Group Whitening: Balancing Learning Efficiency and Representational Capacity

Lei Huang Li Liu Fan Zhu Ling Shao
Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE
{lei.huang, li.liu, fan.zhu, ling.shao} @inceptioniai.org

Abstract

Batch normalization (BN) is an important technique commonly incorporated into deep learning models to perform standardization within mini-batches. The merits of BN in improving model's learning efficiency can be further amplified by applying whitening, while its drawbacks in estimating population statistics for inference can be avoided through group normalization (GN). This paper proposes group whitening (GW), which elaborately exploits the advantages of the whitening operation and avoids the disadvantages of normalization within mini-batches. Specifically, GW divides the neurons of a sample into groups for standardization, like GN, and then further decorrelates the groups. In addition, we quantitatively analyze the constraint imposed by normalization, and show how the batch size (group number) affects the performance of batch (group) normalized networks, from the perspective of model's representational capacity. This analysis provides theoretical guidance for applying GW in practice. Finally, we apply the proposed GW to ResNet and ResNeXt architectures and conduct experiments on the ImageNet and COCO benchmarks. Results show that GW consistently improves the performance of different architectures, with absolute gains of $1.02\% \sim 1.49\%$ in top-1 accuracy on ImageNet and $1.82\% \sim 3.21\%$ in bounding box AP on COCO.

1 Introduction

Batch normalization (BN) [20] represents a milestone technique in deep learning [13, 45, 51], and has been extensively used in various network architectures [13, 45, 56, 44, 15]. BN standardizes the activations within a mini-batch of data, which improves the conditioning of optimization and accelerates training [20, 3, 40]. The stochasticity of normalization introduced along the batch dimension is believed to benefit generalization [51, 41, 18]. However, this stochasticity also results in differences between the training distribution (using mini-batch statistics) and the test distribution (using estimated population statistics) [19], which is believed to be the main cause of BN's small-batch-size problem — BN's error increases rapidly as the batch size becomes smaller [51]. To address this issue, a number of approaches have been proposed [51, 37, 32, 19, 48, 43]. One representative method is group normalization (GN), which divides the neurons into groups and then performs the standardization operation over the neurons of each group, for each sample, independently. GN provides a flexible solution to avoid normalization along the batch dimension, and benefits on visual tasks limited to small-batch-size training [51].

As a widely used operation in data pre-processing, whitening not only standardizes but also decorrelates the data [26], which further improves the conditioning of optimization problem [26, 50, 16]. A whitened input has also been shown to make the gradient descent updates similar to the Newton updates for linear models [26, 50, 16]. Motivated by this, Huang *et al.* [16] proposed batch whitening (BW) for deep models, which performs whitening on the activations of each layer within a mini-batch. BW has been shown to achieve better optimization efficiency and generalization than BN [16, 18, 35]. However, BW further amplifies the disadvantage of BN in estimating the population statistics, where

Preprint. Under review.

the number of parameters to be estimated with BW is quadratic to the number of neurons/channels. Thus, BW requires a sufficiently large batch-size to work well.

To exploit whitening's advantage in optimization, while avoiding its disadvantage in normalization along the batch dimension, this paper proposes group whitening (GW). For each sample, GW divides the neurons into groups for standardization over the neurons of each group (like GN), and then further decorrelates the groups. Unlike BW, GW has stable performance for a wide range of batch sizes, like GN, and thus can be applied to a variety of tasks. GW further improves the conditioning of optimization of GN with its whitening operation.

One important hyperparameter of GW is the group number. We observe that GW/GN has a significantly degenerated training performance when the group number is large, which is similar to the small-batch-size problem of BW/BN. We attribute this to the constraints on the output imposed by the normalization operation, which affect the model's representational capacity. As such, this paper defines the **constraint number** of normalization (as will be discussed in Section C) to quantitatively measure the magnitude of the constraints provided by normalization methods. With the support of the constraint number, we analyze how the batch size (group number) affects the model's representational capacity for batch (group) normalized networks. Our analysis presents a new viewpoint for understanding the small-batch-size problem of BN.

We apply the proposed GW to two representative deep network architectures (ResNet [13] and ResNeXt [52]) for ImageNet classification [39] and COCO object detection and instance segmentation [31]. GW consistently improves the performance for both architectures, with absolute gains of 1.02% ~1.49% in top-1 accuracy for ImageNet and 1.82% ~3.21% in bounding box AP for COCO.

