challenging due to the required architectural modifications. In contrast, *data-free* strategies employ custom regularization with proper hard-thresholding to establish a fixed and constant sparsity pattern [9, 28]. Such a strategy guarantees consistent speedup across distinct input samples. However, the absence of structured regular patterns among zeroed elements confines these model acceleration benefits to dedicated sparse inference engines (e.g., DeepSparse [28]).

To tackle these challenges, we propose an efficient DNN compression pipeline that consists of (i) a novel training scheme that induces semi-structured sparsity in activation feature maps and (ii) an easy-to-implement runtime modification that allows exploiting the semi-structured sparsity of the network's activations at inference time. The proposed sparsity pattern for feature maps is structured in the channel dimension, but unstructured in the spatial dimension. That is,

a set of individual pixels are zeroed across all channels of the feature map. We suggest an effective way to construct such sparsity masks during training and demonstrate how these sparse masks can be used by the runtime during inference. With XNNPACK [13] as an example library, we implement a runtime modification that transforms the semi-structured sparsity of activations into effectively structured sparsity, resulting in reduced computational load through the use of lower ranks in General Matrix Multiplication (GEMM).

To summarize, the primary focus of this study could be outlined as follows:

- We propose a novel training scheme inducing semi-structured activation sparsity in deep neural networks via the propagation of random spatial masks.
- We show that sampling of random masks during training followed by mask freezing improves the performance of DNNs under the constraint of semi-structured sparsity in activations.
- We demonstrate the effectiveness of the proposed training scheme on image classification and object detection tasks and show how it can be combined with structured pruning to get a competitive accuracy-latency trade-off.
- We provide an example of an easy-to-implement runtime modification on top of XNNPACK [13] that allows obtaining latency speedup of up to $2\times$ with relatively low sparsity rates (under 50%).

2. Related Work

Over the past few years, significant progress has been made in the field of deep learning model compression and acceleration, aimed at improving the efficiency of deep neural networks during inference by reducing their memory and computational requirements. Pruning [34, 52] focuses on removing redundant connections or units in the model architecture based on heuristic importance criteria, resulting in streamlined models with improved efficiency. Quantization [24, 37] tackles model size compression by reducing the numerical precision of weights and activations from standard 32-bit floating-point representations to lower bit-widths such as 8-bit, or in more extreme cases, 2-bit or 1-bit. Knowledge distillation [18, 50] involves transferring knowledge from a larger, more complex network to a smaller one, allowing the compact model to attain comparable performance to its larger counterpart. Hand-crafted models, exemplified by architectures like MobileNetV3 [20], EfficientNetV2 [45] and ShuffleNetV2 [36], are often designed with custom operations and blocks optimized for faster inference, thereby enhancing overall efficiency. Furthermore, apart from direct model modifications, there are other strategies aimed at improving the efficiency of deep neural networks. Graph order rewriting involves transforming the network's computational graph to optimize its execution flow, thus enhancing overall performance [1]. Custom runtime optimization [2, 15] aims to maximize model performance at the operator level, harnessing the target hardware's potential. It becomes indispensable in cases where existing operators or processing units cannot directly execute certain model structures, such as unstructured sparse or low-bit quantized models, requiring specific adaptations for seamless and efficient execution.

2.1. Pruning

Pruning methods can be usually categorized according to their granularity [19] or to their importance policy. In terms of granularity, pruning can usually operate with *unstructured* or *structured* sparsity patterns. Unstructured pruning involves removing single connections in the network based on their importance [16, 38]. Targeting individual weights offers flexibility in achieving high accuracy but may lead to challenges in efficient inference due to irregular memory access patterns. A custom runtime with specialized sparse kernels is often necessary to achieve speedup in case of unstructured sparsity (e.g., DeepSparse [23]). Conversely, structured pruning [30, 40] involves the removal of entire channels or filters from the network, which can pose challenges during model training due to its more substantial impact on accuracy. However, pruning at this level of granularity can significantly enhance model efficiency in many existing runtimes, resulting in notable reductions in storage requirements and accelerated inference latency.

Pruning policies encompass various schemes and criteria for efficient model compression. Magnitude-based criteria rely on the absolute weight values to identify less important parameters [16, 35], while first-order methods leverage gradients for importance ranking [4, 39]. Some approaches involve one-time pruning followed by retraining [17], while others adopt iterative pruning techniques [29]. Recent research has explored the efficacy of various pruning methods, offering valuable insights to enhance model compression techniques [48]. Notably, DepGraph [8] introduced a novel method for general structural pruning of arbitrary architectures, efficiently removing coupled parameters for model acceleration. The results demonstrate its superior performance compared to many other techniques.

