# Supplementary Materials of Group Whitening: Balancing Learning Efficiency and Representational Capacity

Lei Huang[1,3] Yi Zhou[2] Li Liu[3] Fan Zhu[3] Ling Shao[3]
[1]SKLSDE, Institute of Artificial Intelligence, Beihang University, Beijing, China
[2]MOE Key Laboratory of Computer Network and Information Integration, Southeast University, China
[3]Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE

**Algorithm 1** The forward pass of group whitening.

```
1: Input: a input sample $\mathbf{x} \in \mathbb{R}^d$.
2: Hyperparameters: $\epsilon$, group number $g$.
3: Output: $\hat{\mathbf{x}} \in \mathbb{R}^d$.
4: Group division: $\mathbf{X}_G = \Pi(\mathbf{x}; g) \in \mathbb{R}^{g \times c}$.
5: $\mu = \frac{1}{c}\mathbf{X}_G \mathbf{1}$.
6: $\mathbf{X}_C = \mathbf{X}_G - \mu \mathbf{1}^T$.
7: $\Sigma = \frac{1}{c}\mathbf{X}_C \mathbf{X}_C^T + \epsilon \mathbf{I}$.
8: Calculate whitening matrix: $\Sigma^{-\frac{1}{2}} = \psi^f(\Sigma)$.
9: $\widehat{\mathbf{X}}_G = \Sigma^{-\frac{1}{2}} \mathbf{X}_C$.
10: Inverse group division: $\hat{\mathbf{x}} = \Pi^{-1}(\widehat{\mathbf{X}}_G) \in \mathbb{R}^d$.
```

**Algorithm 2** The corresponding backward pass of Algorithm 1.

```
1: Input: gradient of a sample: $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}} \in \mathbb{R}^d$, and auxiliary data
   from respective forward pass: (1) $\mathbf{X}_C$; (2) $\Sigma^{-\frac{1}{2}}$.
2: Output: gradient with respect to the input: $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} \in \mathbb{R}^d$.
3: Group division: $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{X}}_G} = \Pi(\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}}; g) \in \mathbb{R}^{g \times c}$.
4: $\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}} = \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{X}}_G} \mathbf{X}_C^T$.
5: Calculate gradient with respect to the covariance matrix: $\frac{\partial \mathcal{L}}{\partial \Sigma} = \psi^b(\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})$.
6: $\mathbf{f} = \frac{1}{c}\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{X}}_G}\mathbf{1}$.
7: $\frac{\partial \mathcal{L}}{\partial \mathbf{X}_G} = \Sigma^{-\frac{1}{2}}(\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{X}}_G} - \mathbf{f}\mathbf{1}^T) + \frac{1}{c}(\frac{\partial \mathcal{L}}{\partial \Sigma} + \frac{\partial \mathcal{L}}{\partial \Sigma}^T)\mathbf{X}_C$.
8: Inverse group division: $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \Pi^{-1}(\frac{\partial \mathcal{L}}{\partial \mathbf{X}_G}) \in \mathbb{R}^d$.
```

## 1. Algorithms

The forward pass of the proposed group whitening (GW) method is shown in Algorithm 1, and its corresponding backward pass is shown in Algorithm 2.[1] Note that we need to specify the method for calculating the whitening matrix $\Sigma^{-\frac{1}{2}}\psi^f(\Sigma)$ in Line 8 of Algorithm 1, as well as its backward operation $\frac{\partial \mathcal{L}}{\partial \Sigma} = \psi^b(\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})$ shown in Line 5 of Algorithm 2. As stated in the submitted paper, we use zero-phase component analysis (ZCA) whitening and its efficient approximation by Newton's iteration ('ItN') [5]. Here, we provide the details.

[1]For GW, we also use the extra learnable dimension-wise scale and shift parameters, like BN [6]. We omit this in the algorithms for simplicity.

**ZCA whitening.** ZCA whitening [3] calculates the whitening matrix by eigen decomposition as: $\Sigma^{-\frac{1}{2}} = \psi^f_{ZCA}(\Sigma) = \mathbf{D}\Lambda^{-\frac{1}{2}}\mathbf{D}^T$, where $\Lambda = \mathrm{diag}(\sigma_1, \dots, \sigma_d)$ and $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvalues and associated eigenvectors of $\Sigma$, *i.e.*$\Sigma = \mathbf{D}\Lambda\mathbf{D}^T$.