2 Preliminaries

To simplify the discussion, we first consider the d-dimensional input vector $\mathbf{x}$, which will be generalized to a convolutional input in the subsequent section. Let $\mathbf{X} \in \mathbb{R}^{d\times m}$ be a data matrix denoting the mini-batch input of size m in a given layer.

Standardization. During training, batch normalization (BN) [20] standardizes the layer input within a mini-batch, for each neuron, as[1]:

$$\widehat{\mathbf{X}} = \phi_{BN}(\mathbf{X}) = \Lambda_d^{-\frac{1}{2}}(\mathbf{X} - \mu_d \mathbf{1}^T). \tag{1}$$

Here, $\mu_d = \frac{1}{m}\mathbf{X}\mathbf{1}$ and $\Lambda_d = \text{diag}(\sigma_1^2, \dots, \sigma_d^2) + \epsilon\mathbf{I}$, where σ_i^2 is the variance over mini-batches for the i-th neuron, $\mathbf{1}$ is a column vector of all ones, and $\epsilon > 0$ is a small number to prevent numerical instability. During inference, the population statistics $\{\widehat{\Lambda}_d^{-\frac{1}{2}}, \hat{\mu}_d\}$ are required for deterministic inference, and they are usually calculated by running average over the training iterations, as follows:

$$\begin{cases} \hat{\mu}_d = (1-\lambda)\hat{\mu}_d + \lambda\mu_d, \\ \widehat{\Lambda}_d^{-\frac{1}{2}} = (1-\lambda)\widehat{\Lambda}_d^{-\frac{1}{2}} + \lambda\Lambda_d^{-\frac{1}{2}}. \end{cases} \tag{2}$$

Such an estimation process can limit BN's usage in recurrent neural networks [25, 9], or harm the performance for small-batch-size training [19, 51].

To avoid the estimation of population statistics shown in Eqn. 2, Ba *et al.* proposed layer normalization (LN) [3] to standardize the layer input within the neurons for each training sample, as:

$$\widehat{\mathbf{X}} = \phi_{LN}(\mathbf{X}) = (\mathbf{X} - \mathbf{1}\mu_m^T)\Lambda_m^{-\frac{1}{2}}. \tag{3}$$

Here, $\mu_m = \frac{1}{d}\mathbf{X}^T\mathbf{1}$ and $\Lambda_m = \text{diag}(\sigma_1^2, \dots, \sigma_m^2) + \epsilon\mathbf{I}$, where σ_i^2 is the variance over the neurons for the i-th sample. LN has the same formulation during training and inference, and is extensively used in natural language processing tasks [47, 55, 53], since it does not normalize within a mini-batch.

Group normalization (GN) [51] further generalizes LN, dividing the neurons into groups and performing the standardization within the neurons of each group independently, for each sample. Specifically, defining the group division operation as $\Pi : \mathbb{R}^{d\times m} \mapsto \mathbb{R}^{c\times gm}$, where g is the group number and $d = gc$, GN can be represented as follows:

$$\widehat{\mathbf{X}} = \phi_{GN}(\mathbf{X}; g) = \Pi^{-1}(\phi_{LN}(\Pi(\mathbf{X}))), \tag{4}$$

[1]BN and other normalization methods discussed in this paper all use extra learnable dimension-wise scale and shift parameters [20].

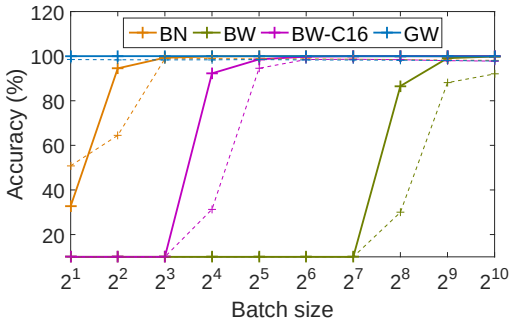


Figure 1: Effects of batch size for different normalization methods. We train a four-layer multilayer perceptron (MLP) with 256 neurons in each layer, for MNIST classification. We compare BN, BW, group-based BW with 16 neurons/channels in each group ('BW-C16'), and GW (we use a group number of 16). We vary the batch size and evaluate the training (thick 'plus' with solid line) and validation (thin 'plus' with dashed line) accuracies at the end of 50 training epochs. These results are obtained using a learning rate of 0.1, but we also obtain similar observations for other learning rates (see *Appendix* B for details).

where $\Pi^{-1} : \mathbb{R}^{c\times gm} \mapsto \mathbb{R}^{d\times m}$ is the inverse operation of Π. It is clear from Eqn. 21 that LN is a special case of GN with $g = 1$. By changing the group number g, GN is more flexible than LN, enabling it to achieve good performance on visual tasks limited to small-batch-size training (*e.g.*, object detection and segmentation [51]).

