Pooling Revisited: Your Receptive Field is Suboptimal

Dong-Hwan Jang[1] Sanghyeok Chu[1] Joonhyuk Kim[1] Bohyung Han[1,2]
[1]ECE & [1]ASRI & [2]IPAI, Seoul National University
{jh01120, sanghyeok.chu, kjh42551, bhhan}@snu.ac.kr

Abstract

The size and shape of the receptive field determine how the network aggregates local features, and affect the overall performance of a model considerably. Many components in a neural network, such as depth, kernel sizes, and strides for convolution and pooling, influence the receptive field. However, they still rely on hyperparameters, and the receptive fields of existing models result in suboptimal shapes and sizes. Hence, we propose a simple yet effective Dynamically Optimized Pooling operation, referred to as DynOPool, which learns the optimized scale factors of feature maps end-to-end. Moreover, DynOPool determines the proper resolution of a feature map by learning the desirable size and shape of its receptive field, which allows an operator in a deeper layer to observe an input image in the optimal scale. Any kind of resizing modules in a deep neural network can be replaced by DynOPool with minimal cost. Also, DynOPool controls the complexity of the model by introducing an additional loss term that constrains computational cost. Our experiments show that the models equipped with the proposed learnable resizing module outperform the baseline algorithms on multiple datasets in image classification and semantic segmentation.

1. Introduction

Despite the unprecedented success of deep neural networks in various applications including computer vision [12, 24, 39, 40], natural language processing [6, 33], robotics [21], and bioinformatics [16], the design of the optimal network architecture is still a challenging problem. While several handcrafted models exhibit impressive performance in various domains, there have been substantial efforts to identify the optimal neural network architecture with associated operations automatically [17, 18, 22, 41]. However, hand-engineered architectures are prone to be suboptimal and suffer from weak generalizability while the approaches based on neural architecture search either incur a huge amount of training cost or achieve minor improvement due to limited search space.

Researchers have been investigating powerful and efficient operations applicable to deep neural networks, which include convolutions, normalizations, and activation functions. However, they have not paid much attention to pooling operations despite their simplicity and effectiveness in aggregating local features. The size and shape of a receptive field are critical; too small or large a receptive field may not be able to effectively recognize large or small objects, respectively. The receptive field is determined by several factors in deep neural networks such as the depth of a model, strides of operations, types of convolutions, etc. To design an efficient receptive field of an operation, variants of convolution operations [5, 29, 43] or special architectures with multi-resolution branches [11, 44] are widely adopted. However, these approaches rely on delicately human-engineered hyperparameters or time-consuming neural architecture search [46, 47].

To alleviate the suboptimality of human-engineered architectures and operations, we propose Dynamically Optimized Pooling operation (DynOPool), which is a learnable resizing module that replaces standard resizing operations. The proposed module finds the optimal scale factor of the receptive field for the operations learned on a dataset, and, consequently, resizes the intermediate feature maps in a network to proper sizes and shapes. This relieves us from the delicate design of hyperparameters such as stride of convolution filters and pooling operators.

Our contributions are summarized as follows:

- Our work tackles the limitations of existing scaling operators in deep neural networks that depend on predetermined hyperparameters. We point out the importance of finding the optimal spatial resolutions and receptive fields in intermediate feature maps, which are still under-explored in designing neural architectures.

- We propose DynOPool, a learnable resizing module that finds the optimal scale factors and receptive fields of intermediate feature maps. DynOPool identifies the best resolution and receptive field of a certain layer using a learned scaling factor and propagates the information to the subsequent ones leading to scale opti-

mization across the entire network.

- We demonstrate that the model with DynOPool outperforms the baseline algorithms on multiple datasets and network architectures in the image classification and semantic segmentation tasks. It also exhibits desirable trade-offs between accuracy and computational cost.

Our paper is organized as follows. Section 2 presents existing related works and Section 3 introduces our motivation for optimizing the size and shape of the receptive field and feature map. We describe the technical details of DynOPool in Section 4 and experimental results in Section 5. Last, we conclude this work and discuss future works in Section 6.

2. Related Works

Neural architecture search Neural Architecture Search (NAS) [22, 25, 31, 46, 47] is an AutoML method that optimizes the structure of a deep neural network architecture by formulating a hyperparameter setting with human inductive bias as a learnable procedure. Previous approaches based on reinforcement learning [31, 46, 47] require huge amount of GPU time. Although several methods have been proposed to accelerate the search process by sharing weights [31] or gradient-based optimization [22, 25], they are still suboptimal due to search space constraints. There exist a couple of prior works to search for input resolutions [13, 41], but finding the optimal feature size and shape for each layer is still a challenging problem.

Dynamic kernel shape Recent approaches [8, 15, 30, 32, 37] adopt variants of convolutions that learn the sizes of receptive fields dynamically. N-Jet [32] employs Gaussian derivative filters to adapt kernel size using the scale-space theory. CKConv [37] uses a continuous kernel parameterization trick to implement kernels of diverse sizes without additional cost. Similarly, FlexConv [8] utilizes the implicit neural representation to generate large-bandwidth filters of varying sizes. These methods identify the optimized receptive fields by learning filter sizes while our approach does it via learning the size of the feature map.