The corresponding backward operation $\frac{\partial \mathcal{L}}{\partial \Sigma} = \psi^b_{ZCA}(\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})$ is as follows:

$$\frac{\partial \mathcal{L}}{\partial \Lambda} = \mathbf{D}^T(\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})\mathbf{D}(-\frac{1}{2}\Lambda^{-3/2}) \quad (1)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{D}} = (\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}} + (\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})^T)\mathbf{D}\Lambda^{-1/2} \quad (2)$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma} = \mathbf{D}\{(\mathbf{K}^T \odot (\mathbf{D}^T\frac{\partial \mathcal{L}}{\partial \mathbf{D}})) + (\frac{\partial \mathcal{L}}{\partial \Lambda})_{diag}\}\mathbf{D}^T, \quad (3)$$

where $(\frac{\partial \mathcal{L}}{\partial \Lambda})_{diag}$ sets the off-diagonal elements of $\frac{\partial \mathcal{L}}{\partial \Lambda}$ as zero.

**'ItN' whitening.** 'ItN' whitening [5] calculates the whitening matrix by Newton's iteration as: $\Sigma^{-\frac{1}{2}} = \psi^f_{ItN}(\Sigma) = \frac{\mathbf{P}_T}{\sqrt{tr(\Sigma_d)}}$, where $tr(\Sigma_d)$ indicates the trace of $\Sigma_d$ and $\mathbf{P}_T$ is calculated iteratively as:

$$\begin{cases} \mathbf{P}_0 = \mathbf{I} \\ \mathbf{P}_k = \frac{1}{2}(3\mathbf{P}_{k-1} - \mathbf{P}^3_{k-1}\Sigma^N_d), \quad k = 1, 2, ..., T. \end{cases} \quad (4)$$

Here, $\Sigma^N_d = \Sigma_d / tr(\Sigma_d)$.

The corresponding backward operation $\frac{\partial \mathcal{L}}{\partial \Sigma} =$

```
def GroupWhitening (X, gamma, beta, g, T=5, eps=1e-5):
    # X input feature with size [m, d] or [m, d, H, W]
    # gamma, beta: the learnable affine
    # g: the group number of group whitening
    # T: the iteration number of Newton's iteration
    size = X.size()
    X_G = X.view( size[0], g, -1)    # group division
    m, g, c = X_G.size()
    # centering
    mean = X_G.mean( -1, keepdim = True)
    X_G _mean = X_G – mean
    # approximate ZCA whitening by Newton's iteration
    P = [ torch.Tensor([]) for _ in range(T+1) ]
    sigma = x_mean.matmul( X_G _mean.transpose(1, 2)) / c
    P[0] = torch.eye(d).to(x).expand(sigma.shape)
    M_zero = sigma.clone().fill_(0)
    trace_inv = torch.addcmul(M_zero, sigma, P[0] ).sum( (1, 2), keepdim= True).reciprocal_()
    sigma_N=torch.addcmul( M_zero, sigma, trace_inv )
    for k in range(T):
        P[k+1] = torch.baddbmm( 1.5, P[k], -0.5, torch.matrix_power(P[k], 3), sigma_N)
    wm = torch.addcmul( M_zero, P[T], trace_inv.sqrt())
    y = wm.matmul( X_G _mean )
    output = y.view_as(X) # inverse group division
    return output * gamma + beta
```

Figure 1. Python code of GW using ItN whitening, based on PyTorch.

$\psi^b_{ItN}(\frac{\partial\mathcal{L}}{\partial\Sigma^{-\frac{1}{2}}})$ is as follows:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\mathbf{P}_T} &= \frac{1}{\sqrt{tr(\Sigma)}}\frac{\partial\mathcal{L}}{\partial\Sigma^{-\frac{1}{2}}}\\
\frac{\partial\mathcal{L}}{\partial\Sigma_N} &= -\frac{1}{2}\sum_{k=1}^{T}(\mathbf{P}^3_{k-1})^T\frac{\partial\mathcal{L}}{\partial\mathbf{P}_k}\\
\frac{\partial\mathcal{L}}{\partial\Sigma} &= \frac{1}{tr(\Sigma)}\frac{\partial\mathcal{L}}{\partial\Sigma_N} - \frac{1}{(tr(\Sigma))^2}tr(\frac{\partial\mathcal{L}}{\partial\Sigma_N}^T\Sigma)\mathbf{I}\\
&\quad - \frac{1}{2(tr(\Sigma))^{3/2}}tr((\frac{\partial\mathcal{L}}{\partial\Sigma^{-\frac{1}{2}}})^T\mathbf{P}_T)\mathbf{I}. \qquad (5)
\end{aligned}$$

