Convolutional neural networks Memory Optimization Inference with Splitting Image

Weihao Zhuang, Tristan Hascoet, Ryoichi Takashima, Tetsuya Takiguchi and Yasuo Ariki
Graduate School of System Informatics
Kobe University
1-1 Rokkodaicho, Nada Ward, Kobe, Japan
Email: zhuangweihao@stu.kobe-u.ac.jp, tristan@people.kobe-u.ac.jp,
rtakashima@port.kobe-u.ac.jp, takigu@kobe-u.ac.jp, ariki@kobe-u.ac.jp

***Abstract*—In this paper, we propose a method to reduce the memory requirement of Convolutional Neural Networks (CNN) in the inference phase. Before feeding an input image into the CNN model, input image will split evenly to several sub-images and feed them into models respectively and combine the output after a certain layer.**

***Index Terms*—Deep learning, Convolutional Neural Networks, memory optimization**

I. Introduction

Convolutional neural networks have demonstrated excellent accuracy and performance in various computer vision tasks due to their powerful feature extraction ability. As various high accuracy CNN models are designed, more and more parameters are required in the model, and the amount of memory occupied by the activation resident on the device memory is also increasing.

Therefore, the recent research direction of CNN is toward device-friendly development, which aims to reduce the consumption of computing resources while having acceptable accuracy. Common methods are lightweight model design and model compression. Model compression can be divided into two category: pruning and quantization.

Efficient CNN architecture design has been proposed by a lot of works that by replacing the traditional vanilla convolution layer. ResNet [4], replace a 3x3 convolution between two 1x1 convolution layers which is the 'bottleneck module', reducing the number of parameters and FLOPs of the model. MobileNet [5] adopt depth-wise convolution as alternatives to normal convolution.

For model compression, pruning is to reduce redundant weight by cutting the unimportant neural connections. Han *et al.* [1] proposed to prune the weight of a pretrained model which below a certain threshold.Quantization can store the type of activation and weight into low bit int the device memory. Courbariaux *et al.* [2] binarized the weights of model into −1 and +1. Jacob *et al.* [3] adopt 8-bit integers for both activation and weight.

We proposed a method to split the input image into 4 parts before feeding them into a pretrained CNN model for inference. In this way, the size of each input image is only a quarter of its original size. In the inference phase, the peak of memory usage is the addition of model parameters and the input activation and output activation of a certain layer. After crossing the original peak point, the output activation from the four inputs is combined. It can reduce the peak of memory usage in the CNN model.

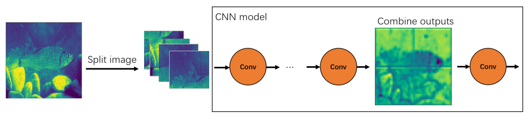
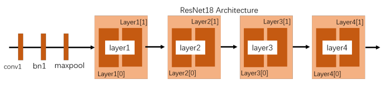


Fig. 1. Splitting input to 4 packs as new input.

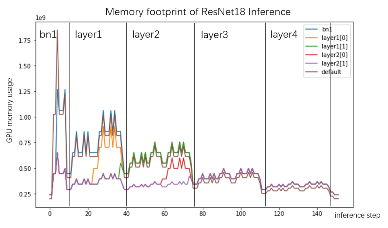
II. Splitting Image Inference

In this section, we explain the proposed method to reduce memory usage of CNN inference. The method is shown in figure 1. Input images $X \in \mathbb{R}^{c \times h \times w}$ $(c, h, w$ representing the channel, height and width) are split into four parts X_1, X_2, X_3 and X_4, size of each part is $c \times \frac{h}{2} \times \frac{w}{2}$ and feed into CNN model one by one. After passing through the peak point of memory usage originally due to the addition of parameters and input activation and output activation of a certain layer, combine four outputs activations concatenation and complete the remaining inference.

This paper uses the ImageNet dataset as the experimental dataset and Pytorch as the experimental framework. Figure 2 demonstrates the effect of combining outputs after different



(a) ResNet18 architecture



(b) Memory footprint of ImageNet dataset inference

Fig. 2. GPU memory utilization of different stages of a ResNet18 forward computation

978-1-7281-9802-6/20/$31.00 ©2020 IEEE

TABLE I
RELATIONSHIP BETWEEN OUTPUT ACTIVATION COMBINING POSITION, INFERENCE ACCURACY AND MEMORY REDUCTION OF CNN MODELS INFERENCE IMAGENET DATASET

