Structured Sparsification with Joint Optimization of Group Convolution and Channel Shuffle (Supplementary material)

Xin-Yu Zhang[*1] **Kai Zhao**[*1] **Taihong Xiao**[2] **Ming-Ming Cheng**[1] **Ming-Hsuan Yang**[2]

[1]TKLNDST, CS, Nankai University
[2]University of California, Merced

A STRUCTURED REGULARIZATION IN GENERAL FORM

Generally, we can relax the constraints that both C^{in} and C^{out} are powers of 2, and assume both C^{in} and C^{out} have many factors of 2. Under this assumption, the potential candidates of cardinality are still restricted to powers of 2. Specifically, if the *greatest common divisor* of C^{in} and C^{out} can be factored as

$$\gcd(C^{\text{in}}, C^{\text{out}}) = r = 2^u \cdot z, \tag{1}$$

```python
import numpy as np

def struc_reg(dim1, dim2, level=None, power=0.5):
    r"""
    Compute the structured regularization matrix.

    Args::
        dim1 (int): number of output channels.
        dim2 (int): number of input channels.
        level (int or None): current group level.
                Specify 'None' if the cost matrix is desired.
        power (float): decay rate of the reg. coefficients.

    Return::
        Structured regularization matrix.
    """
    reg = np.zeros((dim1, dim2))
    assign_val(reg, 1., level, power)
    return reg

def assign_val(arr, val, level, power):
    dim1, dim2 = arr.shape
    if dim1 % 2 != 0 or dim2 % 2 != 0 or level == 0:
        return
    else:
        _l = None if level is None else level - 1
        arr[dim1//2:, :dim2//2] = val
        arr[:dim1//2, dim2//2:] = val
        assign_val(arr[dim1//2:, dim2//2:], val*power, _l, power)
        assign_val(arr[:dim1//2, :dim2//2], val*power, _l, power)
```

[*]Xin-Yu Zhang (xinyuzhang@mail.nankai.edu.cn) and Kai Zhao contribute equally to this work.

Supplement for the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI 2021).

Algorithm 1: Dynamically adjust λ

Initialize $\lambda_1 = 0$, $r_0 = 0$, $N =$ #epochs, $r =$ target sparsity
for $t := 1$ *to* N **do** *train for 1 epoch*
    Determine the current group levels g;
    Compute the current sparsity by Eq. (2) and (3)
    if $r_t - r_{t-1} < \frac{r - r_{t-1}}{N - t + 1}$ **then**
        $\lambda_{t+1} = \lambda_t + \Delta_\lambda$
    else if $\boldsymbol{S}_t > r$ **then**
        $\lambda_{t+1} = \lambda_t - \Delta_\lambda$
end

where z is an odd integer, then the potential candidates of the group level g are $\{1, 2, \cdots, u + 1\}$. For example, if the minimal u is 4 among all convolutional layers[1], the potential candidates of cardinality are $\{1, 2, 4, 8, 16\}$, giving adequate flexibility of the compressed model. The structured regularization and the relationship matrix corresponding to each group level are designed in a similar way. For clarity, we provide an exemplar implementation based on the NumPy library.

B DYNAMIC PENALTY ADJUSTMENT

As the desired compression rate is customized according to user preference, manually choosing an appropriate regularization coefficient λ in Eq. (7) of the main text for each experimental setting is extremely inefficient. To alleviate this issue, we dynamically adjust λ based on the sparsification progress. The algorithm is summarized in Alg. 1.

Concretely, after the t^{th} training epoch, we first determine the current group level g_t of each convolutional layer according to Eq. (8) in the main text. Then, we define the model sparsity based on the reduction of model parameters. For the l^{th} convolutional layer, the number of parameters is reduced by a factor of $2^{g_t^l - 1}$, where $2^{g_t^l - 1}$ is the cardinality. Thus, the original number of parameters and the reduced one are given by

$$p^l = C^l \times C^{l+1} \times k^l \times k^l, \quad \hat{p}_t^l = \frac{p^l}{2^{g_t^l - 1}}. \tag{2}$$

Here, C^l and k^l denote the input channel number and the kernel size of the l^{th} convolutional layer. Therefore, the current model sparsity is calculated as

$$r_t = \frac{\sum_l \hat{p}_t^l}{\sum_l p^l}. \tag{3}$$

Afterwards, we assume the model sparsity grows linearly, and calculate the expected sparsity gain. If the expected sparsity gain is not met, *i.e.,*

$$r_t - r_{t-1} < \frac{r - r_{t-1}}{N - t + 1}, \tag{4}$$

where N is the total training epoch number and r is the target sparsity, we increase λ by Δ_λ. If the model sparsity exceeds the target, *i.e.,* $r_t > r$, we decrease λ by Δ_λ.