Here, $\frac{\partial\mathcal{L}}{\partial\mathbf{P}_k}$ can be calculated by the following iterations:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\mathbf{P}_{k-1}} &= \frac{3}{2}\frac{\partial L}{\partial\mathbf{P}_k} - \frac{1}{2}\frac{\partial\mathcal{L}}{\partial\mathbf{P}_k}(\mathbf{P}^2_{k-1}\Sigma_N)^T - \frac{1}{2}(\mathbf{P}^2_{k-1})^T\frac{\partial\mathcal{L}}{\partial\mathbf{P}_k}\Sigma_N^T\\
&\quad - \frac{1}{2}(\mathbf{P}_{k-1})^T\frac{\partial\mathcal{L}}{\partial\mathbf{P}_k}(\mathbf{P}_{k-1}\Sigma_N)^T, \quad k = T, ..., 1. \qquad (6)
\end{aligned}$$

We also provide the python code of GW using ItN whitening, based on PyTorch [9], in Figure 1.

## 2. More Results on Effects of Batch Size and Group Number

In Figure 1 of the submitted paper, we show the effects of batch size (group number) for batch (group) normalized networks, where the results are obtained with a learning rate of 0.1. Here, we provide more results using different learning rates, shown in Figure 2. We obtain similar observations.

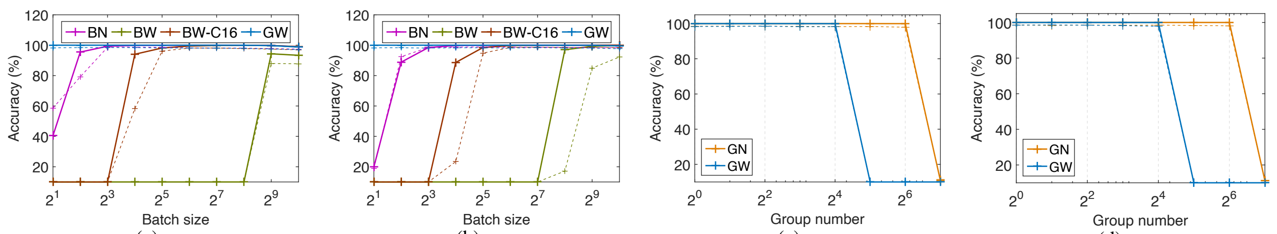
## 3. More Results on Conditioning Analysis

In Figure 2 of the submitted paper, we perform a conditioning analysis on the normalized output, where we report $\kappa_{90\%}$ and use a one-layer and two-layer multilayer perceptron (MLP) as $f(\cdot)$ to obtain the activations. Here, we provide more results, shown in Figure 3. We obtain similar observations.

## 4. Derivation of Constraint Number of Normalization Methods

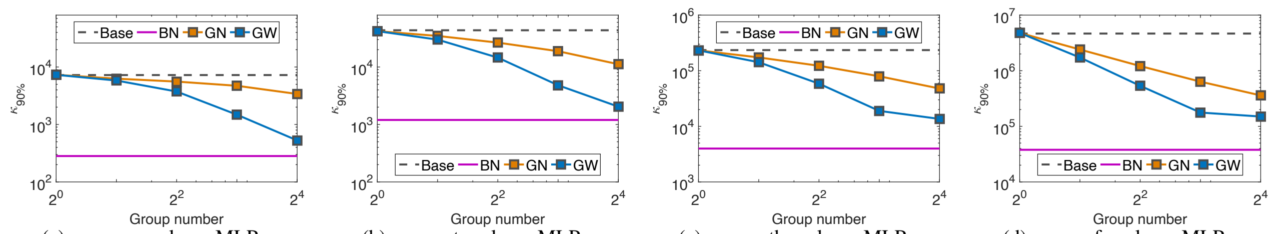
In Section 4 of the submitted paper, we define the constraint number of a normalization operation, and summarize the constraint number of different normalization methods in Table 1 of the submitted paper. Here, we provide the details for deriving the constraint number of batch whitening (BW), group normalization (GN) [11] and our proposed GW, for the mini-batch input $\mathbf{X} \in \mathbb{R}^{d\times m}$.