Whitening. To exploit the advantage of whitening over standardization in improving the conditioning of optimization, Huang *et al.* proposed decorrelated BN [16], which performs zero-phase component analysis (ZCA) whitening to normalize the layer input within a mini-batch, as:

$$\phi^W_{ZCA}(\mathbf{X}) = \Sigma_d^{-\frac{1}{2}}(\mathbf{X} - \mu_d\mathbf{1}^T) = \mathbf{D}\Lambda^{-\frac{1}{2}}\mathbf{D}^T(\mathbf{X} - \mu_d\mathbf{1}^T), \tag{5}$$

where $\Lambda = \text{diag}(\tilde{\sigma}_1, \dots, \tilde{\sigma}_d)$ and $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvalues and associated eigenvectors of Σ, *i.e.* $\Sigma = \mathbf{D}\Lambda\mathbf{D}^T$, and $\Sigma = \frac{1}{m}(\mathbf{X}-\mu_d\mathbf{1}^T)(\mathbf{X}-\mu_d\mathbf{1}^T)^T+\epsilon\mathbf{I}$ is the covariance matrix of the centered input. One crucial problem in Eqn. 5 is the eigen-decomposition, which is computationally expensive on a GPU and numerically instable. To address this issue, iterative normalization ('ItN') [18] was proposed to approximate the ZCA whitening matrix $\Sigma_d^{-\frac{1}{2}}$ using Newton's iteration [4]:

$$\phi^W_{ItN}(\mathbf{X}) = \Sigma_d^{-\frac{1}{2}}(\mathbf{X} - \mu_d\mathbf{1}^T) = \frac{\mathbf{P}_T}{\sqrt{tr(\Sigma_d)}}(\mathbf{X} - \mu_d\mathbf{1}^T), \tag{6}$$

where $tr(\Sigma_d)$ indicates the trace of Σ_d and $\mathbf{P}_T$ is calculated iteratively as:

$$\begin{cases}\mathbf{P}_0 = \mathbf{I} \\ \mathbf{P}_k = \frac{1}{2}(3\mathbf{P}_{k-1} - \mathbf{P}^3_{k-1}\Sigma^N_d), \quad k = 1, 2, ..., T.\end{cases} \tag{7}$$

Here, $\Sigma^N_d = \Sigma_d/tr(\Sigma_d)$. Other BW methods also exist for calculating the whitening matrix [16, 42]; please refer to [23, 17] for more details.

It is necessary for BW to estimate the population statistics of the whitening matrix $\widehat{\Sigma}_d^{-\frac{1}{2}}$ during inference, like BN. However, the number of independent parameters in $\widehat{\Sigma}_d^{-\frac{1}{2}}$ of BW is $d(d+1)/2$, while $\widehat{\Lambda}_d^{-\frac{1}{2}}$ of BN is d. This amplifies the difficulty in estimation and requires a sufficiently large batch size for BW to work well (Figure 1). Although group-based BW [16] — where neurons are divided into groups and BW is performed within each one — can relieve this issue, it is still sensitive to the batch size (Figure 1) due to its inherent drawback of normalizing along the batch dimension.

3 Group Whitening

We propose group whitening (GW). Given a sample $\mathbf{x} \in \mathbb{R}^d$, GW performs normalization as:

$$Group\ division: \quad \mathbf{X}_G = \Pi(\mathbf{x}; g) \in \mathbb{R}^{g\times c}, \tag{8}$$

$$Whitening: \quad \widehat{\mathbf{X}}_G = \phi^W(\mathbf{X}_G) = \Sigma_g^{-\frac{1}{2}}(\mathbf{X}_G - \mu_g\mathbf{1}^T), \tag{9}$$

$$Inverse\ group\ division: \quad \hat{\mathbf{x}} = \Pi^{-1}(\widehat{\mathbf{X}}_G) \in \mathbb{R}^d, \tag{10}$$

where $\Pi : \mathbb{R}^d \mapsto \mathbb{R}^{g\times c}$ and its inverse transform $\Pi^{-1} : \mathbb{R}^{g\times c} \mapsto \mathbb{R}^d$. We can use different whitening operations [16, 42, 23] in Eqn. 24. Here, we use ZCA whitening (Eqn. 5) and its efficient

approximation 'ItN' (Eqn 6), since they work well on discriminative tasks [16, 18, 35]. We provide the full algorithms (forward and backward passes) and PyTorch [36] implementations in the *Appendix A*.