Learnable resizing modules Shape Adaptor [23] controls the receptive field by direct learning of the feature map size. It proposes a differentiable resizing module applicable to a linear combination of a pooled feature map with a ratio (*e.g.* 0.5 or 1.5) and a non-pooled map. However, the resizing module is limited to selecting one of the pre-defined ratios for upsampling or downsampling, and processing the symmetric resizing only. Recently, DiffStride [34] presents a spectral pooling method to determine the optimal stride of the pooling layer. They find an appropriate feature map size and shape by replacing downsampling in the spatial domain with cropping in the frequency domain, where the cropping window size is optimized.

3. Motivation

The information in an image is spread over various levels of locality, and the CNN learns patterns with diverse scales using a series of kernels to learn strong representations. Since the sizes and shapes of semantically meaningful patterns differ greatly for each image, it is important to identify proper receptive fields and extract useful information using the receptive fields from an image. However, the exploration of the optimal receptive field has not been studied actively and the use of adaptive feature map size has hardly been discussed so far although several previous works indirectly learn the receptive field size via other methods such as neural architecture search or design learnable receptive fields with excessive constraints. This section presents why a conventional receptive field with a fixed size and shape is suboptimal and discusses how DynOPool tackles this issue through toy experiments with VGG-16 [40] on CIFAR-100 [19].

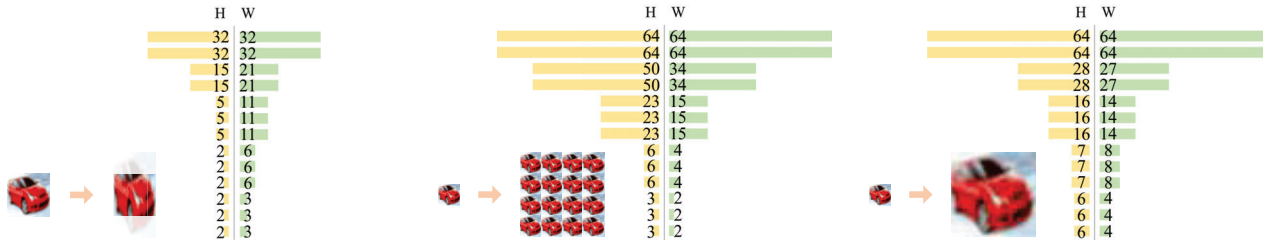
3.1. Asymmetrically Distributed Information

Datasets inherently have information asymmetry due to domain characteristics. For example, the barcode image does not have any information along the vertical direction because the same value is repeated in the direction. Therefore, it is desirable to concentrate on the horizontal direction for representing the barcode images. The problem is that, except for the images with the prior information like barcodes, the inherent asymmetry is not measurable in most cases. Also, input resizing, which is often used as preprocessing, sometimes leads to information asymmetry. In human-designed networks, the aspect ratio of an image is typically adjusted to satisfy the input specifications of models. However, the receptive fields in such networks are not designed to handle the operations.

To demonstrate the potential of the proposed approach, DynOPool, we perform experiments on CIFAR-stretch, a toy dataset in which images of CIFAR-100 are vertically stretched twice in the vertical direction and cropped randomly to a size 32x32. As shown in Figure 1(a), DynOPool adopts a wide feature map and extracts valuable information more in the horizontal direction to achieve improved performance compared to the human-designed model.

3.2. Densely or Sparsely Distributed Information

The level of locality is another interesting component for designing optimal models. CNNs learn the complex representations from an image by aggregating local information in a cascaded manner. However, the value of the local information depends heavily on the properties of each example. For example, when an image is blurred, all micro patterns, such as the texture of an object, are wiped out. In this case, it would be better to extend the receptive field in early layers



0.30G / 71.18% (0.31G / 69.49%)
(a) CIFAR-stretch

1.24G / 60.06% (1.26G / 56.70%)
(b) CIFAR-tile

1.18G / 67.82% (1.26G / 65.49%)
(c) CIFAR-large

Figure 1. We conduct toy experiments on CIFAR-100 with three different synthetic datasets; (a) random crop of a vertically stretched image (b) tile a halved image in a 4×4 grid (c) quadruple a halved image. Although the contents are almost the same, the optimal size and shape of each feature map is greatly different depending on the characteristics of input images. Unlike the human-designed model, which has fixed feature map sizes, our models adjust the feature map sizes to maintain the optimal amount of information in each feature map, leading to improved performance. The numbers in bold face fonts are GMACs and the accuracy given by DynOPool while the numbers in parentheses are from human-designed models.

and concentrate on global information. On the other hand, if an image contains plenty of class-specific information, *e.g.*, texture, local patterns would be more important.

To verify the hypothesis, we construct two variants of the CIFAR-100 dataset, CIFAR-tile and CIFAR-large, as shown in Figure 1. To this end, we first downsample the original images in CIFAR in half and construct 16×16 images. Then, we tile the downsampled image in a 4×4 for CIFAR-tile, and upsample the downsampled images to size 64×64 for CIFAR-Large.

As expected, our models illustrated in Figure 1(b) and (c) outperform the human-designed model by large margins. Although both datasets are constructed with the same set of base images of size 16×16, the learned networks by DynOPool have different shapes; our model trained on CIFAR-tile has larger feature maps than the model trained on CIFAR-large in the early layers. Note that DynOPool for the CIFAR-tile prefers to employ small receptive fields at the beginning of the network because the tiled objects are very small. On the other hand, our model for the CIFAR-large is encouraged to have large receptive fields in the low level because the input image is magnified from a small one and it makes sense to observe large areas in the early layers.