2.2. Activation Sparsity

Another crucial sphere of inquiry revolves around exploiting the inherent sparsity present within neural network feature maps, particularly in the context of computer vision applications. The induction of activation sparsity stands out as a pivotal technique for latency reduction, providing a synergistic complement to weight pruning strategies. Sparsity is naturally present in feature maps due to the presence of ReLU-like activation functions which force feature maps

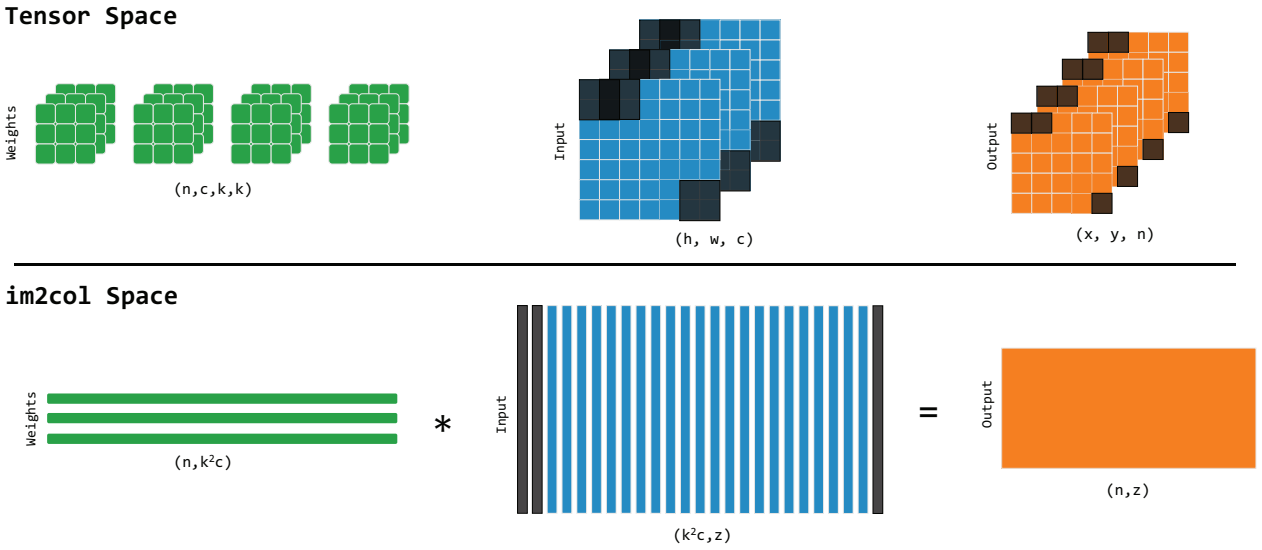

Figure 1. Scheme of induced semi-structured activation sparsity with the proposed sparsity pattern in the in tensor vs. `im2col` spaces.

to become zero when their values fall below certain thresholds [28, 31].

The majority of efforts in the literature have been directed towards harnessing activation sparsity through *data-dependent* mechanisms, tightly linked to input complexity. This strategy entails an informed masking approach, where the sparsity pattern is dynamically generated based on the distribution of less informative pixels within the input samples. Consequently, a distinct sparsity pattern is generated for each input. Some of these techniques necessitate architectural adjustments for on-the-fly pattern generation at runtime [5, 43, 46]. Unfortunately, these requirements significantly hamper their effectiveness when deployed on resource-constrained devices. As a result of these constraints, many of these works often lack real-world hardware validation or predominantly demonstrate latency improvements on higher-performance hardware configurations. For instance, the efficacy of sparsity has been pronounced in GPU deployment scenarios, yielding impressive latency enhancements such as up to $1.88\times$ acceleration on a ResNet50 architecture using a *Mali* GPU [42]. Similarly, the work by Xu et. al [49] tailored custom kernels for Nvidia GPUs, resulting in performance acceleration of 3-4$\times$.

In more recent investigations, novel regularization strategies have emerged to induce activation sparsity featuring a regular and consistent pattern, regardless of varying input samples (*data-free* strategies). Georgiadis et. al [9] proposed to combine sparsity, quantization, and entropy encoding of activation maps to achieve up to $1.6\times$ inference acceleration and up to $6\times$ reduction of the memory footprint for architectures like InceptionV3 and MobileNetV1. Kurtz et al. [28] introduced a new regularization technique and threshold-based sparsification based on a parameterized activation function to maximize sparsity with minimal accuracy drop. While these works are the most similar to our approach, they predominantly emphasize unstructured sparsity among zeroed elements. As a consequence, these model acceleration benefits remain confined to dedicated sparse inference engines like DeepSparse [28].

2.3. Low-Rank GEMM

The widely adopted `im2col`-based General Matrix Multiply (GEMM) technique converts feature maps into column-wise matrices. This transformation paves the way for streamlined matrix multiplication with weight matrices, thus fostering parallel computations and refining the convolutional operations. Moreover, the low-rank GEMM approach focuses on reducing the number of rows (or columns) in one of the two matrices, aiming to decrease computational complexity and memory demands. Dong et al. [5] devised a trainable module learning collaborative kernels to selectively skip activation pixels during computation, yielding a $1.2\times$ speedup. Their analysis focused on two models and relatively simple datasets. In the context of video processing, the Skip-conv network [14] leverages residual images, creating sparsity exploited by low-rank GEMM. This approach suits moving objects, producing notable sparsity. Liu et al. [32] applied sparse adaptive inference for super-resolution, more similar to our approach, but just tailored to low-rank GEMM for specific patches crucial in super-resolution tasks.