**Constraint number of BW.** BW [3] ensures that the normalized output is centered and whitened, which has the

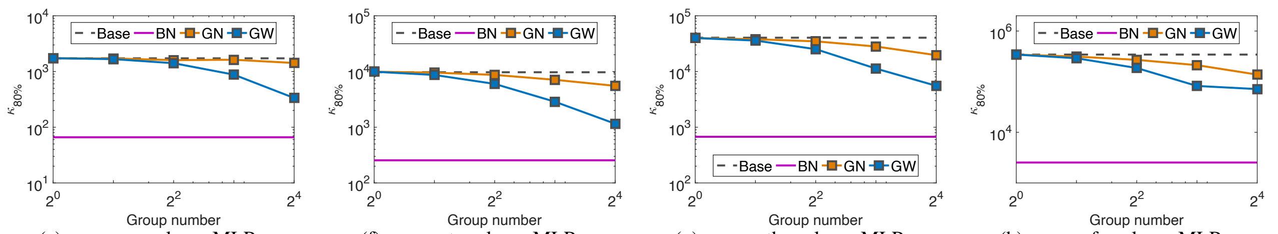


(a) (b) (c) (d)

Figure 2. Effects of batch size (group number) for batch (group) normalized networks. The experimental setup is the same as the one in Figure 1 of the submitted paper. (a) Effect of batch size using a learning rate of 0.01; (b) Effect of batch size using a learning rate of 0.5; (c) Effect of group number using a learning rate of 0.01; (d) Effect of group number using a learning rate of 0.05;



(a) $\kappa_{90\%}$, one-layer MLP (b) $\kappa_{90\%}$, two-layer MLP (c) $\kappa_{90\%}$, three-layer MLP (d) $\kappa_{90\%}$, four-layer MLP



(e) $\kappa_{80\%}$, one-layer MLP (f) $\kappa_{80\%}$, two-layer MLP (g) $\kappa_{80\%}$, three-layer MLP (h) $\kappa_{80\%}$, four-layer MLP

Figure 3. Conditioning analysis on the normalized output. We simulate the activations $\mathbf{X} = f(\mathbf{X}_0) \in \mathbb{R}^{256 \times 1024}$ using a network $f(\cdot)$, where $\mathbf{X}_0$ is sampled from a Gaussian distribution. We evaluate the more general condition number with respect to the percentage: $\kappa_p = \frac{\lambda_{max}}{\lambda_p}$, where $\lambda_p$ is the $pd$-th eigenvalue (in descending order) and $d$ is the total number of eigenvalues. We show the $\kappa_{90\%}$ of the covariance matrix of the output in (a)(b)(c)(d), and the $\kappa_{80\%}$ of the covariance matrix of the output in (e)(f)(g)(h). We use a one-layer MLP as $f(\cdot)$ in (a)/(e); a two-layer MLP in (b)/(f); a three-layer MLP in (c)/(g); and a four-layer MLP in (d)/(h).

constraints $\Upsilon_{\phi_{BW}}(\widehat{\mathbf{X}})$ as:

$$\widehat{\mathbf{X}}\mathbf{1} = \mathbf{0}_d, \quad and \tag{7}$$

$$\widehat{\mathbf{X}}\widehat{\mathbf{X}}^T - m\mathbf{I} = \mathbf{0}_{d \times d}, \tag{8}$$

where $\mathbf{0}_d$ is a $d$-dimensional column vector of all zeros, and $\mathbf{0}_{d \times d}$ is a $d \times d$ matrix of all zeros. Note that there are $d$ independent equations in the system of equations $\widehat{\mathbf{X}}\mathbf{1} = \mathbf{0}_d$. Let's denote $\mathbf{M} = \widehat{\mathbf{X}}\widehat{\mathbf{X}}^T - m\mathbf{I}$. We have $\mathbf{M}^T = \mathbf{M}$, and thus $\mathbf{M}$ is a symmetric matrix. Therefore, there are $d(d+1)/2$ independent equations in the system of equations $\widehat{\mathbf{X}}\widehat{\mathbf{X}}^T - m\mathbf{I} = \mathbf{0}_{d \times d}$. We thus have $d(d+1)/2 + d$ independent equations in $\Upsilon_{\phi_{BW}}(\widehat{\mathbf{X}})$, and the constraint number of BW is $d(d+3)/2$.

