Supplementary of Highly Efficient Salient Object Detection with 100K Parameters

Anonymous ECCV submission

Paper ID 50

1 Implementation of gOctConv

Unlike vanilla OctConv that only processes features with low/high resolutions, gOctConv tasks in features with arbitrary scales and output features with arbitrary scales. Input features X contains arbitrary parts with different resolutions $[X_1, X_2, ..., X_m]$ along the channel dimension. Input features $X = [X_1, X_2, ..., X_m]$ are processed through gOctConv to generate output features with arbitrary resolutions $Y = [Y_1, Y_2, ..., Y_n]$. Note that each scale in both X and Y can be defined with no limitation. Vanilla OctConv is designed as a replacement of standard convolution. Therefore, it can only process features within a stage. While in gOctConv, features with arbitrary scales from different stages of the network can be combined as input features with no post-processing. The output feature $Y_j \in Y$ with scale S_{yj} calculated from a part of feature $X_i \in X$ with scale S_{xi} can be written as follows:

$$Y_j = \begin{cases} Upsample[Conv(X_i)], S_{xi} < S_{yj}; \\ Conv(X_i), S_{xi} = S_{yj}; \\ Conv[Downsample(X_i)], S_{xi} > S_{yj}. \end{cases} \tag{1}$$

The standard convolutional operation within gOctConv can be replaced with various operations such as depthwise convolution, group convolution, and dilated convolution. In vanilla OctConv, each output features must be calculated from all input features with high/low scales. In gOctConv, the link between input and output features with arbitrary scales can be removed. For instance, output features with a scale in gOctConv can be obtained by only the input features with the same scale or with any other selected scales. This scheme gives a much more flexible designing space in gOctConv. The channels of each scale in gOctConv can be obtained by our proposed dynamic weight decay assisting pruning scheme.

2 Instances of gOctConvs

We give implementation details and analysis of different instances of our proposed gOctConvs used in the CSNet.

2.1 Vanilla OctConv

Our proposed gOctConv is a generalized form of vanilla OctConv. Vanilla OctConv can be viewed as an instance of gOctConv. The vanilla OctConv is used in ILBlocks of CSNet to interact among in-stage multi-scale features. Unlike the standard convolution that conducts the convolution operation at a single resolution, vanilla OctConv conducts the convolution operation at two resolutions. In this way, the vanilla OctConv can extract features at different frequencies to better capture fine details and global structures while reducing the computational complexity. Specifically, input features X are divided into two parts with different resolutions $[X_H, X_L]$ along the channel dimension. Input features $X = [X_H, X_L]$ are then processed through vanilla OctConv to generate output features with different resolutions $Y = [Y_H, Y_L]$ as follows:

$$\begin{aligned} Y_H &= Conv(X_H) + Upsample[Conv(X_L)]; \\ Y_L &= Conv(X_L) + Conv[Pool(X_H)]. \end{aligned} \tag{2}$$

High-resolution outputs Y_H are computed from high-resolution inputs X_H and up-sampled low-resolution inputs X_L, while low-resolution outputs Y_L are computed from down-sampled high-resolution inputs X_H and low-resolution inputs X_L. The high-resolution streams, processing features of high-resolution, are capable of capturing the fine details. The low-resolution streams can capture global structures and save FLOPs through processing low-resolution features. By changing the ratio of channels between high-resolution features and low-resolution features, the vanilla OctConv achieves a trade-off in the ability to process global structures and fine details, as well as the computational complexity.

Computational Cost Assuming input and output features have the same shape of $(height, width, channels) = (H, W, C)$, kernal size $= k$, and stride $= 1$. The FLOPs of a standard convolution is:

$$F_c = HWC^2k^2. \tag{3}$$

For vanilla OctConv, in common implementations, the height and width of low-resolution features are set to half size of high-resolution features. The channel numbers for X_H, X_L are set to C_H, C_L, where $C = C_H + C_L$. When ignoring low-cost down-sampling and up-sampling operations, the FLOPs of a vanilla OctConv operation is:

$$\begin{aligned} F_{vo} =& (HWC_HC_H + \frac{HWC_HC_L}{4} \\ &+ HWC_LC_H + \frac{HWC_LC_L}{4})k^2. \end{aligned} \tag{4}$$

Therefore, the FLOPs of a vanilla OctConv requires a minimum of 25% of the standard convolution when all features in vanilla OctConv is low-resolution features. Also, as reported on the original paper of vanilla OctConv [3], vanilla

OctConv requires about 60 % FLOPs to achieves the similar performance to standard convolution. Utilizing vanilla OctConv alone can not achieve an extremely light-weight model with about 0.2% complexity of large models.

