Pooling Revisited: Your Receptive Field is Suboptimal
Supplementary Document

1. Implementation Details

1.1. DynOPool Networks

For a fair comparison, we use the same model architecture and augmentation strategies as Shape Adaptor [3]. Following the prior work, only a single fully-connected layer is used at the end of CNNs for both the human-designed and DynOPool-equipped models to concentrate on the effect of feature shapes.

For classification, we replace the resizing modules (the pooling and strided convolution layers) as described below.

$$\text{Pooling}_s \rightarrow \text{DynOPool}_s, \tag{1}$$

$$\text{ReLU} \circ \text{Conv}_s \rightarrow \text{DynOPool}_s \circ \text{ReLU} \circ \text{Conv}_1. \tag{2}$$

Here, a subscription s represents the stride for each operation. In ResNet-50 [1] for ImageNet [5], the pooling layer comes right after the strided convolution layer in the first two blocks. In such a case, DynOPool is applied twice in a row (refer to (2)) and we use one DynOPool having 0.25 as the initial scaling factor to prevent the training instability. Moreover, if there are more than two branches with the same resolution as in ResNet, the same scale parameter $\boldsymbol{\alpha}$ is shared across the branches to ensure consistency.

For semantic segmentation, we plug-in DynOPool into HRNet-W48 [7] as follows:

$$\text{Conv}_s \rightarrow \text{Conv}_1 \circ \text{DynOPool}_s. \tag{3}$$

In HRNet-W48, each convolution stream contains the strided convolution layers for downsampling and the bilinear interpolation layers for upsampling. We replace the strided convolution layers with DynOPool + vanilla convolution (of stride 1) and maintain the bilinear interpolation layers. Since the resolution of the feature maps should be the same for each level of the convolution stream in HRNet-W48, we share the same DynOPool module instance within each convolution stream.

1.2. Experimental Setup

We list all ingredients and hyperparameters for our models in Table 1 and 2. Initially, we set the scale factor $\boldsymbol{r}$ of DynOPool to 0.5, which is the same as the human-designed model. The model size is adjusted through λ, however, in cases of ResNet-50 on CIFAR-100 and Aircraft, we observe that model size does not increase that much even when $\lambda = 0$. The problem is that accuracies for such models are not that impressive due to the insufficient model size. In such cases, we increase the initial scale factor of the first DynOPool layer to obtain models with good performance. We list up λs and initial scale factors of the first resizing layer for all settings in Table 2.

For the experiments in Section 3 of the main paper (CIFAR-stretch/tile/large), we use the same hyperparameters as VGG-16 [6] on CIFAR-100 [2]. λs are set to 2.33e-4/7.21e-5/5.03e-5, respectively. For EfficientNet-B0 with DynOPool, we use the same hyperparameters as VGG-16 on ImageNet. Also, λ is set to 2.50e-5.

The semantic segmentation model is trained for 100 epochs with the batch size of 16 on a single GPU. The initial learning rate is set to 1.6e-2 for the weights and 8e-3 for the $\boldsymbol{\alpha}$ with the weight decay of 1e-4. Also, λ is set to 7.00e-5. As an optimizer, we use SGD with momentum 0.9 and polynomial learning rate policy with the power of 0.9.

We implement all experiments with PyTorch [4], and use the Automatic Mixed Precision package for the ImageNet experiments.

Table 1. Hyperparameter list for the experiments.

	FGVC-Aircraft			**CIFAR-100**			**ImageNet**		
	VGG-16	ResNet-50	MBN-V2	VGG-16	ResNet-50	MBN-V2	VGG-16	ResNet-50	MBN-V2
Learning Rate	1e-2			1e-1			1e-1	1e-1	5e-2
Learning Rate (α)	1e-2			1e-2			5e-3		
Optimizer	SGD with 0.9 momentum								
Scheduler	Cosine Annealing								
Weight Decay	5e-4	5e-4	4e-5	5e-4	5e-4	4e-5	5e-4	5e-4	4e-5
Batch Size	8			128			64 (per GPU) for 4 GPUs		
Epochs	250			250			120		

Table 2. Coefficient for GMACs and learning rate for shape parameter α for each model.

		FGVC-Aircraft			**CIFAR-100**			**ImageNet**		
		VGG-16	ResNet-50	MBN-V2	VGG-16	ResNet-50	MBN-V2	VGG-16	ResNet-50	MBN-V2
DynOPool-S	λ	3.07e-5	3.99e-5	3.00e-5	2.20e-4	0	5.00e-5	1.00e-4	8.00e-5	4.00e-5
	Init. Scale (first layer)	0.5	0.5	0.5	0.5	0.6	0.5	0.5	0.5	0.5
DynOPool-B	λ	6.51e-6	0	0	1.59e-5	0	1.00e-7	7.00e-5	2.00e-5	1.00e-5
	Init. Scale (first layer)	0.5	0.7	0.5	0.5	0.8	0.5	0.5	0.5	0.5