4. Proposed Method

We discuss the proposed learnable resizing module, referred to as DynOPool, in detail, which includes its concept, optimization, and practical benefits.

4.1. Dynamically Optimized Pooling (DynOPool)

The resizing module in DynOPool, which accepts an input feature map, $x_{\text{in}} \in \mathbb{R}^{H_{\text{in}} \times W_{\text{in}}}$, and returns a resized output, $x_{\text{out}} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}}}$, is defined and optimized as follows.

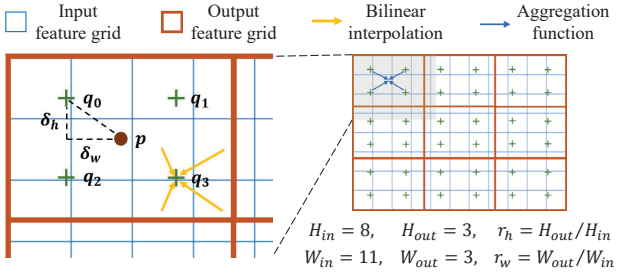


Figure 2. Overview of the proposed resizing module, DynOPool (best viewed in color). We optimize the scale factor $\boldsymbol{r} = (r_h, r_w)$ between a pair of input and output feature maps, denoted by x_{in} and x_{out}, respectively. The brown dot $\boldsymbol{p}$ represents the center of a grid cell in x_{out} while the green crosses indicate four query points $\boldsymbol{q}$ in the same cell. The representation of q_i is given by bilinear interpolation of the features corresponding to the four nearest pixels in x_{in}. An output feature of a grid cell in x_{out} is derived by the feature aggregation of the four query points, where a simple aggregation function such as max-pooling is typically employed.

4.1.1 Design of DynOPool

Figure 2 illustrates how DynOPool works. DynOPool first divides the feature map x_{in} into an $H_{\text{out}} \times W_{\text{out}}$ grid as

$$\begin{aligned} H_{\text{out}} &= \lfloor H_{\text{in}} \cdot r_h \rceil \\ W_{\text{out}} &= \lfloor W_{\text{in}} \cdot r_w \rceil, \end{aligned} \tag{1}$$

where $\boldsymbol{r} = (r_h, r_w)$ indicates the scale factor for height and width of a feature map and $\lfloor \cdot \rceil$ is a round operation. Assuming that $(-1, -1)$ and $(1, 1)$ are the normalized coordinates of the top-left and bottom-right corners of x_{in}, the size of a grid cell in the output feature map becomes $\frac{2}{H_{\text{out}}} \times \frac{2}{W_{\text{out}}}$.

Then, given a grid cell centered at $\boldsymbol{p} = (p_h, p_w)$, the

positions of the four query points are defined as

$$\begin{aligned}\boldsymbol{q} &= (p_h \pm \delta_h,\ p_w \pm \delta_w) \\ &= \left(p_h \pm \frac{1}{4}\cdot\frac{2}{H_{\text{out}}},\ p_w \pm \frac{1}{4}\cdot\frac{2}{W_{\text{out}}}\right),\end{aligned} \tag{2}$$

where $\boldsymbol{\delta} = (\delta_h, \delta_w)$ denotes the displacement from $\boldsymbol{p}$. The representation of each query point is given by bilinear interpolation of four nearest grid cells in x_{in}. Then, DynOPool aggregates the four feature vectors and returns the output representation of each grid cell in x_{out}. We choose max-pooling as an aggregation function, but any other function can replace max-pooling as long as it is effective to compute abstract representations from multiple local features.

The primary benefits of DynOPool with the optimized scale factor $\boldsymbol{r}$ are twofold. First, the location of four query points $\boldsymbol{q}$ are also optimized because $\boldsymbol{\delta}$ is a function of $\boldsymbol{r}$. Second, by obtaining the best resolution of an intermediate feature map through the optimization of $\boldsymbol{r}$, DynOPool adaptively controls the size and shape of receptive fields in deeper layers with other operators intact.

4.1.2 Optimization

The rescaling module is defined by a combination of (1) and (2), which are based on simple operations. However, the rounding operations are not differentiable and hinder the optimization procedure of DynOPool. To remedy this issue, we leverage a differentiable quantization trick, which is a well-known continuous relaxation technique for discrete random variables [14, 26]. Then the rescaling modules are given by reformulating the round functions as follows:

$$H_{\text{out}} = \lfloor H_{\text{in}} \cdot r_h \rceil + H_{\text{in}} \cdot r_h - sg(H_{\text{in}} \cdot r_h), \tag{3}$$

$$W_{\text{out}} = \lfloor W_{\text{in}} \cdot r_w \rceil + W_{\text{in}} \cdot r_w - sg(W_{\text{in}} \cdot r_w), \tag{4}$$

where $sg(\cdot)$ indicates a stop gradient operator [1]. Note that (3) and (4) allow us to feedforward the original discrete values $\lfloor H_{\text{in}} \cdot r_h \rceil$ and $\lfloor W_{\text{in}} \cdot r_w \rceil$ while backpropagating through their continuous surrogate functions $H_{\text{in}} \cdot r_h$ and $W_{\text{in}} \cdot r_w$.