In all experiments, the coefficient is initialized from $\lambda_1 = 0$ and Δ_λ is set to 2×10^{-6}.

C EXPERIMENTAL DETAILS

In this section, we provide more results and details of our experiments. We provide the loss and accuracy curves along with the performance after each stage in appendix C.1, and analyze the compressed model architectures in appendix C.2.

C.1 TRAINING DYNAMICS

We first provide the pre- and post-compression accuracy along with the finetune accuracy of our pipeline in Tab. 1. During compression, we use a *binary search* to decide the threshold p of the grouping criteria (Eq. (8) in the main text) so that the

[1]The standard DenseNet [Huang et al., 2017] family satisfies this condition.

Table 1: Performance along the timeline of our approach. The evaluation is performed on the ImageNet dataset.

Backbone	**ResNet-50**			**ResNet-101**			**DenseNet-201**	
Compression Rate	35%	65%	85%	40%	65%	80%	38%	60%
Pre-compression Acc.	69.07	66.36	64.30	69.56	67.13	64.20	69.10	66.26
Post-compression Acc.	60.92	42.78	8.82	65.78	58.63	18.57	66.15	17.35
Finetune Acc.	76.82	75.10	72,47	78.16	77.62	75.73	77.43	75.86
threshold p	0.127	0.115	0.125	0.095	0.090	0.103	0.098	0.115

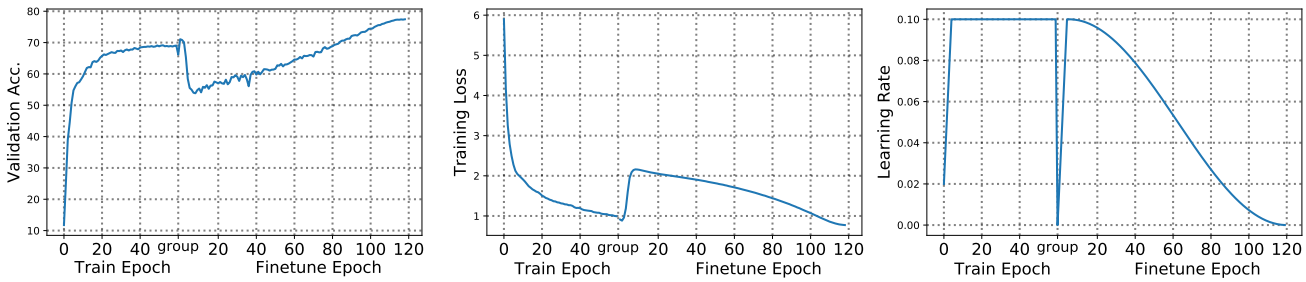


Figure 1: Training dynamics of the full structured sparsification pipeline. We plot the training and finetune curves of the DenseNet-201 backbone with a compression rate of 38%. At the end of the 60^{th} epoch of the training stage, we compress the network following our criteria. Then, we finetune for 120 epochs to recover performance.

network can be compressed at the desired compression rate. The searched thresholds are also illustrated. Apart from this, we further provide the training and finetune curves in Fig. 1. In the training stage, the accuracy gradually increases till saturation, and then the compression leads to a slight performance drop. Finally, the performance is recovered in the finetune stage.