3. Methodology

GEMM-based implementation of the convolution operation is typically favored over the direct one as GEMM enables faster and more efficient matrix operations, making

it a preferred choice for deep learning inference engines. Reducing the rank of the matrices in GEMM operations is generally directly correlated with faster computation, especially on low-power CPUs. Our proposed technique aims to reduce the rank of the input activation matrix (activation feature map in the `im2col` space) to speed up model inference. This is pursued by inducing semi-structured sparsity in the network at training time which will be exploited through lower-rank GEMMs at inference time.

Figure 1 shows the convolution-as-GEMM implementation for convolutional layers, where both weights (green) and activations (blue) are unfolded respectively from 4-D and 3-D tensors to 2-D matrices. The picture shows the standard convolution operation both in the tensor space (i.e., the standard space before the reshaping) and in the `im2col` space. Each of the n filters is reshaped into a row of k^2c size, where k is the kernel size and c is the number of channels. In the same way, the input feature map is reshaped into a $k^2c \times z$ matrix, where each column is composed of all the pixels of the input sliding window (k^2c). The number of rows z depends on the convolution parameters (e.g., stride, padding, and dilation values). Then a standard matrix multiplication of weights and activation matrices is computed to generate an $n \times z$ output matrix.

In order to reduce the rank of the activation matrix, a subset $s < z$ of columns needs to be removed. These columns correspond to elements covered by the sliding local tiles (covering all channels) used during the convolution in the tensor space. To remove the columns at compute time, during each convolution, a subset s of the sliding local tiles needs to be skipped: a binary mask with a `im2col`-based pattern is used to apply hard thresholding to the activation tensors, where the s sparse columns of the activation matrix will be directly skipped during inference. In the two following subsections, we show how to induce (at training time) and how to exploit (at inference time) such semi-structured activation sparsity.

3.1. Training

To induce activation sparsity with the `im2col` pattern, we need to group activations in the tensor space according to their final position after the `im2col` reshaping. We consider this approach as semi-structured as it is unstructured in the $width \times height$ space (spatial dimensions of the feature map) but it is structured across the channel dimension.

Pruning activations with this pattern is a more delicate procedure compared to standard unstructured weight pruning, as the elements of the activation feature map cannot be directly removed from the model. The sparsified elements in the activations for one convolutional window/tile (i.e., one `im2col` column) could be kept dense (unmasked) for the next windows/tiles. Figure 2 demonstrates this concept for a case when a single window (tile) is selected to be sparsified

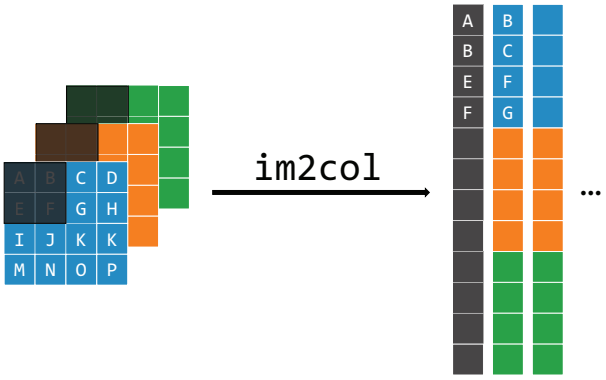

Figure 2. Example of the `im2col` procedure: input activations (left) and the activation matrix after transformation (right). Note that masking (highlighted in black) a sliding tile of the convolution affects only a single column in the reshaped matrix. In the first column, pixels B and F are masked, while they remain non-zero in the second column.

(masked). In this case, the pixels $\{A, B, C, D\}$ are dropped from the computation (including all the pixels/elements with the same $(width, height)$ coordinates in the other channels). This results in the first column of the `im2col` matrix becoming zero, which reduces the rank of the matrices to be multiplied. However, dropping (masking) this block from the feature map altogether should also affect the second column of the matrix, which is not selected to be pruned. For this reason, the pixels B and F will be masked for the first column but will be kept non-zero in the second one.