GW ensures the normalized activation for each sample has the properties: $\widehat{\mathbf{X}}_G\mathbf{1} = \mathbf{0}$ and $\frac{1}{c}\widehat{\mathbf{X}}_G\widehat{\mathbf{X}}_G^T = \mathbf{I}$, which should improve the conditioning, like BW, and benefit training. GW avoids normalization along the batch dimension, and it works stably across a wide range of batch sizes (Figure 1).

Convolutional layer. For 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, BN and BW consider each spatial position in a feature map as a sample [20] and normalize over the unrolled input $\mathbf{X} \in \mathbb{R}^{d\times mHW}$. In contrast, LN and GN view each spatial position in a feature map as a neuron [51] and normalize over the unrolled input $\mathbf{X} \in \mathbb{R}^{dHW\times m}$. Following GN, GW also views each spatial position as a neuron, *i.e.*, GW operations (Eqns. 23, 24 and 25) are performed for each sample with unrolled input $\mathbf{x} \in \mathbb{R}^{dHW}$.

Computational complexity. For a convolutional mini-batch input $\mathbf{X} \in \mathbb{R}^{d\times m\times H\times W}$, GW using ZCA whitening (Eqn. 5) costs $2mHWdg + mO(g^3)$. Using the more efficient 'ItN' operation (Eqn. 6), GW costs $2mHWdg + mTg^3$, where T is the iteration number, while the 3×3 convolution with the same input and output feature maps costs $9mHWd^2$. The relative cost of GW for a 3×3 convolution is $2g/(9d) + Tg^3/(9HWd^2)$.

Difference from group-based BW. Our method is significantly different from the group-based BW [16], in which the whitening operation is also performed within mini-batch data. Specifically, group-based BW has difficulty in estimating the population statistics, as discussed in Section 2. Note that group-based BW is reduced to BN if the channel number in each group $c = 1$, while GW is reduced to GN if the group number $g = 1$.

4 Constraint Analysis for Normalization

The normalization operation ensures that the normalized output $\widehat{\mathbf{X}} = \phi(\mathbf{X}) \in \mathbb{R}^{d\times m}$ has a stable distribution. This stable distribution can be implicitly viewed as the constraints imposed on $\widehat{\mathbf{X}}$, which can be represented as a system of equations $\Upsilon_\phi(\widehat{\mathbf{X}})$. For example, BN provides the constraints $\Upsilon_{\phi_{BN}}(\widehat{\mathbf{X}})$ as:

$$\sum_{j=1}^{m}\widehat{\mathbf{X}}_{ij} = 0 \; and \; \sum_{j=1}^{m}\widehat{\mathbf{X}}_{ij}^2 - m = 0, \; for \; each \; neuron \; i = 1, ..., d. \tag{11}$$

Here, we define the **constraint number** of normalization to quantitatively measure the magnitude of the constraints provided by the normalization method.

Definition 1 *Given the input data $\mathbf{X} \in \mathbb{R}^{d\times m}$, the **constraint number** of a normalization operation $\phi(\cdot)$, referred to as $\zeta(\phi; \mathbf{X})$, is the number of independent equations in $\Upsilon_\phi(\widehat{\mathbf{X}})$.*

As an example, we have $\zeta(\phi_{BN}; \mathbf{X}) = 2d$ from Eqn. 11. Table 1 summarizes the constraint numbers of the main normalization methods discussed in this paper (please refer to the *Appendix C* for derivation details). We can see that the whitening operation provides significantly stronger constraints than the standardization operation.