**Constraint number of GN.** Given a sample $\mathbf{x} \in \mathbb{R}^d$, GN divides the neurons into groups: $\mathbf{Z} = \Pi(\mathbf{x}) \in \mathbb{R}^{g \times c}$, where $g$ is the group number and $d = gc$. The standardization operation is then performed on $\mathbf{Z}$ as:

$$\widehat{\mathbf{Z}} = \Lambda_g^{-\frac{1}{2}}(\mathbf{Z} - \mu_g \mathbf{1}^T), \tag{9}$$

where, $\mu_g = \frac{1}{c}\mathbf{Z}\mathbf{1}$ and $\Lambda_g = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_g^2) + \epsilon\mathbf{I}$ . This ensures that the normalized output $\widehat{\mathbf{Z}}$ for each sample has the constraints:

$$\sum_{j=1}^{c} \widehat{\mathbf{Z}}_{ij} = 0 \; and \; \sum_{j=1}^{c} \widehat{\mathbf{Z}}_{ij}^2 = c, \;\; for \; i = 1, ..., g. \tag{10}$$

In the system of equations 10, the number of independent equations is $2g$. Therefore, the constraint number of GN is $2dm$, when given $m$ samples.

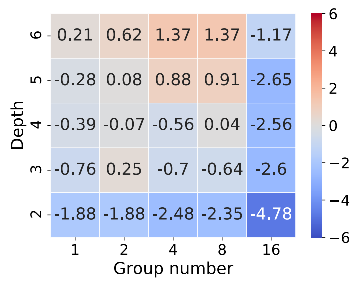


Figure 4. Effects of group number for GN on CNNs for CIFAR-10 classification. We vary the group number of GN, and evaluate the difference in training accuracy between GN and the model without normalization ('Base').

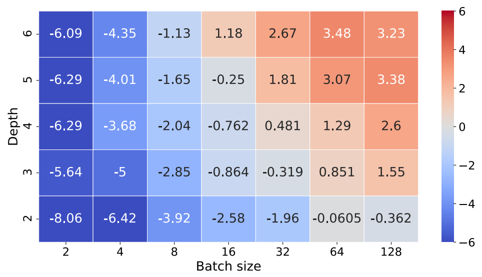


Figure 5. Effects of batch size for BN on CNNs for CIFAR-10 classification. We vary the batch size of BN, and evaluate the difference in training accuracy between BN and 'Base'.

**Constraint number of GW.** Given a sample $\mathbf{x} \in \mathbb{R}^d$, GW performs normalization as:

$$Group\ division: \mathbf{X}_G = \Pi(\mathbf{x}; g) \in \mathbb{R}^{g \times c}, \tag{11}$$

$$Whitening: \widehat{\mathbf{X}}_G = \Sigma_g^{-\frac{1}{2}}(\mathbf{X}_G - \mu_g \mathbf{1}^T), \tag{12}$$

$$Inverse\ group\ division: \hat{\mathbf{x}} = \Pi^{-1}(\widehat{\mathbf{X}}_G) \in \mathbb{R}^d. \tag{13}$$

The normalization operation ensures that $\widehat{\mathbf{X}}_G \in \mathbb{R}^{g \times c}$ has the following constraints:

$$\widehat{\mathbf{X}}_G \mathbf{1} = \mathbf{0}, \quad and \tag{14}$$

$$\widehat{\mathbf{X}}_G \widehat{\mathbf{X}}_G^T - c\mathbf{I} = \mathbf{0}. \tag{15}$$

Following the analysis for BW, the number of independent equations is $g(g+3)/2$ from Eqns. 14 and 15. Therefore, the constraint number of GW is $mg(g+3)/2$, when given $m$ samples.