Introducing activation sparsity in deep neural networks for computer vision is challenging due to the varying positions of the regions of interest in images. Uniformly enforcing sparsity with a fixed pattern across data samples can lead to information loss for some images and retention for others. Achievable sparsity levels (while keeping accuracy degradation low) are often limited compared to weight sparsity, due to the dynamic and context-dependent nature of activation patterns in different input images. It has been shown that inducing structured sparsity through sampling random masks [10] can act as a regularizer that enhances the model's generalization and robustness. We found sampling random masks during training can reduce the accuracy loss when the sparsity rates are kept relatively low. The random ranking mechanism ensures that the selection of pixels to be masked is unbiased, contributing to the robustness of the training process. We propose a novel custom random masking approach, which involves randomly selecting a percentage of pixels from the input image to be masked. The resulting input image mask is then propagated consistently across all layers (employing pooling operations when downsampling is necessary). By propagating this initial random sparse pattern layer-to-layer, we ensure the preservation of the same

masking structure throughout the network. This guarantees translation invariance across the feature maps of different layers, even when they have varying resolutions. The proposed custom random mask sampling is a crucial aspect of our training procedure as it helps the model to prevent overfitting to specific patterns and encourage more generalized learning, yet limiting accuracy loss. The generated binary masks, specific to each sparsity level, enable the model to adapt its weights during training, effectively promoting the benefits of sparsity while maintaining crucial representational capacity. The training process comprises three key stages: (i) initially, a few dense pretraining epochs are performed; (ii) subsequently, our masking technique is applied gradually according to a schedule, incrementing sparsity rate until the desired target [52] is achieved; (iii) finally, the mask freezing stage ensues, where binary masks for each layer are fixed for the rest of the training process, allowing the model to recover from accuracy loss through more precise updates.

Algorithm 1 outlines our sparse training pipeline. The algorithm takes the fixed sparsity percentage s as an input and returns the trained model with a binary constant mask $mask$. The pruning scheduler (line 3) controls the switch between dense (line 8) and sparse forward steps (line 6). The `updateMask` (line 4) scheduler sets when to update or freeze the masks through the `getMask` function (line 5). This mask is used by `maskedForward` to induce the sparsity in the feature maps. At the end of the training, both the model and the masks are returned (line 11). It needs to be highlighted that model weights are kept fully dense, and no weights are pruned. The `getMask` function plays a critical role in our sparse training pipeline, as it is responsible for generating a different binary mask for each forward step. At first, a random 2-D score is generated according to the input image resolution (line 13). This is propagated through the layers, downscaling the resolution when needed (lines 15-16). At last, the function ranks the model's score and generates the binary mask (lines 17-19).

Algorithm 1: Sparse Training

```
1  Function main (model, steps, s) :
2      for t in steps do
3          if pruneStep (t) then
4              if updateMask (t) then
5                  mask = getMask (model, s)
6              maskedForward (model, mask)
7          else
8              forward (model)
9          backward (model)
10     end
11     return model, mask
12 Function getMask (model, s) :
13     score = randomScore2d (model.input_res)
14     for layer in model do
15         ratio = input_res // layer.res
16         layer_score = avg_pool2d (score, ratio)
17         idx = rankPixels (layer_score)
18         mask = ones_like (model)
19         mask[idx] = 0
20     end
21     return mask
22 return
```

3.2. Inference

To accelerate the processing of the models with sparse activation maps, we implemented custom modifications to the XNNPACK [13] inference engine. We used TensorFlow lite (TFLite) [12] built from source with XNNPACK [13] as a delegate. Given a TFLite model, a binary mask, and layer-wise sparsity levels as inputs, our inference engine computes the convolution of sparse activations. Our modifications are specific to convolutional layers only. The full pipeline consists of three main stages: (i) custom `im2col` reshaping, (ii) dense GEMM, and (iii) custom post-processing of the dense GEMM output.

The first step consists of reshaping the tensors into a 2-D matrix for activations, as shown in Fig. 1. Considering that the XNNPACK [13] `im2col` routine is based on an indirection buffer [6], we developed a custom transformation to facilitate the skipping of rows of an indirection matrix. After this is done, the compute range of the GEMM is downsized to $output\_size - (sparsity * output\_size)$ to enable a low-rank GEMM in the following step. In the second stage, standard GEMM is employed, utilizing a low-rank matrix of activations. However, the subsequent layer assumes dense activation, necessitating an efficient post-processing stage. In this implementation, zeroed elements are inserted into the GEMM output based on the binary masks used in the initial stage. These modifications follow a consistent pattern across different inference engines, all designed to work with commonly used general-purpose processors. For more detailed information on the runtime modifications, please refer to Appendix A.