C.2 COMPRESSED ARCHITECTURES

We illustrate the compressed architectures by showing the cardinality of each convolution layer in Fig. 2 and Fig. 3. Note that our method is applied to all convolution operators, *i.e.,* both 3×3 convolutions and 1×1 convolutions, so a high compression

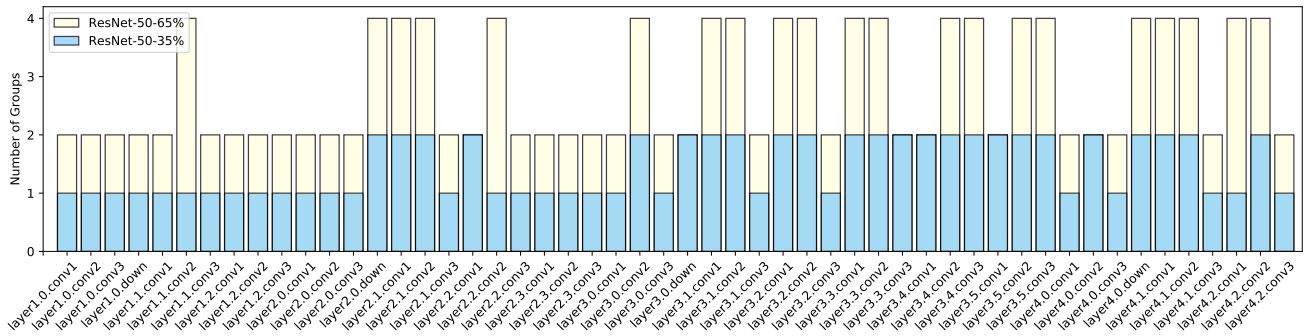


Figure 2: Learned cardinalities of the ResNet-50 backbone with the compression rates of 35% and 65%.

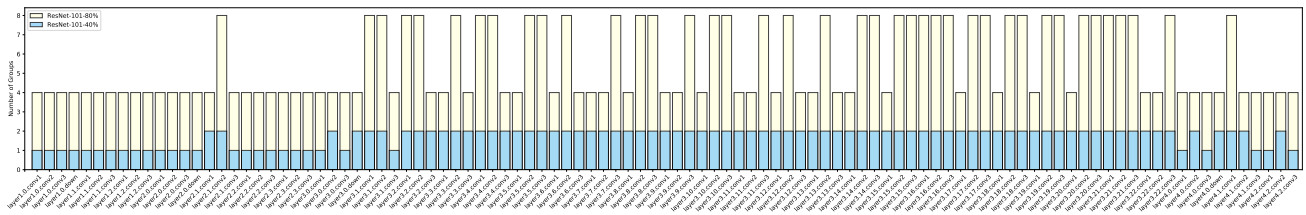


Figure 3: Learned cardinalities of the ResNet-101 backbone with the compression rates of 40% and 80%.

Table 2: Confusion matrices of the adjacent GroupConvs. Here, the neuron connectivity between "Layer4-Bottleneck1-conv1" and "Layer4-Bottleneck1-conv2" of the ResNet-50-85% model is demonstrated. Left: the learned neuron connectivity; Right: the neuron connectivity of the ShuffleNet [Zhang et al., 2018].

	G1	G2	G3	G4	G5	G6	G7	G8
G1	6	6	10	8	9	6	13	6
G2	9	8	7	9	11	8	4	8
G3	11	8	11	6	4	8	7	9
G4	16	9	5	9	10	4	6	5
G5	7	9	7	7	8	10	9	7
G6	5	7	10	6	7	11	7	11
G7	4	8	7	14	6	8	7	10
G8	6	9	7	5	9	9	11	8

	G1	G2	G3	G4	G5	G6	G7	G8
G1	8	8	8	8	8	8	8	8
G2	8	8	8	8	8	8	8	8
G3	8	8	8	8	8	8	8	8
G4	8	8	8	8	8	8	8	8
G5	8	8	8	8	8	8	8	8
G6	8	8	8	8	8	8	8	8
G7	8	8	8	8	8	8	8	8
G8	8	8	8	8	8	8	8	8

rate, *e.g.,* 80%, can be achieved. As discussed in Sec. 4.4, the learned cardinality distribution is prone to uniformity, but there are still certain patterns. For example, shallow layers are relatively more difficult to be compressed. A possible explanation is that shallow layers have fewer filters, so a large cardinality will inevitably eliminate the communication between certain groups. Moreover, we observe 3×3 convolutions are generally more compressible than 1×1 convolutions. This is intuitive as 3×3 convolutions have more parameters, thus leading to heavier redundancy.