## 5. Investigating Representational Capacity on CNNs

In Section 4.2 of the submitted paper, we empirically show how normalization affects the representational capacity of a network with experiments conducted on MLPs. Here, we conduct experiments on convolutional neural networks (CNNs) for CIFAR-10 classification. Note that the number of neurons to be normalized for GN is $dHW$, given the convolutional input $\mathbf{X} \in \mathbb{R}^{d \times m \times H \times W}$, where H and W are the height and width of the feature maps. The number of samples to be normalized for BN is $mHW$. We use a CNN with $n$ convolutional layers, following average pooling and one fully connected layer. We use $d = 16$ channels in each layer and vary the depth $n$. We apply stochastic gradient descent (SGD) with a momentum of 0.9. We train over 160 epochs and divide the learning rate by 5 at 60 and 120 epochs. We evaluate the best training accuracy among the learning rates of $\{0.001, 0.01, 0.05, 0.1, 0.5\}$.

Figure 4 shows the results of GN with varying group number (we use a batch size of 256), where we report the difference in training accuracy between GN and the model without normalization ('Base'). We observe that: 1) GN has significantly degenerated performance when the group number is too large (relative to the channel number), *e.g.*, GN has worse performance than 'Base' when $g = d = 16$; 2) The net gain of GN over 'Base' is amplified as the depth increases. These observations are consistent with the experiments on the MLPs shown in Section 4.2 of the submitted paper.

Figure 5 gives the results of BN with varying batch size, wher we report the difference in accuracy between BN and 'Base'. We observe that: 1) BN has significantly degenerated performance when the batch size is too small, *e.g.*, BN has worse performance than 'Base' when $m = 2$ or $m = 4$; 2) The net gain of BN over 'Base' is amplified as the depth increases. These observations suggest that there is also a trade-off for BN between the benefits of normalization on optimization and its constraints on representation.

## 6. More Experimental Results on ImageNet

### 6.1. Learning Decorrelated Feature Representations

As described in Section 5.1.1 of the submitted paper, we investigate the effect of inserting a GW/BW layer after the last average pooling (before the last linear layer) to learn the decorrelated feature representations, as proposed in [5]. We provide the results in Table 1. This can slightly improve the performance (0.10% on average) when using GW (comparing Table 1 to Table 2 of the submitted paper). We note that $BW_\Sigma$ benefits the most from this kind of architecture.

### 6.2. Running Time Comparison

In this section, we compare the wall-clock time of the models described in Section 5.1 of the submitted paper. We run the experiments on GPUs (NVIDIA Tesla V100). All implementations are based on the API provided by PyTorch, with CUDA (version number: 9.0). We use the same experimental setup as described in Section 5.1 of the submitted paper. We evaluate the training time for each iteration, averaged over 100 iterations. The ResNets-50 baseline (BN)

| | S1 | S1-B1 | S1-B2 | S1-B3 | S1-B12 |
|---|---|---|---|---|---|
| Baseline (BN) | 76.24 | 76.24 | 76.24 | 76.24 | 76.24 |
| BW [5] | 76.91 (↑0.67) | 76.94 (↑0.70) | 76.93 (↑0.69) | 76.78 (↑0.54) | 76.79 (↑0.55) |
| $\mathrm{BW}_\Sigma$ [4] | **77.09** (↑0.85) | 77.04 (↑0.80) | 77.21 (↑0.97) | 77.10 (↑0.86) | 77.11 (↑0.87) |
| GW | 76.86 (↑0.62) | **77.63** (↑1.39) | **77.80** (↑1.56) | **77.75** (↑1.51) | **77.48** (↑1.24) |

Table 1. Effects of inserting a GW/BW/$\mathrm{BW}_\Sigma$ layer after the last average pooling of ResNet-50 to learn decorrelated feature representations for ImageNet classification. We evaluate the top-1 validation accuracy on five architectures (**S1**, **S1-B1**, **S1-B2**, **S1-B3** and **S1-B12**), described in the submitted paper. Note that we also use an extra BN layer after the last average pooling for the Baseline (BN).

| | S1 | S1-B1 | S1-B2 | S1-B3 | S1-B12 |
|---|---|---|---|---|---|
| Baseline (BN) | 419 | 419 | 419 | 419 | 419 |
| GW | 437 (Δ4.3%) | 518 (Δ23.6%) | 514 (Δ22.7%) | 634 (Δ51.3%) | 589 (Δ40.6%) |

Table 2. Time costs ($ms$) of five architectures when applying GW on ResNet-50 (**S1**, **S1-B1**, **S1-B2**, **S1-B3** and **S1-B12**). Note that $\Delta\ x\%$ indicates the additional time cost is $x\%$, compared to the baseline.

costs 419 $ms$. Replacing the BNs of ResNet-50 with our GWs ($g$=64) costs 796 $ms$, a 90% additional time cost on ResNet-50. This is one factor that drives us to investigate the position at which to apply GW.