Although the optimization is now feasible, there remains an additional challenge in learning the scale factor $\boldsymbol{r}$. As expressed in (2), the rescaling module involves the displacement function, $\boldsymbol{\delta}$, which depends on $\boldsymbol{r}$. However, the gradient with respect to $\boldsymbol{r}$ is unstable when either r_h or r_w is small because the gradient is inversely proportional to r_h^2 or r_w^2 as

$$\frac{d\delta_h}{dr_h} \propto -\frac{1}{r_h^2} \quad \text{and} \quad \frac{d\delta_w}{dr_w} \propto -\frac{1}{r_w^2} \tag{5}$$

Since this gradient explosion results in significant changes in the resolution of x_{out} during training, we reparameterize $\boldsymbol{r}$ using $\boldsymbol{\alpha} = [\alpha_h, \alpha_w]$ as follows:

$$[\alpha_h, \alpha_w] = [r_h^{-1}, r_w^{-1}]. \tag{6}$$

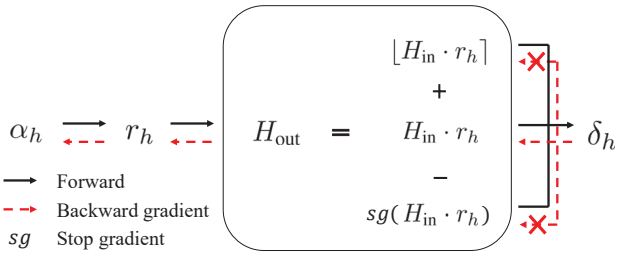


Figure 3. Computational flows inside DynOPool. Although the forward pass employs the discretized value, $\lfloor H_{\text{in}} \cdot r_h \rceil$, its continuous counterpart ($H_{\text{in}} \cdot r_h$) is adopted in the backward pass to backpropagate the gradients into $\boldsymbol{\alpha}$. The same optimization process is applied with respect to width.

By defining $\boldsymbol{\alpha}$ as a learnable scale parameter and optimizing it instead of $\boldsymbol{r}$, the training procedure is greatly stabilized in practice. Figure 3 illustrates the overall optimization process.

4.2. Constraints for Model Complexity

To maximize the accuracy of models, DynOPool sometimes has a large scale factor and increases the resolution of intermediate feature maps. Therefore, to constrain computational cost and reduce model size, we introduce an additional loss term $\mathcal{L}_{\text{GMACs}}$, which is given by a simple weighted sum of layerwise GMACs counts at each training iteration t as follows:

$$\begin{aligned}\mathcal{L}_{\text{GMACs}} &= \sum_{\ell=1}^{N} w_\ell^t \cdot \text{GMACs}[\ell] \\ &= \sum_{\ell=1}^{N} \frac{H_{\text{out}}^{t\ (\ell)} \cdot W_{\text{out}}^{t\ (\ell)}}{H_{\text{out}}^{0\ (\ell)} \cdot W_{\text{out}}^{0\ (\ell)}} \cdot \text{GMACs}[\ell],\end{aligned} \tag{7}$$

where N is the total number of layers in the model, GMACs$[\ell]$ represents the GMACs counts in the ℓ^{th} layer at the initial state, and w_ℓ^t is the ratio between the feature map sizes in the ℓ^{th} layer at the initial stage ($H_{out}^{0\ (\ell)}$, $W_{out}^{0\ (\ell)}$) and the current training iteration t ($H_{out}^{t\ (\ell)}$, $W_{out}^{t\ (\ell)}$). By definition, $\mathcal{L}_{\text{GMACs}}$ reflects the degree of the computational cost increase as the scale factor $\boldsymbol{r}$ changes during training, compared to the initial state of the model.

4.3. Loss

We train the model with DynOPool by a linear combination of the task-specific objectives ($\mathcal{L}_{\text{task}}$) and the proposed GMACs loss ($\mathcal{L}_{\text{GMACs}}$) as follows:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda \cdot \mathcal{L}_{\text{GMACs}}, \tag{8}$$

where λ is a hyperparameter that controls the computational complexity of a model and maintains the balance with the

task-specific loss. The model is trained to maximize its performance by jointly learning the optimal spatial resolutions of intermediate feature maps.

4.4. Versatility of DynOPool

Due to its model-agnostic property, DynOPool can replace all kinds of resizing operators in any given network. To analyze the superiority of the optimized scale factor $\boldsymbol{r}$ to the predetermined methods relying on hyperparameters, we replace all types of resizing operators in the baseline network with DynOPool except for the last global average pooling layer; pooling operations (*e.g*. max-pooling) are replaced by DynOPool and strided convolutions are replaced by the combinations of a vanilla convolution (with stride 1) and DynOPool. For a detailed description of each model, please refer to the supplementary document.

Unlike other methods that require to select either downsampling or upsampling in advance and depend on the pre-defined pooling ratios, DynOPool learns to resize feature maps without the constraint for the scale factor and the pooling ratio. In practice, the upsampling process of DynOPool is the same as the downsampling. A tricky thing in upsampling is that it can use the features of the same set of pixels to calculate the features of different query points. However, it does not incur any issue because the distances to the pixels from each query point are different and the features for each query point are different.

5. Experiments

This section summarizes the experimental results with DynOPool on various types of networks and datasets. For the classification task, we use three datasets and three types of networks for evaluation. We compare our model with human-designed models and Shape Adaptor [23] in terms of accuracy and GMACs, and present that dynamic resizing layers boost performance with almost no extra cost. Furthermore, we apply our module to EfficientNet [41] to show compatibility to the NAS algorithms, and conduct an additional experiment on PascalVOC [7] to prove applicability to the semantic segmentation task.

