Accelerating Deep Neural Networks via Semi-Structured Activation Sparsity Supplemental Material

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

A. Inference Engine Modification

In the context of runtime modifications for activation sparsity inference, we used XNNPACK [6] as our inference engine and made minor adaptations to ensure robust support for inference. Algorithm 1 illustrates a simplified pseudocode representation of these crucial modifications. Additional information about these referenced functions is readily available within the code repository [6]. The implementation comprises three stages: (i) a custom indirection-based `im2col`, (ii) a standard dense GEMM, and (iii) custom post-processing components.

At first, the `xnn_indirection_init_conv2d_sparse` (lines 1-11) function illustrates our custom approach for efficiently skipping rows within an indirection matrix. Within the indirection-based `im2col` function, we deviate from memory-intensive transformations and instead, store input value pointers in the indirection buffer. This strategy adheres to the loop structure commonly employed in the standard `im2col` transformation. Diverging from the original procedure, our implementation skips converting the entire convolution patch into a single row when mask values equate to 0 (line 5). To accurately allocate the appropriate input value pointer to a designated position (`index`) within the `indirection_buffer`, we employed the `base_address` and `offset` variables. These variables are computed according to the `conv2d` parameters. Upon completing this initial step, the computation range of the GEMM is reduced to $output\_size - (sparsity \times output\_size)$. The GEMM function operates as a subroutine that efficiently conducts dense matrix multiplication between weight and activation values. This function remains unaltered, with no modifications made to the underlying kernel.

Lastly, the `post_process_conv2d_sparse` function (lines 12-26) manages the output for the subsequent layer, incorporating a transformation that involves the insertion of zeros based on the corresponding mask value. When the mask value is 0 (lines 16-18), the function inserts zeros. Alternatively, when the mask value is 1 (lines 19-21), data is copied from one position to another within the same output channel size, denoted as `outch_size` in the algorithm. This customized procedure is tailored for post-processing the output subsequent to low-rank GEMM operations, and it is invoked within the `xnn_run_operator` method [6].

Algorithm 1: Inference Engine Modification

```
1  Function xnn_indirection_init_conv2d_sparse:
2      indirection_buffer ← empty list;
3      for out_y to output_height do
4          for out_x to output_width do
5              if mask[out_x][out_y] == 1 then
6                  indirection_buffer[index] ← (const void*)
                     ((uintptr_t) input + base_address +
                     offset);
7              end
8          end
9      end
10     return indirection_buffer;
11 end

12 Function post_process_conv2d_sparse:
13     outch_size = output_channels * sizeof(float);
14     for out_y → output_height - 1 to 0 do
15         for out_x → output_width - 1 to 0 do
16             if mask[out_x][out_y] == 0 then
17                 memset(op→ output[out_y * op→
                     output_height + out_x], 0, outch_size);
18             end
19             else
20                 memcpy(output[out_y * out-
                     put_height + out_x], output[id],
                     outch_size);
21                 id ← id - 1;
22             end
23         end
24     end
25     return output;
26 end
```

B. Additional Details on Experiments

B.1. Datasets

CIFAR-100 [8] : It comprises $60,000$ RGB images, each measuring 32×32 pixels, and annotated with 100 distinct labels with $45,000$ training, $5,000$ validation, and $10,000$ testing samples.

Flowers102 [9]: This dataset is a collection of 102 categories of flower species, with each category containing a variable number of RGB images. Each image is of arbitrary size and comes with appropriate labels indicating the corresponding flower species. We used 224×224 image resolution.

Food101 [1]: It comprises a diverse set of food images spanning 101 distinct classes, the dataset offers a valuable resource for food recognition tasks. Each RGB image in the dataset is associated with a specific food category. We used 224×224 image resolution.

ImageNet [3]: This dataset comprises 1M of RGB images belonging to a vast array of classes, enabling in-depth evaluation of image classification capabilities. The pipeline leveraged subsets of the ImageNet dataset, ensuring a representative and diverse range of images for training, validation, and testing purposes.

PASCAL VOC [4]: The PASCAL VOC dataset, derived from the PASCAL Visual Object Classes Challenge, encompasses $15,870$ RGB images with 37,813 object annotations for 20 different categories. The pipeline adhered to the recommended approach outlined in, utilizing the VOC07 and VOC12 trainval data for training, while the VOC07 dataset was employed for testing purposes. We used 480×480 image resolution.

