Differentiable Learning-to-Group Channels via Groupable Convolutional Neural Networks

Appendices

A. Extension

In the main text, we employ an illustrative example when $C^{\text{in}} = C^{\text{out}}$ to explain the construction of binary relationship matrix U. In fact, DGConv can be generalized to case where $C^{\text{in}} \neq C^{\text{out}}$ by using the following simple transformations.

GroupUp. When a certain DGConv layer conducts convolution by increasing the channel size by r times, *i.e.* $rC^{\text{in}} = C^{\text{out}}$, we first construct $\tilde{U} \in \{0,1\}^{C^{\text{in}} \times C^{\text{in}}}$ that is the same as the main text. We can derive the relationship matrix U as follow:

$$U = \tilde{U}\tilde{I_u}, \quad \tilde{I_u} = [\underbrace{I_{in}, \cdots, I_{in}}_{r's}] \tag{1}$$

where $\tilde{I_u} \in \{0,1\}^{C^{\text{in}} \times C^{\text{out}}}$ is a matrix concatenated by identity matrices $I_{in} \in \{0,1\}^{C^{\text{in}} \times C^{\text{in}}}$.

GroupDown. When a certain DGConv layer conducts convolution by decreasing the channel size by r times, *i.e.* $C^{\text{in}} = rC^{\text{out}}$. Similar to GroupUp, we firstly construct $\tilde{U} \in \{0,1\}^{C^{\text{out}} \times C^{\text{out}}}$. Then relationship matrix U can be computed by

$$U = \tilde{I_d}\tilde{U}, \quad \tilde{I_d} = [\underbrace{I_{out}, \cdots, I_{out}}_{r's}] \tag{2}$$

where $\tilde{I_d} \in \{0,1\}^{C^{\text{in}} \times C^{\text{out}}}$ is a matrix concatenated by identity matrices $I_{out} \in \{0,1\}^{C^{\text{out}} \times C^{\text{out}}}$.

B. Back-Propagation of DGConv

Before introducing back-propagation of a DGConv layer, we first describe the parameterization process of DGConv. As shown in Fig.1, the convolution kernel $\tilde{w}$ is produced by multiplying a binary matrix U on a element basis. U can be decomposed into multiple sub-matrixes $U_k, k = 1, \cdots, K$ by Kronecker product and each submatrix U_k has diagonal of ones and non-diagonal of g_k. Since g is obtained by a sign function that is not differentiable and the gradient is manually designed, we treat g as a learnable parameter and back-propagate the gradient wrt. g. Here we write down forward computation as follows:

Figure 1. Illustration of parameterization process of a DGConv layer. f and o indicate the input and output of a DGConv layer. $*$ denotes convolution.

$$o_{ij} = \sum_{m=0}^{k-1}\sum_{n=0}^{k-1} f_{(i+m)(j+n)}(U \odot \omega_{mn}), \tag{3}$$

$$U = \bigotimes_{k=1}^{K} U_k, \quad U_k = \begin{bmatrix} 1 & g_k \\ g_k & 1 \end{bmatrix}. \tag{4}$$

where $f_{(i+m)(j+n)} \in \mathbb{R}^{N \times C^{\text{in}}}$ represents the hidden units of the input feature map F, $\omega_{mn} \in \mathbb{R}^{C^{\text{in}} \times C^{\text{out}}}$ represents the convolution weights with kernel size $k \times k$ and o_{ij} denotes the output of a DGConv layer.

Denote a top-down gradient $\frac{\partial \mathcal{L}}{\partial o_{ij}}$ as d_{ij}, where $i \in [H]$ and $j \in [W]$. For simplicity, we consider the case where $C^{\text{in}} = C^{\text{out}} = C$. We use chain rule to back-propagate the

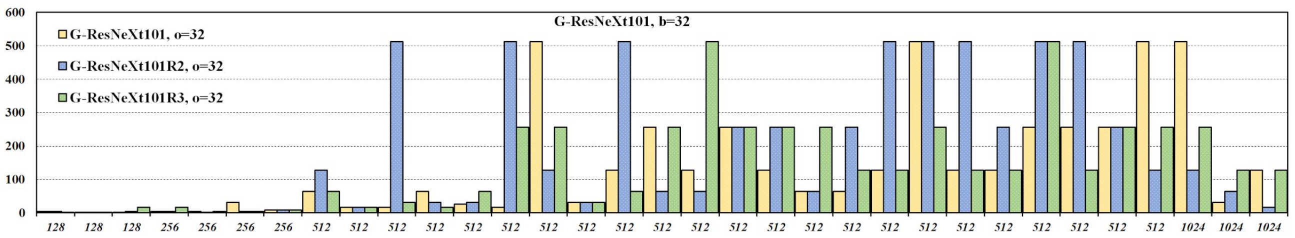

Figure 2. Learned number of groups for each DGConv layer in G-ResNeXt101 $b = 32$. The x-axis denotes the number of channels in DGConv layers under network's input to output direction, and the y-axis is the group number of corresponding layers.