ResNet18*	combination position	bn1	layer[0]	layer1	layer2[0]	layer2	-	-	-	Resize Image	Retrain Model
	memory reduction (%)	3.65	5.55	6.47	7.08	7.08				8.53	8.53
	accuracy (ours)	67.78	68.63	68.81	68.13	66.95				47.73	60.97
	accuracy baseline	69.67									
ResNet50*	combination position	layer1[0]	layer1[1]	layer1[2]	layer2[0]	layer2[1]	layer2 [2]	-	-	Resize Image	Retrain Model
	memory reduction (%)	0.34	0.34	1.82	4.70	4.70	4.70			6.62	6.62
	accuracy (ours)	75.33	75.63	75.61	75.67	75.38	74.94			60.10	68.03
	accuracy baseline	76.10									
MobileNetV2*	combination position	features[0]	features[1]	features[2]	features[3]	features[4]	features[5]	features[6]	features[7]	Resize Image	-
	memory reduction (%)	-0.42	-0.42	12.60	17.59	21.19	21.19	21.19	21.19	29.27	
	accuracy (ours)	70.27	69.48	71.30	70.43	71.05	69.95	69.43	70.26	51.54	
	accuracy baseline	71.68									
EfficientNet-b0**	combination position	_blocks[1]	_blocks[2]	_blocks[3]	_blocks[4]	_blocks[5]	_blocks[6]	_blocks[7]	_blocks[8]	Resize Image	-
	memory reduction (%)	10.70	14.39	17.75	17.75	17.75	17.75	17.75	17.75	23.14	
	accuracy (ours)	76.09	75.99	75.60	75.43	75.37	75.13	75.02	74.23	54.85	
	accuracy baseline	76.18									
EfficientNet-b3**	combination position	_blocks[3]	_blocks[5]	_blocks[6]	_blocks[8]	_blocks[9]	_blocks[10]	_blocks[13]	_blocks[14]	Resize Image	-
	memory reduction (%)	18.27	25.76	25.76	25.76	25.76	25.76	25.76	25.76	25.08	
	accuracy (ours)	80.48	80.40	80.32	80.28	80.23	80.21	79.74	79.56	71.43	
	accuracy baseline	80.63									
EfficientNet-b7**	combination position	_blocks[6]	_blocks[10]	_blocks[11]	_blocks[18]	_blocks[19]	_blocks[27]	_blocks[28]	_blocks[37]	Resize Image	-
	memory reduction (%)	15.25	0.21	26.10	26.28	25.98	25.98	25.98	25.98	26.50	
	accuracy (ours)	83.86	83.91	83.88	83.89	83.88	83.82	83.73	83.36	81.15	
	accuracy baseline	83.88									

*Pretrained model from https://github.com/pytorch/vision/tree/master/torchvision/models
** Pretrained model from https://github.com/lukemelas/EfficientNet-PyTorch

layers. Figure 2 (a) shows the architecture of ResNet18. Figure 2 (b) shows the amount of GPU memory use (in GB, on the y axis) at different stages of the ResNet18 computation on the Imagenet dataset. The x axis represents different layers of the network in the order of their execution, starting from the input layer on the left-most part of the plot, until the top layer of the network displayed on the right-most part of the plot. The brown curve corresponds to a baseline model, without applying our method. The peak memory usage appears during the execution of the first batch normalization layer (bn1) of the model. The blue curve illustrates the memory usage using our proposed method to combine the output activation between bn1 and the following layer. We can see that when combining outputs between bn1 and layer1, the peak memory usage was reduced. The orange curve, green curve, red curve, and purple curve illustrate the memory usage using our method to combine the output activation after different gradually layers of the network. It is important to choose a suitable layer to combine the different activations. The choice of such layer not only affects the memory usage but also the accuracy of the model. In the next section, the results of the accuracy will be shown.

III. EXPERIMENTAL RESULTS

In this section, we perform the experiments to search the relationship between the position of combining output activation, memory reduction, and accuracy.

The results of the relationship between combined position, inference accuracy, and memory reduction are shown in table I. Due to space limitations, we only compared the effects of the results of eight different layer selection. And the name of 'combination position' is determined by the implementation of the model. We also compared the results of the resized image to the original half for inference without using our proposed method. And we retrain ResNet18 and ResNet50 with resized images to verify the effectiveness of our proposed method.

Comparing our proposed method, resize image method, and retrain image method, we can conclude that our method can obtain higher accuracy and the memory reduction is comparable to the latter two. It is surprising that EfficientNet [6], a state-of-the-art model, after using our method, the trade-off between accuracy and memory reduction is better than other models. Specifically, EfficientNet-b7 reduced memory usage by 26% without accuracy dropping.

IV. CONCLUSION

In this paper, we have proposed a memory optimization method of CNN inference by split input image in four parts, feed them into model respectively, and combine output activation after a certain layer. Experimental results have verified the effectiveness of our proposed method.

REFERENCES

[1] Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

[2] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. arXiv preprint arXiv:1602.02830.

[3] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu,Matthew Tang, Andrew G Howard, Hartwig Adam, andDmitry Kalenichenko. Quantization and Training of Neu-ral Networks for Efficient Integer-Arithmetic-Only Inference.InCVPR, 2018.

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[6] Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946.