2.2 Simplified Instance of gOctConv

The simplified instance of gOctConv (simplified gOctConv for short) in ILBlock indicates that each input channel corresponds to an output channel with the same resolution. All cross-scale operations are removed. And depthwise operation within each scale is utilized to further save computational cost. This simplified gOctConv in ILBlock requires less computational cost compared with the vanilla OctConv. In ILBlock, two simplified gOctConvs are following a vanilla OctConv. Thus, the simplified gOctConv also contains two scales. Input features $X = [X_H, X_L]$ are processed through the simplified gOctConv to generate output features with different resolutions $Y = [Y_H, Y_L]$ as follows:

$$\begin{aligned} Y_H &= DepthwiseConv(X_H); \\ Y_L &= DepthwiseConv(X_L). \end{aligned} \tag{5}$$

The cross-scale operations are removed in simplified gOctConv, since interacting features with different scales in every layer is unnecessary. Features from each resolution are processed through a depthwise conv to get the output features with the same resolution. In simplified gOctConv, each input channel corresponds to an output channel, saving a large amount of computational cost compared with vanilla OctConv.

Computational Cost Using the same features with the shape of (H, W, C) as used in the complexity analysis of vanilla OctConv. In simplified gOctConv, the kernal size and stride are set to k and 1 as well. The channel numbers for X_H, X_L are also set to C_H, C_L, where $C = C_H + C_L$. The FLOPs of a simplified gOctConv operation with kernel size k is:

$$F_{so} = (HWC_H + \frac{HWC_L}{4})k^2. \tag{6}$$

The simplified gOctConv requires $1/(C_L + C_H)$ FLOPs of the vanilla OctConv, saving a great amount of computational cost and parameters.

2.3 Cross-stage Instance of gOctConv

The cross-stage instance of gOctConv (cross-stage gOctConv for short) is used to fuse features from each stage of the feature extractor. Features with different scales are used as input. In our implementation, we choose to use features from the last three stages as a trade-off between model complexity and performance. Experimental results of utilizing more or less stages are shown in Section 3.2 of supplementary. The output scales can be arbitrarily set. In the cross-stage fusion

stage	output feature size	config [op, kernal size, stride]	
stage1	[224×224×10, 112×112×10]	[OctConv 3×3, 1 gOctConv 3×3, 1 gOctConv 3×3, 1]	×1
		[OctConv 1×1, 1 gOctConv 3×3, 1 gOctConv 3×3, 1]	×2
stage2	[112×112×20, 56×56×20]	[OctConv 3×3, 2 gOctConv 3×3, 1 gOctConv 3×3, 1]	×1
		[OctConv 1×1, 1 gOctConv 3×3, 1 gOctConv 3×3, 1]	×3
stage3	[56×56×40, 28×28×40]	[OctConv 3×3, 2 gOctConv 3×3, 1 gOctConv 3×3, 1]	×1
		[OctConv 1×1, 1 gOctConv 3×3, 1 gOctConv 3×3, 1]	×5
stage4	[28×28×40, 14×14×40]	[OctConv 3×3, 2 gOctConv 3×3, 1 gOctConv 3×3, 1]	×1
		[OctConv 1×1, 1 gOctConv 3×3, 1 gOctConv 3×3, 1]	×3

Table 1. Architecture for the feature extractor in CSNet×1.

part of CSNet, the output scale of the first cross-stage gOctConv is set to be the same with the larger scale in each stage, resulting in three scales in output features. The second cross-stage gOctConv outputs features with the largest scale. Each scale of input features are calculated through Equation (1) to get output features with different scales. The computational cost of the cross-stage gOctConv is similar to the vanilla OctConv except that arbitrary scales are used in cross-stage gOctConv. More detailed usage of the cross-stage gOctConv in CSNet is introduced in Section 3.2 of supplementary.