5.1. Experiment Setup

Models We mainly apply DynOPool to three baselines: VGG-16 [40], ResNet-50 [12], and MobileNetV2 [39]. We also use EfficientNet-B0 [41] to check compatibility with NAS. DynOPool is adopted to the downscaling module of each model and keeps the rest of the structure the same as the human-designed architecture. It is worth noting that there is no increase in the number of parameters of the models with DynOPools except for the scale parameter $\boldsymbol{\alpha}$.

Datasets We conduct the experiment on three datasets including FGVC-Aircraft [27], CIFAR-100 [19] and ImageNet [38]. Unlike CIFAR-100 and ImageNet that contain diverse general objects, Aircraft is a fine-grained dataset for aircraft classification. CIFAR-100 is a dataset with small (32×32) images while the size of the images in Aircraft and ImageNet are large (224×224). The experiment setting is to verify that DynOPool performs well regardless of image sizes or data characteristics.

Implementation details For optimization, we employ the same hyperparameters as Shape Adaptor except for the number of epochs. According to our experience, DynOPool requires more epochs than Shape Adaptor for training to allow both the scale factor and weights to converge sufficiently in response to the dynamic model structure changes. Especially, CIFAR-100 and Aircraft, which have relatively small datasets, are greatly affected by the epoch. Accordingly, we increase the epoch from 200 to 250 for models with DynOPool on both datasets.

The learning rate for the scale parameter $\boldsymbol{\alpha}$ is lower than that of the model parameter similar to other dynamic networks [5, 23] since the scale parameter affects the entire model even with its slight changes. To prevent the feature map size from reducing to 1 during training, we bound the output feature map size by $H_{\text{out}} = \lfloor \max(H_{\text{in}} \cdot r_h, 1.5) \rceil$, which ensures that the size of a feature map is at least 2 in each dimension while allowing a model to backpropagate gradients through the feature map in any dimension smaller than 2. For other hyperparameters and experimental settings, we list details in the supplementary document.

5.2. Comparison with Human-Designed Model

We discuss the performance and the characteristics of the proposed approaches in comparison with the human-designed models.

5.2.1 Main results

Table 1 presents the performance of DynOPool in terms of GMACs and accuracy. We compare the human-designed model with two variants of our model with DynOPool: 1) a model with a small computational cost similar to that of the human-designed model (DynOPool-S) and 2) a model learned mainly for accuracy (DynOPool-B).

DynOPool-S improves accuracy significantly with almost the same or fewer GMACs as the human-designed model in most cases, and DynOPool-B outperforms the human-designed model in all settings. Note that we greatly improve the performance by changing the size and shape of feature maps with little increase in the number of parameters. To achieve this goal with NAS, it would take at least a few dozen GPU days since the search space is huge due to a large number of resizing layers and the consideration of information asymmetry. On the contrary, DynOPool solves

Table 1. Top-1 accuracy (%) and GMACs comparisons between human-designed models and models with DynOPool. The sizes and shapes of the feature maps for each block in the network architecture are also reported. DynOPool-S outperforms human-designed models with comparable GMACs in almost all cases. Notably, DynOPool-S compresses the model up to 33% lighter than the human-designed VGG-16 for the ImageNet dataset while maintaining the accuracy of the model. DynOPool-B outperforms the human-designed models with significant margins in all cases.

	Dataset	FGVC-Aircraft			CIFAR-100			ImageNet		
		Acc.	GMACs	Feature map sizes	Acc.	GMACs	Feature map sizes	Acc.	GMACs	Feature map sizes
VGG-16	Human	85.3	15.40	[224,224] [112,112] [56,56] [28,28] [14,14]	75.4	0.31	[32,32] [16,16] [8,8] [4,4] [2,2]	73.9	15.39	[224,224] [112,112] [56,56] [28,28] [14,14]
	DynOPool-S	87.0	13.90	[224,224] [114,142] [52,53] [30,19] [17,7]	75.5	0.36	[32,32] [21,14] [10,7] [5,4] [2,2]	73.8	10.16	[224,224] [88,87] [40,37] [24,23] [12,12]
	DynOPool-B	**87.4**	32.39	[224,224] [127,256] [76,102] [46,37] [20,11]	**79.8**	1.71	[32,32] [37,32] [21,18] [12,9] [7,4]	**74.1**	20.92	[224,224] [151,152] [67,68] [32,30] [15,13]
ResNet-50	Human	81.6	4.12	[224,224] [56,56] [28,28] [14,14] [7,7]	78.5	1.31	[32,32] [16,16] [8,8] [4,4]	77.2	4.11	[224,224] [56,56] [28,28] [14,14] [7,7]
	DynOPool-S	82.3	3.57	[224,224] [58,63] [18,17] [9,4] [4,2]	80.3	1.01	[32,32] [10,9] [5,4] [2,2]	77.6	6.20	[224,224] [71,71] [27,26] [12,11] [4,2]
	DynOPool-B	**87.2**	38.53	[224,224] [225,210] [68,66] [16,17] [4,4]	**80.6**	1.73	[32,32] [18,17] [7,6] [2,3]	**78.1**	12.80	[224,224] [102,99] [43,41] [16,17] [4,4]
MBN-V2	Human	77.6	0.33	[224,224] [112,112] [56,56] [28,28] [14,14] [7,7]	73.8	0.09	[32,32] [16,16] [8,8] [4,4]	71.7	0.31	[224,224] [112,112] [56,56] [28,28] [14,14] [7,7]
	DynOPool-S	78.7	0.34	[224,224] [98,119] [39,42] [36,18] [21,9] [12,4]	74.0	0.08	[32,32] [13,13] [6,6] [4,4]	72.1	0.49	[224,224] [111, 111] [55,50] [32,27] [20,16] [9,7]
	DynOPool-B	**82.6**	2.35	[224,224] [181,150] [132,174] [87,80] [51,36] [22,13]	**76.2**	0.21	[32,32] [22,21],[12,12] [7,7]	**73.8**	1.16	[224,224] [181,171] [95,93] [53,53] [31,29] [10,10]