Global Wheat [2]: The Global Wheat Head Dataset is a collection of images designed to support the development of accurate wheat head detection models for applications in wheat phenotyping and crop management. The dataset contains over 3000 images in the training set, and approximately 1000 images for validation taken in different regions. We train and evaluate with 480×480 image resolution in our experiments.

B.2. Training

Figures 1 and 2 report an example of the training curves to offer comprehensive insights into the proposed method's learning behavior, providing a deeper understanding of the training dynamics and overall training performance.

References

[1] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 446–461. Springer, 2014. 2

[2] Etienne David, Simon Madec, Pouria Sadeghi-Tehran, Helge Aasen, Bangyou Zheng, Shouyang Liu, Norbert Kirchgessner, Goro Ishikawa, Koichi Nagasawa, Minhajul A Badhon, et al. Global wheat head detection (gwhd) dataset: a large and diverse dataset of high-resolution rgb-labelled images to develop and benchmark wheat head detection methods. *Plant Phenomics*, 2020. 2

[3] 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. 2

[4] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015. 2

[5] 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. 4

[6] Google. Xnnpack. https://github.com/google/XNNPACK, 2023. 1

[7] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, Jan. 2023. 4

[8] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[9] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pages 722–729. IEEE, 2008. 2

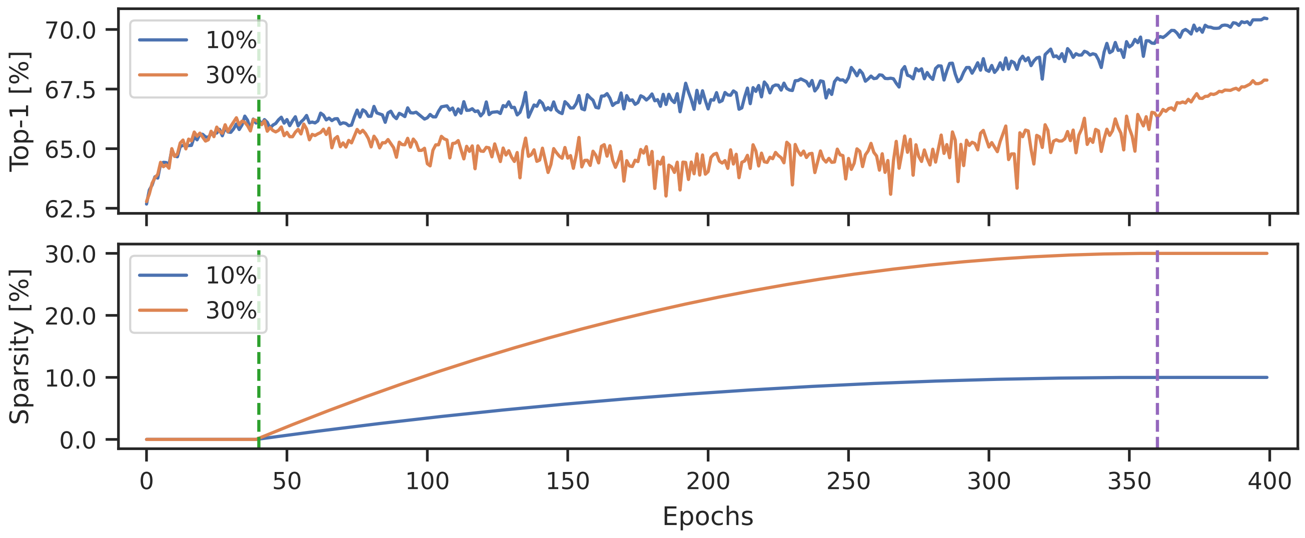

Figure 1. Training curves for ResNet18 on the ImageNet dataset for two different sparsity levels. The two vertical lines split the training curve according to the three different stages. From epoch 0 to epoch 40 (green line) the dense pretraining steps, from epoch 40 to epoch 360 (purple line) the sparse training steps with variable random masking, and, at last, from epoch 360 to the end the mask freezing stage.