The normalization operation can also be regarded as a way to find a solution $\widehat{\mathbf{X}}$ satisfying the constraints $\Upsilon_\phi(\widehat{\mathbf{X}})$. To ensure the solution is feasible, it must satisfy the following condition:

$$\zeta(\phi; \mathbf{X}) \leq \chi(\widehat{\mathbf{X}}), \tag{12}$$

where $\chi(\widehat{\mathbf{X}}) = md$ is the number of variables in $\widehat{\mathbf{X}}$. Based on Eqn. 12, we have $m >= 2$ for BN to ensure a feasible solution. We also provide the ranges of batch size/group number for other normalization methods in Table 1. Note that the batch size m should be larger than/equal to d to achieve a numerically stable solution for BW when using ZCA whitening in practice [16]. This also applies to GW, where g should be less than/equal to $\sqrt{d}$.

4.1 Analysis on Representational Capacity

It is believed that the constraints introduced by normalization affect the representational capacity of neural networks [20], while the batch size of the optimization algorithm significantly affects the performance of batch normalized networks [51, 18]. Here, we provide a unified analysis based on the

Table 1: Summary of $\zeta(\phi;\mathbf{X})$, $\zeta(\phi;\mathbb{D})$ and ranges of m/g for normalization methods.

	Normalization along a batch		Normalization along a group of neurons	
	BN	BW	GN	GW
$\zeta(\phi;\mathbf{X})$	$2d$	$\frac{d(d+3)}{2}$	$2gm$	$\frac{mg(g+3)}{2}$
$\zeta(\phi;\mathbb{D})$	$\frac{2Nd}{m}$	$\frac{Nd(d+3)}{2m}$	$2gN$	$\frac{Ng(g+3)}{2}$
Ranges of m/g	$m \geq 2$	$m \geq \frac{d+3}{2}$	$g \leq \frac{d}{2}$	$g \leq \frac{\sqrt{8d+9}-3}{2}$

constraint number of normalization, and show how the batch size m of the optimization algorithm affects the representational capacity of the model using BN/BW, but not GN/GW. Our analysis is based on the following assumption.

Assumption 1 *The constraint number of the normalization and the representational capacity of the normalized model have a negative correlation*[2].

Batch normalized networks. Given training data $\mathbb{D}$ of size N, we consider the optimization algorithm with batch size m (we assume N is divisible by m). We calculate the constraint number of normalization over the entire training data $\zeta(\phi;\mathbb{D})$. The results for different normalization methods are shown in Table 1. We find that $\zeta(\phi_{BN};\mathbb{D})$ or $\zeta(\phi_{BW};\mathbb{D})$ is inversely proportional to m, which suggests that the representational capacity of batch normalized networks decreases with decreasing batch size, based on Assumption 1. For example, the normalized outputs of BN are constrained to be $(1,-1)$ or $(-1,1)$ [7] when $m=2$, which heavily reduces the representational capacity of the model and results in significantly degenerated training performance, as shown in Figure 1. To the best of our knowledge, our analysis is the first to show how the batch size of an optimization algorithm affects the model's representational capacity for batch normalized networks.

Group normalized networks. We also observe that $\zeta(\phi_{GN};\mathbb{D})$ or $\zeta(\phi_{GW};\mathbb{D})$ is not related to m, which demonstrates that the batch size of an optimization algorithm does not affect the model's representational capacity for group normalized networks. Our analysis provides a new understanding of why GN is not sensitive to batch size [51], from the perspective of a model's representational capacity. Although unrelated to m, $\zeta(\phi_{GN};\mathbb{D})$ and $\zeta(\phi_{GW};\mathbb{D})$ are positively proportional to g, which suggests that the representational capacity of group normalized networks will decrease with increasing group number, according to Assumption 1. Note that a large group number contributes to the distribution stability of the normalized output, which may benefit training. Therefore, there exists a trade-off between the reduced model representational capacity and the increased learning efficiency, when increasing the group number.

We conduct experiments on MNIST with random labels [58]. The results are shown in Figure 2. We observe that the model with GN/GW has significantly degenerated training accuracy when g is too large, which means that a large group number heavily limits the model's representational capacity. We note that GW is more sensitive to the group number than GN. The main reason is that $\zeta(\phi_{GW};\mathbb{D})$ is quadratic to g, while $\zeta(\phi_{GN};\mathbb{D})$ is linear to it, from Table 1. We also observe that the best training accuracy of GW is higher than that of GN (85.86% *vs.* 81.40%). We attribute this to the fact that the whitening operation is better for improving the conditioning of optimization, compared to standardization.