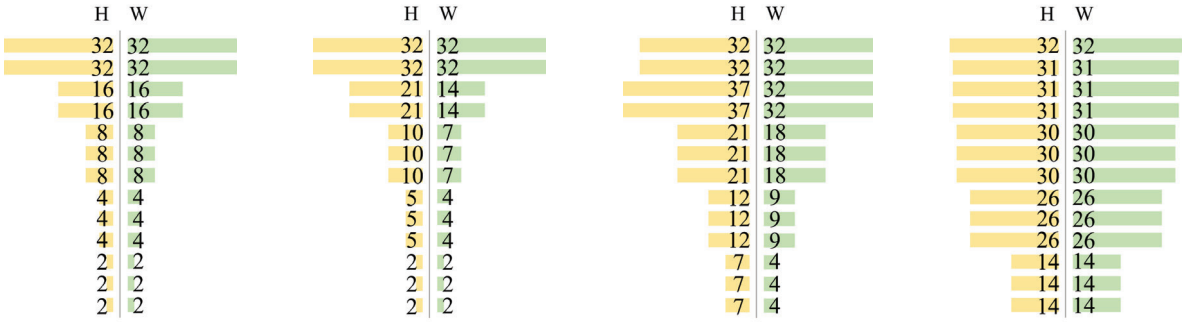


Figure 4. Visualization of trained models with DynOPool and Shape Adaptor from the human designed VGG-16 on the CIFAR-100 dataset. We visualize the sizes and shapes of intermediate feature maps in each model with GMACs and accuracy. By learning the optimal scale parameter $\boldsymbol{\alpha}$ for the dataset, DynOPool illustrates competitive performance compared to the human-designed model and Shape Adaptor.

the above problem successfully and identifies an optimized network without an exhaustive search process.

On FGVC-Aircraft, as presented in Table 1, trained networks have many non-square feature maps with the receptive fields of the reciprocal shapes and achieve the largest performance improvements among all the tested datasets. Since the images in the fine-grained dataset share relatively many patterns in common than in general images, it may be critical to find the optimal shape of the receptive field to achieve better accuracy. It is interesting that DynOPool-S models have wide feature maps in the early layers but end up with tall feature maps in the deeper layers. This fact implies that the proposed dynamic resizing modules concentrate on the information in the horizontal direction in analyzing local patterns, which forces the information in the vertical direction to become more important in identifying semantic structures in images. As a result, it turns out to achieve superior performance with less computation than the human-designed models relying on the feature maps with the standard sizes and shapes.

Table 1 illustrates another interesting results about the feature map shapes of the networks trained on CIFAR-100 and ImageNet, which contain more general object categories in images than FGVC-Aircraft. The feature maps are optimized for vertical shapes, *i.e.*, $H > W$, in almost all settings, which also aligns with the result from the previous work [34]. This implies that the amount of information in the ImageNet and CIFAR-100 datasets have is asymmetric in the spatial dimensions and we can extract more information by observing the details in the vertical direction than in the horizontal direction.

Furthermore, we visualize the feature map sizes of the human-designed model, DynOPool-S/B, and Shape Adaptor in Figure 4. As shown in Figure 4(b) and (c), DynOPool-S/B learn to utilize non-square feature maps and exhibit the data-driven model selection capability. In particular,

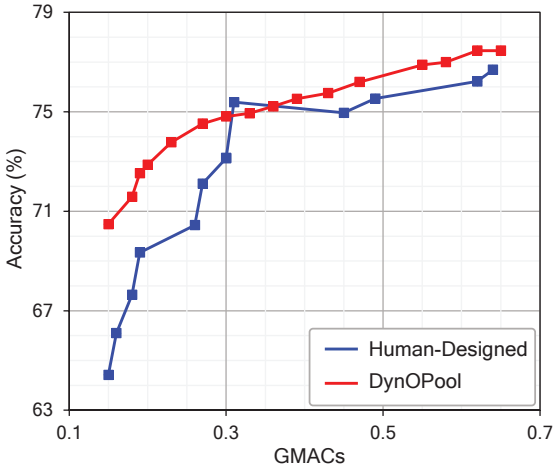


Figure 5. GMACs-Accuracy tradeoffs between human-designed VGG-16 and VGG-16 with DynOPool on CIFAR-100. The models with DynOPool are trained with different values of λ while human-designed models are trained by varying input resolutions.

DynOPool-B even increases the feature map size after the first pooling layer, which leads to substantial accuracy gain by 4.4%p compared to the human-designed model.. The result of DynOPool-B shows that the full use of local information in the front layers is sometimes helpful while enlarging the receptive field size later to reduce the sizes of the corresponding feature maps.