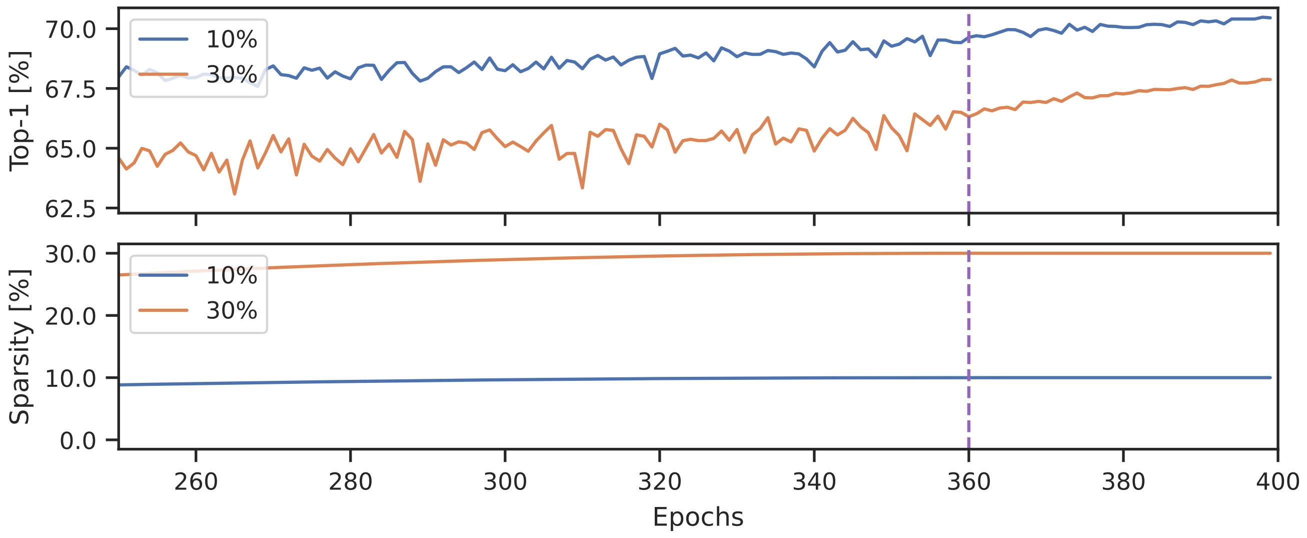

Figure 2. Training curves for ResNet18 on the ImageNet dataset for two different sparsity levels. The same training curves of Fig. 1, here zoomed in on the last epochs to better show the effects of the mask freezing stage (from epoch 360 to 400).

Structured Weight Pruning				Structured Weight Pruning + Activation Sparsity							
Depgraph [5]		**Fine-tuned** [7]		**5%**		**10%**		**20%**		**30%**	
S	A	S	A	OS	A	OS	A	OS	A	OS	A
1.0	/	1.0	92.02	1.07	91.8	1.11	90.06	1.25	89.50	1.41	88.60
2.0	89.46	1.8	89.58	1.90	89.07	1.96	88.88	2.24	87.53	2.51	86.45
3.0	86.27	2.6	87.17	2.74	86.31	2.83	86.34	3.21	84.86	3.58	82.88
4.0	85.18	3.4	86.23	3.55	85.58	3.71	84.99	4.18	83.67	4.68	82.03
5.0	81.93	3.9	82.92	4.04	82.31	4.25	82.19	4.77	80.24	5.33	78.29
6.0	79.87	4.7	81.12	5.01	80.94	5.22	80.22	5.83	78.47	6.51	77.01
7.0	79.44	5.3	79.80	5.51	79.23	5.76	78.84	6.51	77.05	7.16	73.78
8.0	78.27	5.6	79.26	5.83	79.15	6.11	78.52	6.77	76.17	7.59	74.37
9.0	76.01	6.0	77.15	6.42	76.29	6.68	75.52	7.48	73.34	8.35	70.69
10.0	74.65	6.3	75.77	6.68	75.52	6.77	74.50	7.26	72.44	7.59	69.90

Table 1. Latency-accuracy results for structured pruning without (first four columns) and with activation sparsity (last eight columns) for ResNet18 on the Flowers102 dataset. For each pair of structured pruning columns, we report speedup (S, $\times$) and top-1 accuracy (A, %). The first group shows the results obtained using the original training code of Depgraph [5] with the estimated speedups, while the second one the results obtained with further fine-tuning using Ultralytics training code [7] with the real speedups measured on the device. For each pair of columns of structured pruning with activation sparsity, we report overall speedup (OS, $\times$) and top-1 accuracy (A, %) at different levels of sparsity, trained using the same Ultralytics training code [7].