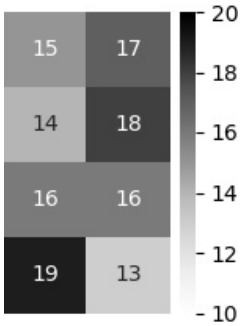
Furthermore, we illustrate the learned neuron connectivity and compare with the ShuffleNet [Zhang et al., 2018] counterpart. Here, we consider the channel permutation between two group convolutions (GroupConvs) and demonstrate the connectivity via the *confusion matrix*. Specifically, we assume the first GroupConv is of cardinality G_1 and the second of G_2, then the confusion matrix $\boldsymbol{D}$ is a $G_1 \times G_2$ matrix with $D_{i,j}$ denoting the number of channels that come from the i^{th} group of the first GroupConv and belong to the j^{th} group of the second.

In Tab. 2, we can see that the inter-group communication is guaranteed as there are connections between every two groups. Furthermore, the learnable channel shuffle scheme is more flexible. The ShuffleNet [Zhang et al., 2018] scheme uniformly partitions and distributes channels within each group, while our approach allows small variations of the number of connections for each group. In this way, the network can itself control the information flow from each group by customizing its neuron connectivity. More examples can be found in Fig. 4. All models illustrated in this section are trained on the ImageNet dataset.

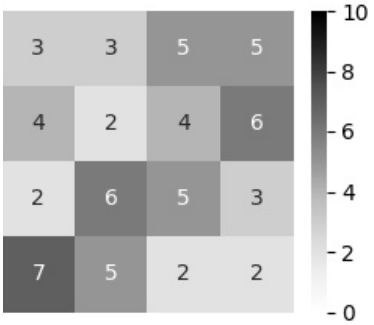
References

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

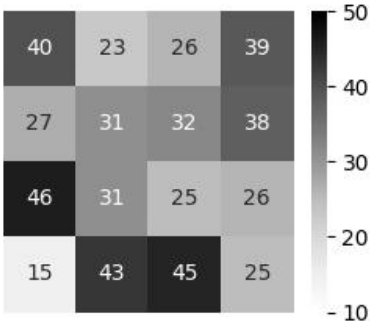
Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018.



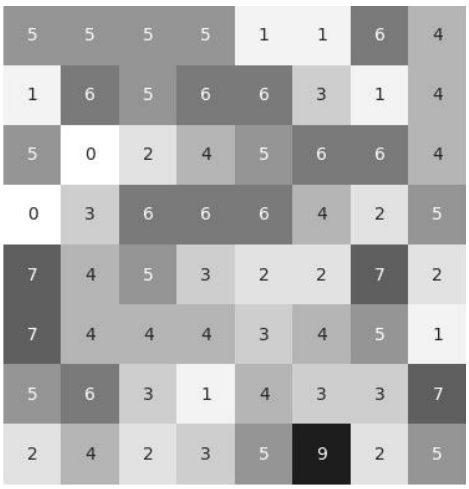
(a) DenseNet-201-60%
Block4-Layer24-conv1-2



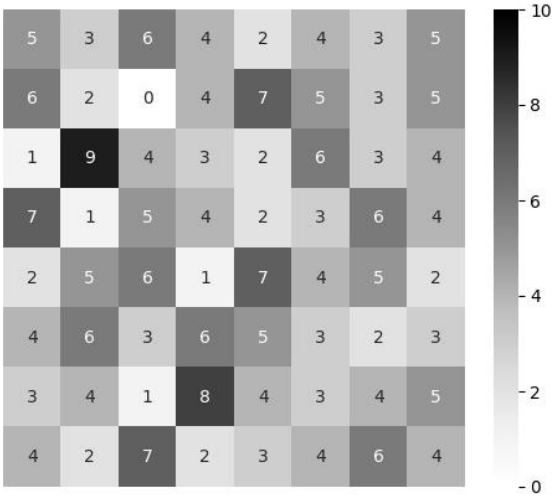
(b) ResNet-50-85%
Layer1-Bottleneck1-conv2-3



(c) ResNet-101-80%
Layer4-Bottleneck2-conv2-3



(d) ResNet-50-85%
Layer3-Bottleneck4-conv1-2



(e) ResNet-101-80%
Layer3-Bottleneck1-conv1-2

Figure 4: More examples of the confusion matrices.