4. Results

4.1. Training Setup

The proposed pipeline was validated on several image classification and object detection datasets, including CIFAR100, Flowers102, Food101, and ImageNet for classification and PASCAL VOC and Global Wheat for object detection (further details in Appendix B). We have performed experiments on ResNet18, ResNet50, and MobileNetV2 architectures for the image classification task, and used YOLOv5n [25] as a base architecture for the object detection experiments. Note that a few of the base architectures we

	Sparsity	ResNet18	ResNet50	MobileNetV2
Flowers102	0%	92.00	92.50	92.57
	10%	91.20 (-0.80)	91.80 (-0.70)	91.46 (-1.11)
	20%	90.25 (-1.75)	91.02 (-1.48)	90.11 (-2.46)
	30%	88.89 (-3.22)	90.13 (-2.37)	88.52 (-4.05)
Food101	0%	82.20	86.17	77.20
	10%	81.07 (-1.13)	85.10 (-1.07)	82.35 (-1.77)
	10%	80.27 (-1.93)	84.10 (-2.07)	81.04 (-1.32)
	30%	78.59 (-3.61)	82.40 (-3.77)	79.32 (-4.80)
CIFAR100	0%	77.20	78.00	73.10
	30%	76.37 (-0.83)	77.26 (-0.74)	71.30 (-1.80)
	30%	75.30 (-1.90)	75.80 (-2.20)	70.57 (-2.53)
	30%	74.11 (-3.09)	74.78 (-3.22)	68.60 (-4.50)

Table 1. Top-1 accuracy result (%) for different architectures on Flowers102, Food101, and CIFAR100 datasets. The relative inference speedups are reported in Fig. 3.

Sparsity	ResNet18	MobileNetV2
0%	70.53	72.19
10%	70.48 (-0.05)	70.43 (-1.76)
20%	69.42 (-1.11)	69.94 (-2.25)
30%	67.88 (-2.65)	67.92 (-4.27)

Table 2. Top-1 accuracy results for different architectures on Imagenet dataset. The relative inference speedups are reported in Fig. 3.

Sparsity	VOC	Global Wheat
0%	80.20	96.38
10%	78.08 (-2.12)	96.00 (-0.38)
20%	76.63 (-3.57)	95.49 (-0.89)
30%	74.13 (-6.07)	94.80 (-1.58)

Table 3. mAP_{50} results for YOLOv5n on VOC and Globat Wheat datasets. The relative inference speedups are reported in Fig. 3.

used (e.g., MobileNetV2, YOLOv5n) were initially designed as lightweight efficient architectures, which makes it more challenging to obtain competitive latency speedup with low accuracy degradation.

For image classification, we used the training code provided by Ultralytics [25] with default values of hyperparameters except for the number of epochs (Adam optimizer, initial learning rate 10^{-4}, 400 epochs, batch size 64). ImageNet pre-trained weights were used for model initialization for both the dense baseline as well as for sparse training. We set the dense training stage to stop at 10% of the training steps and the freezing stage to start at 90% of the steps. For object detection experiments, the training code provided by Ultralytics [25] was also used with default values of hyperparameters. COCO pre-trained weights were used to initialize the models both for the dense baseline as well as for sparse training.

4.2. Sparse Model Deployment

The latency speedup from using semi-structured activation sparsity was measured on a Raspberry Pi 4B [11] device, featuring a quad-core ARM Cortex-A72 processor operating at 1.5GHz, with 4GB of RAM. We ran Ubuntu 18.04 64-bit OS on this platform and GNU gcc version 11.0 for compilation. For deployment, we used TFLite [12] inference engine built with XNNPACK [13] delegate with custom modifications for sparse inference.

4.3. Sparse vs. Dense Model Performance

In this section, we evaluate the efficacy of the semi-structured activation sparsity approach for enhancing DNN speed, prioritizing high-speed improvements at the expense of marginal accuracy degradation.

4.3.1 Low Accuracy Loss Regime

Using the same sparse training procedure, we induced the activation sparsity at three different levels $S = \{10\%, 20\%, 30\%\}$. Table 1 shows that the accuracy loss is low (under 2.5%) for the first two sparsity rate levels in image classification tasks, while it is close to 3% for the highest sparsity rate chosen (30%) depending on the architecture. ResNet models are found to be more resilient to activation sparsity compared to MobileNetV2, in fact, they have an average 1.81% of accuracy loss instead of 2.72% for MobileNetV2. On the more challenging ImageNet dataset (Table 2), ResNet18 at 10% sparsity rate provides almost the same accuracy (-0.05%) as the dense counterpart. For clarity, we included further details on the training procedure in Appendix B. To evaluate the generalization capabilities of our proposed compression pipeline, we carried out experiments for the object detection task using the YOLOv5n model. The obtained results on VOC and Global Wheat datasets are summarized in Table 3, showcasing the impact of compression on accuracy. Notably, results for object detection appear to be comparable to those of image classification, with limited mAP_{50} degradation on a simpler dataset (Global Wheat) and higher accuracy loss observed on a more large-scale task (VOC). These findings highlight the effectiveness of our compression techniques in preserving model accuracy across different tasks.