3 Details of CSNet

We introduce the implementation details of our proposed CSNet.

3.1 Details of Initial Feature Extractor

Table 1 in supplementary shows the details of the feature extractor in CSNet×1. The feature extractor of CSNet is composed of our proposed ILBlocks. Features

name	output feature size	config
gOctConv	[224×224×20, 112×112×40, 56×56×80, 28×28×80]	gOctConv kernel size 1×1 dilation 1
Parallel DilatedConvs	[224×224× (1+1+1+1+1), 112×112×(2+2+2+2+5), 56×56×(5+5+5+5+6), 28×28×(5+5+5+5+6)]	[DilatedConvs kernel size 3×3 dilations [1, 2, 4, 8, 16]] × scales
gOctConv	224×224×70	gOctConv kernel size 1×1 dilation 1
StandardConv	224×224×1	StandardConv kernel size 1×1 dilation 1

Table 2. Architecture for the cross-stage fusion part using four stages in CSNet×1.

with high/low resolutions within a stage are synchronously processed by IL-Blocks. The kernal size of vanilla OctConv in the first ILBlock of a stage is set to 3 while the results are set to 1. Beginning from stage2, the feature resolution is downsampled in each stage. Also, the stride of this conv is set to 2 for downsampling the feature resolution of this stage. Initially, we roughly double the channels of ILBlocks as the resolution decreases, except for the last two stages that have the same channel number. Unless otherwise stated, the channels for different scales in ILBlocks are set evenly. Models with channel numbers expanded to k times are denoted by CSNet-×k. Learnable channels of OctConvs then are obtained through the scheme as described in Sec. 3.3 of the manuscript.

3.2 Details of Cross-stage Fusion Part

Table 2 in supplementary shows the architecture for the cross-stage fusion part using four stages in CSNet×1. The output of each stage as shown in Table 1 of supplementary are combined as the input of the cross-stage gOctConv. The cross-stage gOctConv then outputs features with the larger scale in each stage, resulting features with four scales when the input uses features from four stages. Each scale of features are processed through a group of dilated convolutions to get the output features with the same resolution. The dilation rate within a group is set to 1, 2, 4, 8, 16, respectively. The output channel numbers for dilated convolutions in a group are evenly divided according to the input channels. The initial channel number for dilated convolution has limited impact since our proposed scheme will get the learned channel numbers after training. Another

Stages	4	3 to 4	2 to 4	1 to 4
FLOPs	0.58G	0.66G	0.72G	1.29G
Parms.	134k	139k	140k	177k
F_β	90.0	91.0	91.6	91.8

Table 3. Using different stages of the extractor in CSNet×2-L.

cross-stage gOctConv is used to take in features processed by parallel dilated convolutions and get features with the highest resolution. A standard convolution then outputs the prediction result.

Using different stages. As salient object detection requires a high output resolution. Utilizing more high resolution features results in better results. We now give the ablation study of using different number of stages. To minimize the impact of the initial channel numbers, we use the model CSNet×2-L with learned channels for gOctConvs. As shown in Table 3 of supplementary, using more stages from the feature extractor achieves better performance and causes larger model complexity. CSNet×2-L using stage 1to4 achieves 0.2% gain (91.8 vs 91.6) with 37k (177k vs 140k) more parameters compared with CSNet×2-L using stage 2to4. Therefore, in our work, as a trade-off between efficiency and performance, we choose to use the last three stages as the input of the cross-stage fusion part.

