# Supplementary Material for Global Feature Guided Local Pooling

Takumi Kobayashi
National Institute of Advanced Industrial Science and Technology
1-1-1 Umezono, Tsukuba, Japan
takumi.kobayashi@aist.go.jp

This material details the implementation of the models used in the paper.

## A. Embedding the proposed pooling into CNNs

We can straightforwardly replace the pooling layers *explicitly* embedded in CNNs, such as avg/max-pooling, with the proposed pooling layer of the same pool size and the striding step size; see Fig. A*ii*&*iii* in which *skip*-pooling is replaced with the proposed one. Those explicit pooling layers are found in such as VGG models [8] and the second layer (of max-pooling) in ResNet models [3, 10]. On the other hand, in the ResNet models, the *strided* convolution *implicitly* contains a pooling to downsize the feature map, and the proposed pooling is also applicable to it as follows.

### A.1. Strided convolution

The *strided* convolution (Fig. A*i*) is decomposed into the ordinary convolution without striding and the *skip*-pooling as shown in Fig. A*ii*. The *skip*-pooling simply picks up one neuron activation in the receptive field and pass it to the next layer; for example, $(2, 2)$-striding in convolution corresponds to the *skip*-pooling of $2 \times 2$ pool size with $(2, 2)$ striding step size, which picks up the top-left corner neuron activation in the $2 \times 2$ local region. The *skip*-pooling is then subject to the replacement by the proposed pooling (Fig. A*iii*).

### A.2. ResBlock

The ResNet models [3, 10] stack the residual blocks, denoted by ResBlocks, containing several convolution layers and a shortcut path. For downsizing a feature map, the ResBlock is implemented as shown in Fig. B*i* which leverages *strided* convolution operations in the both paths. By decomposing the *strided* convolutions, without changing the output, the ResBlock results in the form that explicitly applies the *skip*-pooling (Fig. B*ii*). Then, the proposed pooling substitutes for the *skip*-pooling lying at the last layer of the ResBlock as depicted in Fig. B*iii*.

## B. Embedding the comparison modules into VGG

In Sec. 5.1.3, the proposed method is compared to the modules of NiN [7], ResNiN [7, 3] and squeeze-and-excitation (SE) [4]. Those modules are embedded into the VGG model as shown in Fig. C. It should be noted that, for fair comparison, (1) they are inserted just before the max-pooling and (2) they contain the same number of additional parameters by using the same structure of MLP as ours (Fig. A*iii*); the NiN and ResNiN applies the ReLU at the last activation in stead of sigmoid.

## C. Training procedure

All the CNN models are implemented by using MatConvNet [9] and for fair comparison they are trained from scratch on NVIDIA Tesla P40.

The VGG-based models [8, 6] are trained by following the training procedure presented in MatConvNet toolbox which provides a good practice to train those models of high performance while requiring smaller number of training epochs; SGD is applied to train the CNNs with the batch size of 64, the momentum of 0.9, the weight decay of 0.0005, and the learning rate reducing from 0.1 to 0.0001 linearly in a log-scale over 20 training epochs.

The ResNet-based models [3, 10] are also trained based on the approach presented in [3]; we apply the batch size of 256 and SGD with momentum of 0.9 and weight decay of 0.0001, while the learning rate starts from 0.1 and is divided by 0.1 every 30 epochs throughout 100 training epochs.

Practically, we can obtain the performance close to the reported ones [1, 2]; for VGG-16, ours are 27.94% (top-1) and 9.25% (top-5) while the reported ones are 28.3% (top-1) and 9.5% (top-5); for ResNet-50, ours are 23.53% (top-1) and 7.00% (top-5) while the reported ones are 24.6% and 7.7% (top-5); for ResNeXt-50, ours are 22.69% (top-1) and 6.65% (top-5) while the reported ones are 22.60% (top-1) and 6.49% (top-5).