5.2.2 Trade-off between accuracy and GMACs

Figure 5 illustrates the GMACs-accuracy tradeoffs between our model with DynOPool and human-designed model with VGG-16 on CIFAR-100. We adjust the input image size to obtain the accuracies of the human-designed model, VGG-16 with respect to different computational cost in terms of GMACs. This is motivated by the strategy of several NAS algorithms that include the input size in the search space [22, 41]. For DynOPool, we control GMACs by varying the coefficient for the GMACs loss λ in (8).

DynOPool shows superior trade-off between accuracy and GMACs compared to the human-designed model in almost all cases, especially when the models are compressed significantly. This is because, by using our approach, the model structure is optimized dynamically and effectively for the target GMACs. In the case of the human-designed model, the performance is optimized with a good trade-off when the input image size is exactly 32×32 (0.31 GMACs). We believe that this is because the CIFAR-100 dataset has been tested extensively for years using its original image size and most of the human-designed models are optimized best for the input size. Also, the human-designed models may not be effective to handle non-conventional input image sizes other than numbers to the power of 2 due to the potential errors given by extra paddings.

Table 2. Comparison between DynOPool and Shape Adaptor on the CIFAR-100 dataset. DynOPool consistently outperforms Shape Adaptor with lower computational costs.

Backbone	**Model**	Acc.	GMACs
VGG-16	Shape Adaptor	79.2	5.21
	DynOPool (ours)	**79.8**	**1.71**
ResNet-50	Shape Adaptor	80.3	4.93
	DynOPool (ours)	**80.6**	**1.73**
MobileNetV2	Shape Adaptor	75.7	0.92
	DynOPool (ours)	**76.2**	**0.21**

Table 3. Performance of DynOPool with EfficientNet-B0 on the ImageNet dataset.

Backbone	**Model**	Acc.	GMACs
EfficientNet-B0	Human	71.8	0.42
EfficientNet-B1		72.8	0.75
EfficientNet-B0	DynOPool (ours)	72.3	0.58

5.3. Comparison with Shape Adaptor

Table 2 compares the accuracy and the GMACs between DynOPool and Shape Adaptor [23]. Although both algorithms aim to find the optimal feature map sizes by introducing learnable resizing modules, DynOPool outperforms Shape Adaptor in terms of both accuracy and efficiency. We demonstrate the feature map sizes of Shape Adaptor in Figure 4(d) together with the accuracy and the computational complexity.

We believe that the following trait of our method drives the difference. Shape Adaptor determines the output feature map size by a linear interpolation of two pre-defined candidate size scales. This strategy results in large approximation errors by forcibly considering potentially irrelevant features for aggregations under the predicted scale factor. On the contrary, DynOPool adjusts the feature map size naturally using a single scale factor $\boldsymbol{r}$, which is reparametrized by $\boldsymbol{\alpha}$ for stable optimization. More detailed comparisons between the two approaches are discussed in the supplementary document.

5.4. Compatibility with NAS algorithms

While NAS is a more general concept than DynOPool, the feature map size is not typically considered in the search space in NAS and the architecture can be optimized by NAS jointly with DynOPool. We adopt DynOPool for the optimization of EfficientNet [41], which is one of the state-of-art architectures identified by NAS. As seen in Table 3,

Table 4. Semantic segmentation results of HRNet-W48 on PascalVOC. DynOPool compresses the human-designed model up to 16% with slight improvement of mIoU.

Model	mIoU	GMACs	Feature map sizes
Human	76.2	82.55	[240,240] [120,120] [60,60] [30,30] [15,15]
DynOPool (ours)	**76.4**	**69.39**	[367,349] [134,130] [52,50] [22,21] [10,9]

EfficientNet-B0 with DynOPool shows competitive performance in terms of accuracy and GMACs compared to both EfficientNet-B0 and EfficientNet-B1.

Although the benefit of DynOPool is not impressive in this result, the combination of NAS and DynOPool are formulated as a differentiable optimization task even in the feature map scale dimension; it has potential to lead to higher accuracy with less computational cost. Note that, while the architecture of EfficientNet-B1 is identified from a combinatorial search space in 1) width, 2) depth, and 3) resolution dimensions, we can find competitive models with the optimized feature map sizes at a substantially reduced search time using DynOPool.

5.5. Semantic Segmentation Results

To further verify the effectiveness of DynOPool, we conduct additional experiments on semantic segmentation. The semantic segmentation task involves various objects and stuff in a scene with various scales, identifying the optimal receptive field corresponding to each object is critical to improve the final accuracy. To get semantically richer and spatially more precise representations, multi-scale representation learning is is the prevalent approach in semantic segmentation models [3, 4, 42, 45]. For example, HRNet [42] maintains high-resolution representations throughout the whole process and connects the high-to-low resolution convolution streams in parallel.

To evaluate the performance of DynOPool in semantic segmentation, we employ HRNet-W48, a variant of HRNet, as our backbone model, and replace the strided convolutions in the model by a combination of DynOPool and a vanilla convolution (with stride 1). We train the models on the PascalVOC [7] dataset to check if there exists further room for improvement. As seen in Table 4, DynOPool successfully compresses the human-designed model up to 16% with a slight improvement of the mIoU. Interestingly, our model enlarges the resolution of the convolution stem and the upper branch of the parallel convolution stream, and consistently reduces the resolution of the remaining three branches of parallel convolution streams. This highlights the importance of maintaining the feature maps with data-driven feature map sizes to improve performance with less computational burden. We present detailed experimental settings in the supplementary document.