4.3.2 High Speedup Regime

In our findings, we observe a consistent trend where activation sparsity contributes to notable and reliable speed improvements throughout the network layers, with the magnitude of the speedup roughly proportional to the degree of activation sparsity achieved. To visually depict and quantify

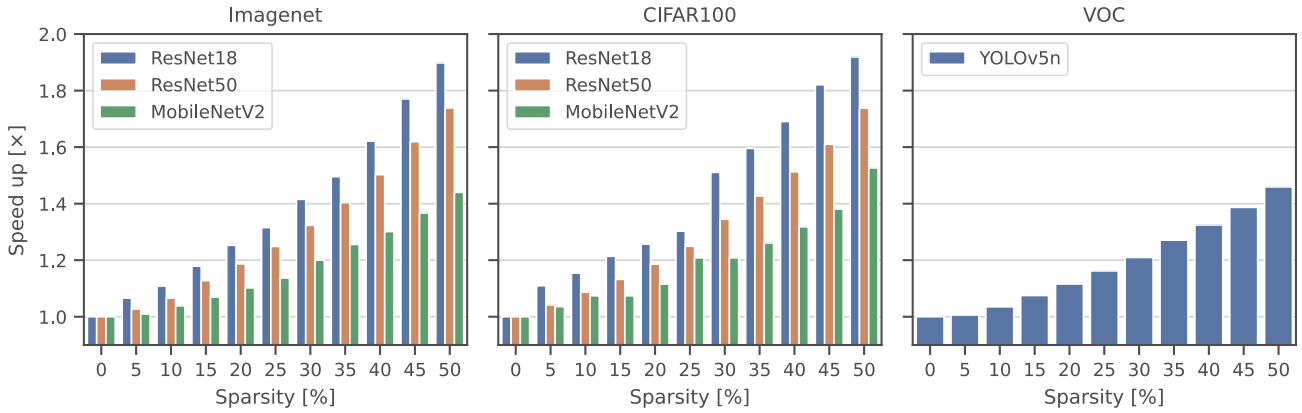

Figure 3. Speed-up vs. sparsity rate for ImageNet, CIFAR100, and VOC datasets on different architectures. Flowers102 and Food101 speed-up results are identical to those of ImageNet.

these results, we present Fig. 3, which illustrates the end-to-end speedup outcomes for four distinct models: ResNet18, ResNet50, MobileNetV2, and YOLOv5n.

ResNet18 exhibits a nearly linear relationship between the sparsity percentage and the speedup for all the sparsity levels. For, ResNet50, MobileNetV2, and YOLOv5n, due to the larger amount of layers and complexity, experience a slightly diminished speedup when compared to ResNet18. This slight reduction in speedup can be attributed to the presence of additional steps that involve custom `im2col` and post-processing transformations, which offset the gains obtained from reduced GEMM computations. For ResNet50, the speedup achieved is approximately $1.75\times$, while MobileNetV2 and YOLOv5n attain speedups of around $1.44\times$ and $1.46\times$, respectively, all based on 50% sparsity.

In summary, our findings indicate that activation sparsity within the network layers leads to consistent and significant improvements in inference latency. The overall trend suggests that activation sparsity offers a valuable approach to enhancing the efficiency of deep learning models across a variety of architectures.

4.4. Ablation Study

To comprehensively evaluate the efficacy of our proposed sparse training scheme, we conducted two ablation studies focusing on the custom features involved to reduce accuracy loss: mask propagation and mask freezing. For both studies, we trained ResNet18 on the Flowers102 dataset using the same hyperparameters described in the Subsection 4.1.

Mask Propagation Figure 4 depicts the comparison of accuracy and sparsity achieved by the ResNet18 model with and without mask propagation. The plot clearly demonstrates the advantages of employing the mask propagation method, revealing a significant improvement in the model's resilience to sparsification. The use of mask propagation provides up to 1.28% of accuracy boost at 30% sparsity rate and an average of 0.83% for the three tested sparsity levels.

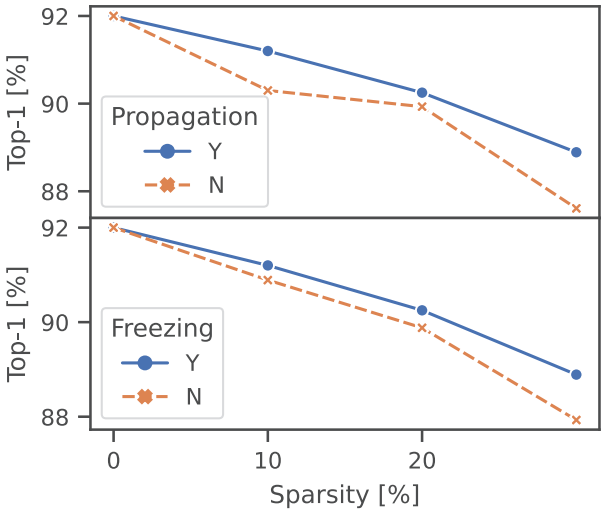

Figure 4. Ablation results for mask propagation and mask freezing for ResNet18 on Flowers102 dataset.