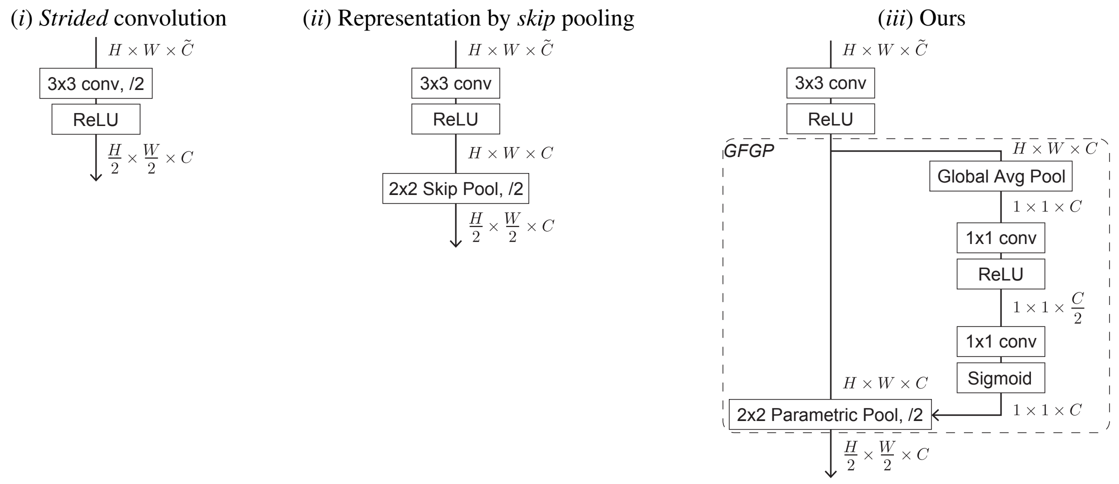
(i) Strided convolution
$H \times W \times \tilde{C}$
3x3 conv, /2
ReLU
$\frac{H}{2} \times \frac{W}{2} \times C$
(ii) Representation by skip pooling
$H \times W \times \tilde{C}$
3x3 conv
ReLU
$H \times W \times C$
2x2 Skip Pool, /2
$\frac{H}{2} \times \frac{W}{2} \times C$
(iii) Ours
$H \times W \times \tilde{C}$
3x3 conv
ReLU
GFGP
$H \times W \times C$
Global Avg Pool
$1 \times 1 \times C$
1x1 conv
ReLU
$1 \times 1 \times \frac{C}{2}$
1x1 conv
Sigmoid
$H \times W \times C$
$1 \times 1 \times C$
2x2 Parametric Pool, /2
$\frac{H}{2} \times \frac{W}{2} \times C$

Figure A. *Strided* convolution of $(2,2)$ step size is replaced by the proposed pooling method. '/2' indicates the striding step size of $(2,2)$ in the convolution or pooling operation in which the filter or pooling size is shown in the form of '$* \times *$'. The size of feature map passed through layers is also denoted by '$Height \times Width \times Channel$'. The strided convolution (*i*) corresponds to the form (*ii*) explicitly using the skip pooling. The parametric pooling in (*iii*) is implemented such as by Eq.(7) (in the submitted paper) with $\boldsymbol{\rho} = \mathbf{0}$. Note that the Batch-Normalization [5] is applied right after each convolution layer.