6. Conclusion and Future Works

6.1. Conclusion

We presented a Dynamically Optimized Pooling, referred to as DynOPool, which facilitates finding an optimized sizes and shapes of receptive fields and feature maps. DynOPool identifies the optimal size and shape of feature maps without relying on human inductive bias or exhaustive architecture search. Our module achieved superior performance with various recognition models on multiple datasets, and showed desirable trade-offs between accuracy and computational cost, compared to the human-designed model and the previous work. We also showed that DynoPool is compatible with the recent NAS algorithms and naturally applicable to semantic segmentation model. We hope that our module allows the vision community to optimize deep neural networks more effectively.

6.2. Future Works

Although we focus on the two-dimensional tasks in this work, our module could be extended to higher dimensional scaling modules. For example, in an action recognition task, we can also employ DynOPool to capture temporal relation from a dataset by adjusting the number of frames required for temporal pooling.

Furthermore, similar to our findings, in cognitive science, it has been well-known for decades that the human visual system perceives vertical lines to be slightly longer than horizontal ones [9, 20, 35] and judge the symmetry based more on the horizontal symmetry than the vertical counterpart [10, 36]. In other words, our visual system has been adapted to be more sensitive to vertical information changes. Despite the long history, the exact cause has not yet been identified and is still under discussion [2, 28]. It would be worthwhile to investigate the connection between the findings from our work and the observations in cognitive science, which makes a synergy to understand the asymmetric behavior of computer vision and human vision systems and bridges a missing link between two research fields.

Acknowledgement Dong-Hwan Jang is grateful for financial support from Hyundai Motor Chung Mong-Koo Foundation. This research was supported in part by Samsung Advanced Institute of Technology and by the Bio & Medical Technology Development Program of the National Research Foundation (NRF) [No. 2021M3A9E4080782] and Institute of Information & communications Technology Planning & Evaluation (IITP) grants [No.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University) ; No.2021-0-02068, Artificial Intelligence Innovation Hub] funded by the Korea government (MSIT).

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 4

[2] John W Berry, John Widdup Berry, Ype H Poortinga, Marshall H Segall, and Pierre R Dasen. *Cross-cultural psychology: Research and applications*. Cambridge University Press, 2002. 8

[3] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, 2016. 8

[4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 8

[5] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017. 1, 5

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 1

[7] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010. 5, 8

[8] Hendrik P. A. Lensch Fabian Groh, Patrick Wieschollek. Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In *ACCV*, 2018. 2

[9] Frank W Finger and David K Spelt. The illustration of the horizontal-vertical illusion. *Journal of Experimental Psychology*, 37(3):243, 1947. 8

[10] Celia B Fisher and Maria P Fracasso. The goldmeier effect in adults and children: Environmental, retinal, and phenomenal influences on judgments of visual symmetry. *Perception*, 16(1):29–39, 1987. 8

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *TPAMI*, 2015. 1

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 5

[13] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019. 2

[14] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017. 4

[15] Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. In *CVPR*, 2017. 2

[16] Wei Ji, Shuang Yu, Junde Wu, Kai Ma, Cheng Bian, Qi Bi, Jingjing Li, Hanruo Liu, Li Cheng, and Yefeng Zheng. Learning calibrated medical image segmentation via multi-rater agreement modeling. In *CVPR*, 2021. 1

[17] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. In *ICML*, 2020. 1

[18] Heewon Kim, Seokil Hong, Bohyung Han, Heesoo Myeong, and Kyoung Mu Lee. Fine-grained neural architecture search. *arXiv preprint arXiv:1911.07478*, 2019. 1

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009. 2, 5

[20] Teodor M Künnapas. An analysis of the" vertical-horizontal illusion.". *Journal of Experimental Psychology*, 49(2):134, 1955. 8

[21] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Single-view robot pose and joint angle estimation via render & compare. In *CVPR*, 2021. 1

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2018. 1, 2, 7

[23] Shikun Liu, Zhe Lin, Yilin Wang, Jianming Zhang, Federico Perazzi, and Edward Johns. Shape adaptor: A learnable resizing module. In *ECCV*, 2020. 2, 5, 7

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 1

[25] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018. 2

[26] C Maddison, A Mnih, and Y Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017. 4

[27] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 5

[28] Pascal Mamassian and Marie de Montalembert. A simple model of the vertical–horizontal illusion. *Vision Research*, 50(10):956–962, 2010. 8

[29] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 1

[30] George Papandreou, Iasonas Kokkinos, and Pierre-André Savalle. Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. In *CVPR*, 2015. 2

[31] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018. 2

[32] Silvia L Pintea, Nergis Tomen, Stanley F Goes, Marco Loog, and Jan C van Gemert. Resolution learning in deep convolutional networks using scale-space theory. *IEEE Transactions on Image Processing*, 2021. 2

[33] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. 1

[34] Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour. Learning strides in convolutional neural networks. *arXiv preprint arXiv:2202.01653*, 2022. 2, 6

[35] James Outram Robinson. *The psychology of visual illusion*. Courier Corporation, 2013. 8

[36] Irvin Rock and Robin Leaman. An experimental analysis of visual symmetry. *Acta Psychologica*, 1963. 8

[37] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021. 2

[38] 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. 5

[39] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 1, 5

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2, 5

[41] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 1, 2, 5, 7

[42] 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. 8

[43] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 1

[44] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. In *ECCV*, 2020. 1

[45] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017. 8

[46] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1, 2

[47] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 2