gradient of loss $\mathcal{L}$ through the above transformations,

$$\frac{\partial \mathcal{L}}{\partial \omega_{mn}} = \left(\sum_{i,j}^{H,W} {f_{(i+m)(j+n)}}^{\top} d_{ij} \right) \odot U, \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial g_k} = \frac{\partial \mathcal{L}}{\partial U} \bullet \frac{\partial U}{\partial g_k} \tag{6}$$

where $\bullet$ denotes the Frobenius inner product. Now, we provide detailed derivations of $\frac{\partial \mathcal{L}}{\partial U}$ and $\frac{\partial U}{\partial g_k}$. The first term can be derived as follow:

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{m=0,n=0}^{k-1,k-1} \omega_{mn} \odot \left(\sum_{i,j}^{H,W} {f_{(i+m)(j+n)}}^{\top} d_{ij} \right) \tag{7}$$

To compute the derivative of U wrt. g_k, we refer to the following remark.

Remark 1 *For $A \in \mathbb{R}^{m_1 \times n_1}, B \in \mathbb{R}^{m_2 \times n_2}$, then*

$$B \otimes A = S_{m_1,m_2}(A \otimes B) S_{n_1,n_2} \tag{8}$$

where $S_{m,n} = \sum_{i=1}^{m}(e_i{}^{\top} \otimes I_n \otimes e_i) = \sum_{j=1}^{n}(e_j \otimes I_m \otimes e_j{}^{\top})$ is the perfect shuffle permutation matrix. e_i denotes the i-th canonical vector that is the vector with 1 in the i-th coordinate and 0 elsewhere. I_n is an identity matrix with size of n.

Denote $\tilde{U}_k = \bigotimes_{t=k+1}^{K} U_t \bigotimes_{t=1}^{k-1} U_t$, by Remark 1, U can be reformulated as below:

$$U = \left(S_{\frac{C}{2},2}^{k-1} \right) U_k \otimes \tilde{U}_k \left(S_{\frac{C}{2},2}^{k-1} \right)^{\top} \tag{9}$$

By Eqn.(9), it can be inspected that

$$\frac{\partial U}{\partial g_k} = \left(S_{\frac{C}{2},2}^{k-1} \right) \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \tilde{U}_k \left(S_{\frac{C}{2},2}^{k-1} \right)^{\top} \tag{10}$$

By putting Eqn.(6), (7) and (10) together, we can compute the gradient of the loss $\mathcal{L}$ with respect to g.

Architecture	top-1	top-5
ResNet50	70.95	89.4
ResNeXt50	71.06	89.6
G-ResNeXt50	**72.58**	**90.6**

Table 1. Results of action recognition in Kinetics-400 using TSN.

C. Transferability

Action Recognition in Video We evaluate the effectiveness of DGConv on action recognition task on Kinetics-400 dataset [1]. The network used here is temporal segment network(TSN) [4]. In the training phase, one video is divided into 5 equal-length segments, and one frame is randomly sampled from each segment. The network take frames as input, then the outputs at the final layer are averaged and passed into Softmax layer giving the classification result of this video. During the evaluation, 25 frames are taken uniformly from each video. The data argumentation, hyper-parameters and training process all follow TSN's setting. The batch size is 128 (32 videos per GPU). We use SGD with initial learning rate 0.001 for optimization. The learning rate is decayed by ratio 0.1 at step 50k, 80k, and the training stops at step 100k. We give the results of three backbones(ResNet50, ResNeXt50, G-ResNeXt50) in Table. 1. For G-ResNeXt50, the number of groups in GConv is fixed after pretrained from ImageNet.

Object Detection We use Faster R-CNN [3]+FPN as detection framework, and evaluate the results on COCO [2]. Both the training and evaluation processes follow Caffe2-Detectron [5]. During training, each GPU takes two images as input, and the total batch size is 16. The optimizer we use is SGD. The initial learning rate(lr) is 0.02, and the momentum is 0.9. The total training step is 180k, and the lr is decayed by ratio 0.1 at step 120k and 160k. We report the results of three backbones(ResNet50, ResNeXt50, G-ResNeXt50) in Table. 2

D. Reproducibility

Fig. 2 depicts group number distribution of reproducibility experiments. As we can see, although the learned models have similar performance (showed in Section 4), their

Backbone	AP	$\mathrm{AP}_{.5}$	$\mathrm{AP}_{.75}$	AP_l	AP_m	AP_s
ResNet50	37.9	59.3	41.1	49.9	41.1	21.5
ResNeXt50	38.4	60.2	41.4	49.5	41.3	**22.8**
G-ResNeXt50	**38.9**	**60.7**	**42.3**	**50.4**	**42.1**	**22.8**

Table 2. Results of object detection in COCO using Faster R-CNN+FPN.

group number distributions vary slightly, implying that the optimal grouping strategy is not unique. DGConv expresses a flexibility to learn among these grouping strategies.

References

[1] Kay Will et al. The kinetics human action video dataset. 2017. 2

[2] Lin Tsung-Yi et al. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755, 2014. 2

[3] Ren Shaoqing et al. Faster r-cnn: Towards real-time object detection with region proposal networks. pages 91–99, 2015. 2

[4] Wang Limin et al. Temporal segment networks for action recognition in videos. 2018. 2

[5] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018. 2