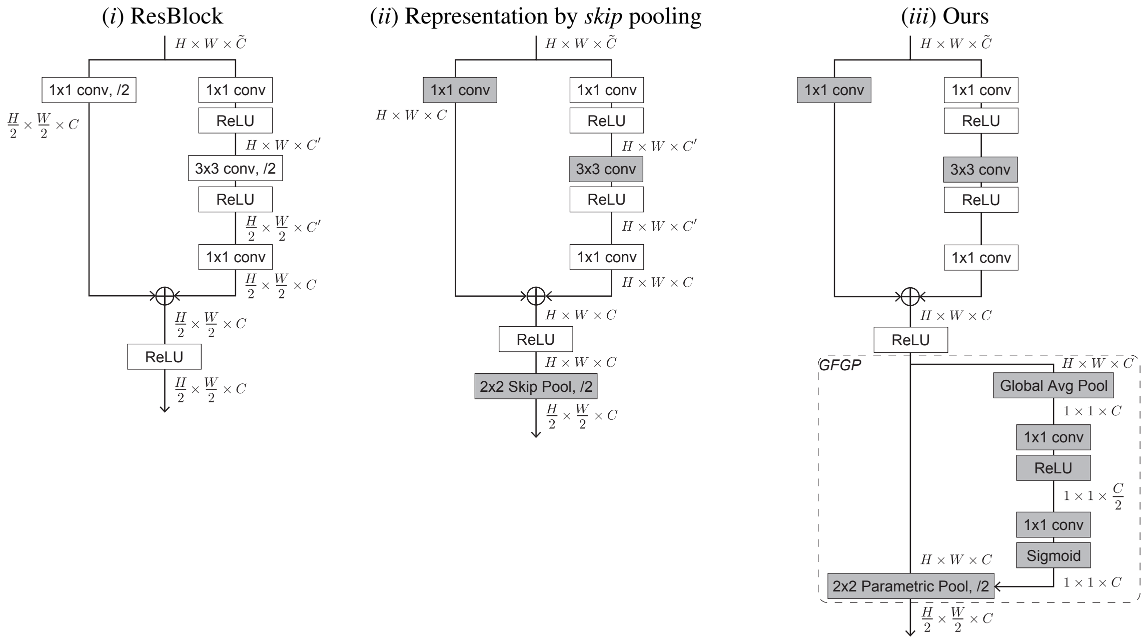

Figure B. ResBlock that downsizes an input feature map is modified by embedding the proposed pooling. In (*ii*, *iii*), the gray-colored box indicates the modified or added layer, compared to the original architecture (*i*).

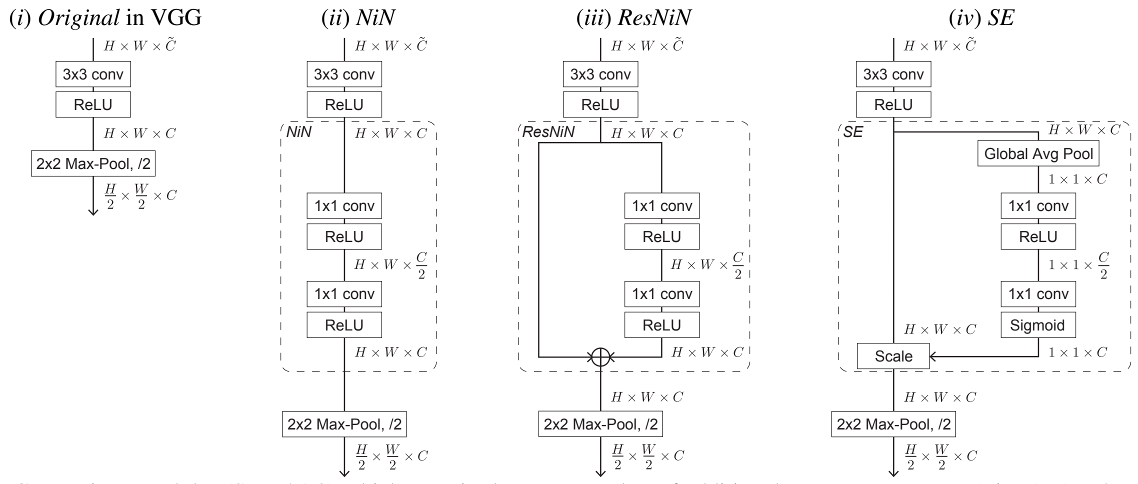

Figure C. Comparison modules (Sec. 5.1.3) which contain the same number of additional parameters as ours (Fig. A*iii*). Those modules are inserted just before the max-pooling since they do not render pooling functionality.

## References

[1] http://www.vlfeat.org/matconvnet/pretrained/.

[2] https://github.com/albanie/mcnResNeXt.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[4] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Journal of Machine Learning Research*, 37:448–456, 2015.

[6] Takumi Kobayashi. Analyzing filters toward efficient convnets. In *CVPR*, pages 5619–5628, 2018.

[7] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[9] Andrea Vedaldi and Karel Lenc. MatConvNet – convolutional neural networks for matlab. In *ACM MM*, 2015.

[10] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 5987–5995, 2017.