Comparision with ASPP module The parallel dilated convolutions used in CSNet is a simplified version of ASPP module in DeepLabv3. The main difference between our proposed cross-stage fusion (CSF) and the ASPP module is that CSF utilizes features from different stages while ASPP relay on the output features of the extractor. As shown in Table 3 of supplementary, only using features from the last stage of the extractor results in poor performance compared with using multi-stages features. We also compare the CSF with ASPP on large models such as ResNet. Comparing with the DeepLabv3 model that contains ASPP module, CSF+ResNet50 achieves better performance (94.0% vs 93.7%) with only 24% FLOPs (18.4G vs. 76.2G). DeepLabv3 eliminates downsampling operations in backbone to maintain a high feature resolution on high-levels of the backbone. While the CSF obtains both high and low resolution features accross different stages of the backbone, getting a high-resolution output while saving a large amount of computational cost.

4 Details of Channel Learning

We propose to get learnable channels for each scale in gOctConv by utilizing our proposed dynamic weight decay assisted pruning during training.

4.1 Pruning

Fixed pruning ratio/threshold. We follow [42] to use the weight of BatchNorm layer as the indicator of the channel importance. We modify the pruning method

	threshold/ratio	Parms.	F_β
Fixed threshold	1e-18	140K	91.5
	1e-19	140K	91.6
	1e-20	140K	91.6
	1e-21	141K	91.6
	1e-22	141K	91.5
Fixed ratio	38%	300k	87.7
	51%	400k	91.1

Table 4. Pruning with fixed threshold/ratio in CSNet×2-L.

in [42] by using a fixed threshold to eliminate channels instead of using a fixed pruning ratio. Table 4 in supplementary shows that pruning with fixed threshold achieves better performance with fewer parameters compared with using fixed pruning ratio. The reason behind this result is that different layers require different number channels. Therefore, pruning with a threshold can get a unique channel number for each layer.

Pruning thrshold. Fig. 6 in the manuscript shows the obvious gap between large and small bn values, indicating the pruning threshold 1e-20 is easy to choose. We also use different thresholds to prune channels as shown in Table 4 of supplementary. The pruned models have almost the same parameters and performance under different thresholds, showing that our dynamic weight decay assisted pruning method is not sensitive to the threshold.

4.2 Generalization of Dynamic Weight Decay

Dynamic weight decay maintains a stable weights distribution among channels, helping to improve the performance under same computational cost. Our work focus on the task of salient object detection (SOD). Still, the dynamic weight decay generalizes well on other tasks.

Semantic segmentation. We train the CSNet×1.8 on the task of semantic segmentation following common settings using Cityscapes Dataset. On the val set of Cityscapes Dataset, CSNet trained with dynamic weight decay achieves 70.1% mIoU, while the result trained with standard weight decay is 69.0%.

Classification. For classification, we verify the effectiveness of dynamic weight decay on CIFAR100 dataset using ResNet32. ResNet32 trained with dynamic weight decay achieves 71.5% top1 acc. on CIFAR100, while the result trained with standard weight decay is 70.6%.

5 Transfer Models to SOD Task.

We test the performance of models designed for other tasks in Sec. 4.2 of the manuscript. When transferring classification models to SOD task, the fc layer is

	Batch size	Run-time	FPS
CSNet×1	1	2.05ms	487
	10	1.19ms	836
	100	1.16ms	858
CSNet×1-L	1	1.54ms	647
	10	0.78ms	1272
	100	0.75ms	1328

Table 5. Run-time of CSNet on RTX TITAN GPU.

replaced with the conv 1×1 to output saliency maps. For segmentation models, the channel number of the output layer is changed from the number of classes to 1. All those models share the same training scheme as introduced in Sec. 4.1 of the manuscript.

6 Run-time on GPU.

We also test the run-time of CSNet on RTX TITAN GPU as shown in Table 5 of supplementary. Even the light-weighted CSNet is not designed for the powerful GPU platform, CSNet still achieves impressive performance on GPU. Due to the excessive computing resources of GPU, it is difficult for models with a single batch to fully show its speed. While larger batch size brings more parallel computation, thus improving the speed to 1328 FPS. Also, CSNet-L achieves almost $2 \times$ speedup with negligible performance drop compared with CSNet. To the best of our knowledge, our work is the first SOD model that achieves more than 1000 FPS speed. Still, as the CSNet is extremely low-cost, the IO of GPU becomes the new bottleneck of speed. We believe that with the optimization of IO in future hardware, CSNet can achieve faster speed.