Mask Freezing The mask freezing approach ensures that the binary masks used for sparsity remain fixed during the last training epochs, thereby allowing the model to recover from accuracy loss more effectively with precise updates. This mechanism, widely used in literature [52], is crucial for our training scheme where the masks are randomly changed after each step. Figure 4 shows the clear advantage of integrating the mask freezing method into the training process: the model trained with mask freezing showcases up to 0.96% higher accuracy than the one without.

4.5. Weight Pruning vs. Activation Sparsity

In this section, we conduct a comprehensive comparison of our activation sparsity method with a state-of-the-art

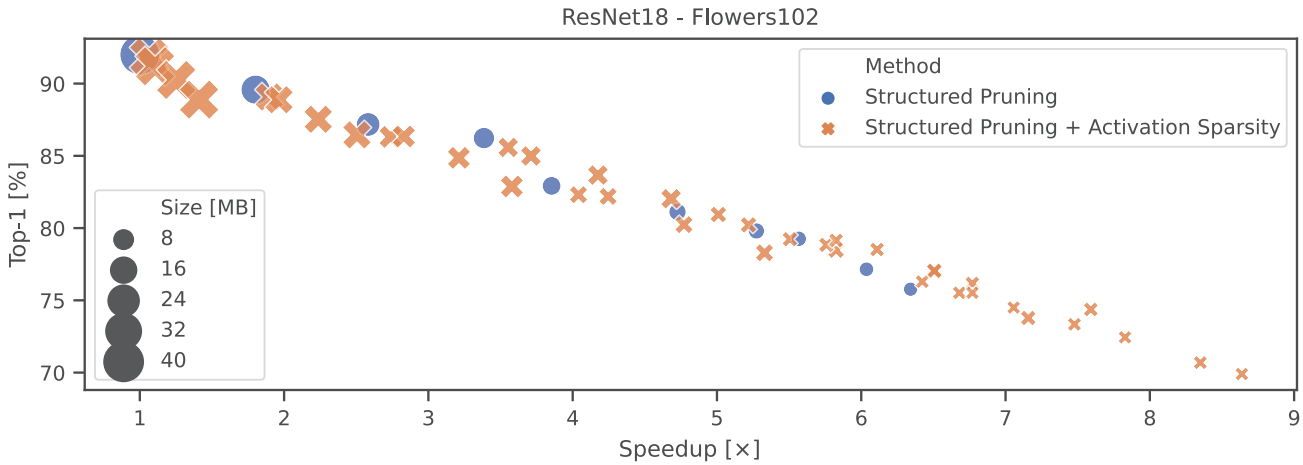

Figure 5. Latency-accuracy trade-off distribution for structured weight pruning with and without activation sparsity (ResNet18, Flowers102). A detailed table with all the numerical values is available in Appendix B.

structured weight pruning technique represented by DepGraph [8]. By utilizing DepGraph as a robust baseline, we aim to thoroughly assess the effectiveness and potential of our activation sparsity approach in comparison to leading compression techniques. While the work by Kurtz et al. [28] appears conceptually aligned with our approach, we refrain from direct comparison due to the need for a custom sparse kernel to achieve the desired latency boost. Moreover, their research primarily focuses on higher-performance platforms, such as AWS C5.12xlarge CPU and NVIDIA K80 GPUs, rather than exploring embedded CPUs, limiting the scope of direct comparison with our solution.

Since structured weight pruning and activation sparsity can be applied independently, we decided to apply activation sparsity on models pruned using DepGraph to see the impact on performance. Figure 5 depicts the latency vs. accuracy trade-off achievable by structured pruning with and without our proposed activation sparsity technique. We performed these experiments on ResNet18 with the Flowers102 dataset. The pruned models were obtained using the original codebase provided by DepGraph authors with different values of the speedup proxy parameter (MACs count ratio) from $2.0\times$ to $10.0\times$ [8]. Then, we induced activation sparsity in the pruned models for four different sparsity levels (5%, 10%, 20%, 30%), using the Ultralytics training code for image classification [25]. The same training code was also used to further finetune the pruned models (without sparsity) for fair comparison. The experimental results show that while the solely structured pruning is Pareto optimal for lower speedup rates, a combination of both techniques becomes more favorable for beyond $3.5\times$ speedup. Furthermore, while structured pruning offers high scaling ability, activation sparsity acts as a fine-grained control knob in the accuracy vs. latency solution space. Latency measurement experiments carried out on the Raspberry Pi 4B [11] showcase a significant difference between the real and theoretical speedup of pruned models. A detailed table with all the different speedups is available in Appendix B.

Activation sparsity applied to pruned models shows notable performance improvements, especially for high pruning ratios. This behavior can be attributed to the understanding that models pruned beyond a certain limit may experience reduced capacity and subsequently degraded performance. In such cases, activation sparsity proves to be an effective approach by capitalizing on zeros in the activation maps, which remain independent of the model's capacity, leading to optimal results.