Table 2 shows the time costs of five architectures, **S1**, **S1-B1**, **S1-B2**, **S1-B3** and **S1-B12**, which have 1, 17, 17, 17 and 33 GW modules, respectively. Note that applying GW in the **S1-B3** architecture results in a clearly increased computational cost, compared to **S1-B1/S1-B2**. This is because the channel number of the third normalization layer is 4× larger than that of the first/second normalization layer, in the bottleneck blocks [1].

Table 3 shows the time costs of ResNets [1] and ResNeXts [12] (the corresponding models in Table 3 of the submitted paper) for ImageNet classification.

## 6.3. Results on Advanced Training Strategies

In Section 5.1 of the submitted paper, we show the effectiveness of our GW on ResNets [1] and ResNeXts [12], under the standard training strategy (*e.g.*, using learning rate step decay). Here, we also conduct experiments using more advanced training strategies: 1) We train over 100 epochs with cosine learning rate decay [7]; 2) We add the label smoothing tricks with a smoothing constant $\varepsilon = 0.1$ [2]; 3) We use mixup training with $\alpha = 0.2$ in the Beta distribution [13]. The results are shown in Table 4, where GW improves the baselines consistently.

| Method | ResNet-50 | ResNeXt-50 |
|---|---|---|
| Baseline (BN) [6] | 77.16 | 78.84 |
| GN [11] | 76.09 (↓1.07) | 76.90 (↓1.94) |
| $\mathrm{BW}_\Sigma$ [4] | 78.29 (↑1.13) | 79.55 (↑0.71) |
| GW | **78.46** (↑1.30) | **80.07** (↑1.23) |

Table 4. Comparison of validation accuracy on 50-layer ResNet [1] and ResNeXt [12] for ImageNet on more advanced training strategies (*e.g.*, cosine learning rate decay [7], label smoothing [2] and mixup [13]). Note that we use an additional layer for $\mathrm{BW}_\Sigma$ to learn the decorrelated feature, as recommended in [4].

## 7. Additional Experiments on Neural Machine Translation

Our GW does indeed generalize layer normalization (LN), which is a widely used technique in NLP tasks. We thus believe our GW has the potential to improve the performance of LN in NLP tasks. We conduct additional experiments to apply our GW on Transformer [10] (where LN is the default normalization) for machine translation tasks using *fairseq-py* [8]. We evaluate on the public IWSLT14 German-to-English (De-EN) dataset using BLEU (higher is better). We use the hyper-parameters recommended in *fairseq-py* [8] for Transformer and train over 50 epochs with five random seeds. The baseline LN has a BLEU score of $35.02 \pm 0.09$. GW (replacing all the LNs with GWs) has a BLEU score of $35.27 \pm 0.06$. Note that the hyperparameters were designed for LN, and may not be optimal for GW.

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5, 6

[2] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019. 5

[3] Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. In *CVPR*, 2018. 1, 2

[4] Lei Huang, Lei Zhao, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. An investigation into the stochasticity of batch whitening. In *CVPR*, 2020. 5

[5] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization to-

| Method | ResNet-50 | ResNet-101 | ResNeXt-50 | ResNeXt-101 |
|---|---|---|---|---|
| Baseline (BN) [6] | 419 | 672 | 574 | 912 |
| $\text{BW}_\Sigma$ [5] | 550 $(\Delta 31.3\%)$ | 882 $(\Delta 30.9\%)$ | 798 $(\Delta 39.0\%)$ | 1587 $(\Delta 74.0\%)$ |
| GW | 514 $(\Delta 22.7\%)$ | 810 $(\Delta 20.5\%)$ | 738 $(\Delta 28.6\%)$ | 1180 $(\Delta 29.3\%)$ |

Table 3. Time costs $(ms)$ of ResNets [1] and ResNeXts [12] for ImageNet classification. Note that $\Delta\ x\%$ indicates the additional time cost is $x\%$, compared to the baselines.

wards efficient whitening. In *CVPR*, 2019. 1, 4, 5, 6

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 1, 5, 6

[7] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. In *ICLR*, 2017. 5

[8] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019. 5

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017. 2

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 5

[11] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 2, 5

[12] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 5, 6

[13] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 5