2. Comparison with Shape Adaptor

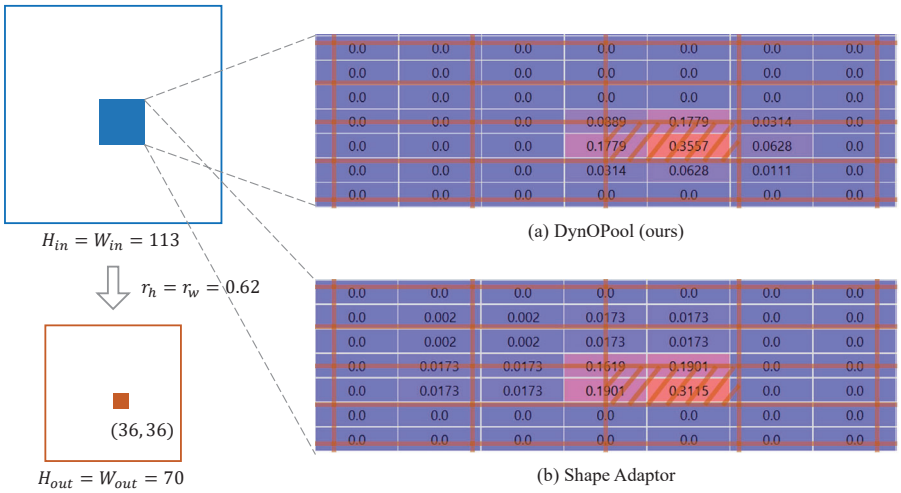

Figure 1. **Visualization of feature aggregation ranges** (a) DynOPool (ours) and (b) Shape Adaptor. The brown grid represents the grid of output features overlaid on the input feature. The numbers written in each bin of the input feature indicate the contribution of each feature to compose a shaded output bin. Since Shape adaptor finds the optimal pooling ratio by linear interpolation between two branches with different pre-defined pooling ratios, its aggregation range is highly quantized and unrelated features are involved.

In Figure 1, we visualize the feature aggregation range for both Shape Adaptor [3] and ours. Shape Adaptor uses linear interpolation between the features with the pre-defined rescaling ratios to find the optimal pooling ratio. However, since the resulted ratio lays between two pre-defined ratios, its receptive field gets much bigger than it actually needs.

Furthermore, Shape Adaptor uses pooling before the aggregation, the overall performance is closely affected by the pre-defined pooling ratio. Therefore, setting a small pooling ratio smaller than 0.5, to broaden the search space of the pooling ratio, can potentially harm the overall performance. Since we use less quantization for feature selection, each bin of the output feature can aggregate more closely related features from the input.

3. Resource Efficiency

Table 3. The relative computational cost of DynOPool compared to the entire model. DynOPool brings negligible computational overheads compared to other operators in the networks.

Dataset	FGVC-Aircraft	CIFAR-100	ImageNet
VGG-16	0.07 %	0.07 %	0.07 %
ResNet-50	0.26 %	0.13 %	0.26 %
MobileNetV2	3.88 %	1.00 %	3.86 %

In Table 3, we calculate the percentage of DynOPool's GMACs compared to the overall GMACS of the entire models. The amount of computational cost brought by DynOPool is only a fraction of the total cost. For VGG-16 and ResNet-50, our modules' costs are less than 1% of the entire models' costs. Even in MobileNetV2, where the convolution operation is relatively very light, the percentages do not exceed 10%.

The reason for the relatively small computational costs of our modules can be two-fold. First, the convolutional operation is much more expensive than the operation of bilinear interpolation. The amount of computation for convolution is proportional to $C_{\text{in}} \cdot C_{\text{out}} \cdot H_{\text{out}} \cdot W_{\text{out}}$, while the cost of bilinear interpolation is proportional only to the size of the output, $C_{\text{out}} \cdot H_{\text{out}} \cdot W_{\text{out}}$, where C_{in} and C_{out} are the number of channels of the input and output, respectively. Second, the number of pooling layers is much smaller than the number of convolution layers. For example, there are only four pooling layers in VGG-16. In conclusion, our scaling module allows for improving performance by optimizing the receptive field with a few additional parameters and computational cost.

4. Limitations and Social Negative Impacts

Since the model size is indirectly adjusted through λ, there is a limitation in finding a model of the exact target GMACs. Also, we could only estimate the distribution of information in each dataset through the obtained feature shapes, however, it would be great if future research could provide a way to numerically analyze this.

Lastly, since DynOPool was prone to increase the complexity of the model to maximize the accuracy, it could adversely affect global warming due to additional GPU operations.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[2] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009. 1

[3] Shikun Liu, Zhe Lin, Yilin Wang, Jianming Zhang, Federico Perazzi, and Edward Johns. Shape adaptor: A learnable resizing module. In *ECCV*, 2020. 1, 3

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 1

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 1

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[7] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. Deep high-resolution representation learning for visual recognition. *TPAMI*, 2019. 1