5. Conclusion

This paper presents an efficient DNN compression pipeline leveraging semi-structured activation sparsity to reduce inference latency. The proposed training procedure induces activation sparsity through the propagation and freezing of random spatial masks, being cognizant of element positions during GEMM-based convolutions. Additionally, we provide an illustrative example of a practical runtime modification integrated into XNNPACK to measure latency speedup on a Raspberry Pi 4B device. Our experimental results showcase the impact of activation sparsity on accuracy and speedup across diverse test cases encompassing image classification and object detection tasks. Furthermore, we demonstrate the potential to combine our compression pipeline with other structured pruning algorithms, offering enhanced accuracy-speed trade-offs, especially for high compression ratios. In future work, we plan to explore advanced regularization techniques to determine optimal sparsity levels across layers.

References

[1] Byung Hoon Ahn, Jinwon Lee, Jamie Menjay Lin, Hsin-Pai Cheng, Jilei Hou, and Hadi Esmaeilzadeh. Ordering chaos: Memory-aware scheduling of irregularly wired neural networks for edge devices. *Proceedings of Machine Learning and Systems*, 2:44–57, 2020. 2

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: end-to-end optimization stack for deep learning. *arXiv preprint arXiv:1802.04799*, 11(20), 2018. 1, 2

[3] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020. 1

[4] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in neural information processing systems*, 30, 2017. 2

[5] 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. 1, 3

[6] Marat Dukhan. The indirect convolution algorithm. *arXiv preprint arXiv:1907.02129*, 2019. 5

[7] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2019. 1

[8] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16091–16101, 2023. 1, 2, 8

[9] Georgios Georgiadis. Accelerating convolutional neural networks via activation map compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7085–7095, 2019. 1, 3

[10] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*, 31, 2018. 1, 4

[11] Google. Raspberry pi. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, 2023. 6, 8

[12] Google. Tflite. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite, 2023. 5, 6

[13] Google. Xnnpack. https://github.com/google/XNNPACK, 2023. 2, 5, 6

[14] Amirhossein Habibian, Davide Abati, Taco S Cohen, and Babak Ehteshami Bejnordi. Skip-convolutions for efficient video processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2695–2704, 2021. 3

[15] 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. 1, 2

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 2

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[19] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021. 2

[20] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019. 2

[21] 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. 1

[22] 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. 1

[23] Eugenia Iofinova, Alexandra Peste, Mark Kurtz, and Dan Alistarh. How well do sparse imagenet models transfer? *CoRR*, abs/2111.13445, 2021. 2

[24] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018. 2

[25] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, Jan. 2023. 5, 6, 8

[26] Anis Koubaa. Gpt-4 vs. gpt-3.5: A concise showdown. 2023. 1

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[28] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In *International Conference on Machine Learning*, pages 5533–5543. PMLR, 2020. 1, 3, 8

[29] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 2

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 2

[31] Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*, 2022. 1, 3

[32] Ming Liu, Zhilu Zhang, Liya Hou, Wangmeng Zuo, and Lei Zhang. Deep adaptive inference networks for single image super-resolution. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 131–148. Springer, 2020. 3

[33] Ye Liu and Michael K Ng. Deep neural network compression by tucker decomposition with nonlinear response. *Knowledge-Based Systems*, 241:108171, 2022. 1

[34] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017. 1, 2

[35] 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. 2

[36] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018. 2

[37] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: Wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017. 1, 2

[38] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017. 2

[39] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272, 2019. 2

[40] 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. 2

[41] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021. 1

[42] Chanyoung Oh, Junhyuk So, Sumin Kim, and Youngmin Yi. Exploiting activation sparsity for fast cnn inference on mobile gpus. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–25, 2021. 3

[43] 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. 1, 3

[44] Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, Youngeun Kwon, and Stephen W Keckler. Compressing dma engine: Leveraging activation sparsity for training deep neural networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 78–91. IEEE, 2018. 1

[45] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021. 2

[46] Chen Tang, Wenyu Sun, Zhuqing Yuan, and Yongpan Liu. Adaptive pixel-wise structured sparse network for efficient cnns. *arXiv preprint arXiv:2010.11083*, 2020. 1, 3

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1

[48] Huan Wang, Can Qin, Yue Bai, and Yun Fu. Why is the state of neural network pruning so confusing? on the fairness, comparison setup, and trainability in network pruning. *arXiv preprint arXiv:2301.05219*, 2023. 2

[49] Weizhi Xu, Yintai Sun, Shengyu Fan, Hui Yu, and Xin Fu. Accelerating convolutional neural network by exploiting sparsity on gpus. *ACM Transactions on Architecture and Code Optimization*, 2019. 3

[50] Xinchuan Zeng and Tony R. Martinez. Using a neural network to approximate an ensemble of classifiers. *Neural Processing Letters*, 12:225–237, 2000. 2

[51] Yiren Zhao, Oluwatomisin Dada, Xitong Gao, and Robert D Mullins. Revisiting structured dropout. *arXiv preprint arXiv:2210.02570*, 2022. 1

[52] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 1, 2, 5, 7