FLAR-SVD: Fast and Latency-Aware Singular Value Decomposition for Model Compression

Supplementary Material

6. Downstream Tasks

We investigate the applicability of our compressed models on downstream tasks. Specifically, we use our compressed version of Swin-L [16] and ConvNeXt-B [17] (both without any finetuning after compressing) as backbones for object detection models. We perform instance segmentation using Mask R-CNN [11]. The experiments are conducted on the COCO dataset [15], a standard benchmark for evaluating detection and segmentation models. All models are trained using a 1x schedule, which consists of 12 epochs, following the training protocol outlined in the official mmdetection Swin configs. The training is using the mmdetection [4] frameworks and configs. We further evaluate our models on segmentation tasks on the ADE20K [30] dataset using the mmsegmentation [5] toolbox. We put a upernet [26] head to our models and follow the schedule outlined for Swin and ConvNeXt in mmsegmentation. The results are presented in Table 7 and Table 8. As can be seen, in both tasks our compressed model outperforms the non-compressed baseline in terms of throughput. Moreover, we also have lower latency than the SVD-LLM compressed backbone. However, in mIoU and mAP our models cannot catch up with the non-compressed baseline. The missing finetuning can likely be identified as the culprit for that gap. However, compared to SVD-LLM our approach still results in better segmentation performance, despite the strong numbers reported by SVD-LLM compressed Swin on ImageNet classification. Overall, while there are slight predictive performance drawbacks due to missing finetuning, our strong throughput performance warrant further investigation.

Table 7. Mask R-CNN and Retina object detection mAP and throughput on COCO dataset at 512x512 resolution with torch-compile in reduce-overhead mode on Nvidia V100.

Mask R-CNN Object Detection on COCO								
Model	Method	mAP^b	mAP^b_{50}	mAP^b_{75}	mAP^m	mAP^m_{50}	mAP^m_{75}	$(\frac{\text{Img}}{\text{s}}) \uparrow$
Swin-Large	Base [16]	47.7	69.5	52.7	43.1	66.8	46.6	16.4
	ours	46.1	68.0	50.6	42.1	65.3	45.4	**21.5**
ConvNext	Base [17]	-	-	-	-	-	-	30.1
	ours	-	-	-	-	-	-	**31.7**

7. Latency for Compression

For the latency-awareness module, we trained a random forest regressor to work as a latency predictor based on the input to the layer, the number of output features, and the selected rank. To train it, we used 1500 samples generated with a randomized pipeline. The constraints to generate each layer for a subset of individual samples is as follows:

- **Input-size**:
  - batch size by a power of 2
  - randomly choose between [7, 8, 12, 14, 16, 30, 32]
  - randomly choose between [96, 128] and multiply by a power of 2
- **Output features**: Input-size[-1] by (1, 3, 4)

Next, we obtain the latency measurements for layer uncompressed ($uncomp$) and compressed to rank 1 ($comp_{rank1}$) as a reference. Then, we create a dummy low-rank sequential module at **random rank k** ($comp_k$) and obtain the following ratio $\mathbf{Y} = \frac{comp_k - comp_{rank1}}{uncomp - comp_{rank1}}$.

With this, we fit the random forest regressor (100 estimators) on 1200 samples with the first, giving us a R^2 score of 0.95.

Table 8. Semantic segmentation MIoU and throughput on ADE20K dataset at 512x512 resolution with torch-compile in reduce-overhead mode on Nvidia V100.

Semantic Segmentation on ADE20K			
Model	Method	MIoU	Throughput $(img/s) \uparrow$
Swin-Large	Base [16]	51.5	16.8
	SVD-LLM [24]	40.1	20.4
	ours	49.0	**21.0**
ConvNext	Base [17]	50.7	26.5
	ours	47.9	**31.3**