4.2 Discussion of Previous Work

Previous analyses on BN are mainly derived from the perspective of optimization [40, 29, 24, 6]. One argument is that BN can improve the conditioning of the optimization problem [40, 6, 11, 22, 10], either by avoiding the rank collapse of pre-activation matrices [10] or alleviating the pathological sharpness of the landscape [40, 22]. This argument has been further investigated by computing the spectrum of the Hessian for a large-scale dataset [11]. The improved conditioning enables large learning rates, thus improving the generalization [5, 33]. Another argument is that BN is scale invariant [20, 3], enabling it to adaptively adjust the learning rate [8, 14, 1, 6, 59, 28], which stabilizes and further accelerates training [20, 3]. Other analyses focus on investigating the signal and gradient propagation, either by exploiting mean-field theory [54, 49], or a neural tangent kernel (NTK) [21].

Different from these works, we are the first to investigate how BN/GN affects a model's representational capacity, which opens new doors in analyzing and understanding normalization methods. We

[2]This assumption is intuitively reasonable, since the constraint number indicates the constraints (represented by equations) imposed on the normalized output in neural networks. The representational capacity of the model should be reduced, when provided with more constraints.

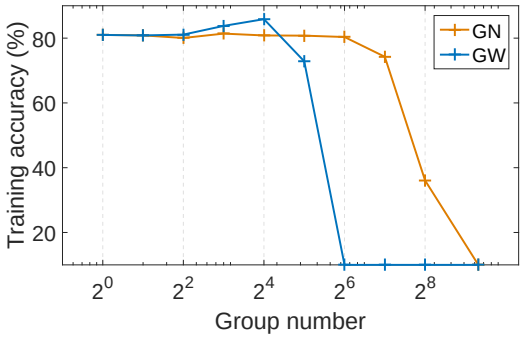
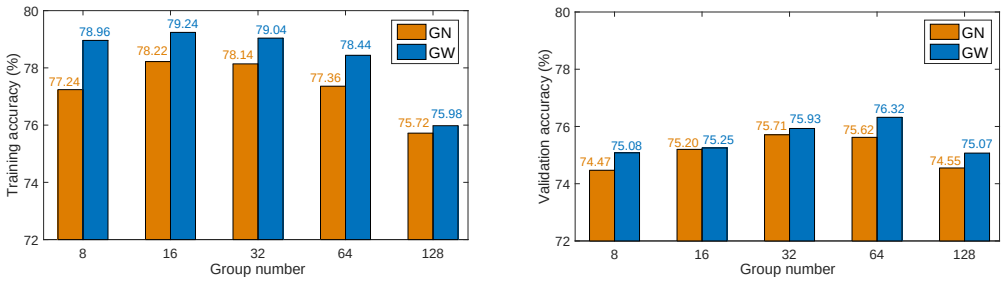


Figure 2: Effects of group number for group normalized networks. We train a four-layer MLP with 1280 neurons in each layer for MNIST, with random labels [58]. We train the model for 100 epochs with a batch size of 16. We vary the group number of GN/GW and evaluate the training accuracy.



(a) Training accuracy (b) Validation accuracy

Figure 3: Effects of group number of GW/GN on ResNet-50 for ImageNet classification. We evaluate the top-1 training and validation accuracies.

further investigate how batch size affects the training performance of batch normalized networks (Figure 1), from the perspective of a model's representational capacity. Several works [41, 18, 17] have shown that batch size is related to the magnitude of stochasticity [2, 46] introduced by BN, which also affects the model's training performance. However, the stochasticity analysis [18] is specific to normalization along the batch dimension, and cannot explain why GN with a large group number has significantly worse performance (Figure 2), while our work provides a unified analysis for batch and group normalized networks.

5 Experiments on Large-scale Visual Recognition Tasks

We investigate the effectiveness of our proposed GW on large-scale ImageNet classification [39], as well as COCO object detection and segmentation [31]. We use the more efficient and numerically stable 'ItN' (with $T = 5$) [18] to calculate the whitening matrix for both GW and BW, in all experiments. Our implementation is based on PyTorch [36].

5.1 ImageNet Classification

We experiment on the ImageNet dataset with 1000 classes [39]. We use the official 1.28M training images as a training set, and evaluate the top-1 accuracy on a single-crop of 224x224 pixels in the validation set with 50k images. We investigate the ResNet [13] and ResNeXt [52] models.

5.1.1 Ablation Study on ResNet-50

We follow the same experimental setup as described in [13], except that we use two GPUs and train over 100 epochs. We apply SGD with a mini-batch size of 256, momentum of 0.9 and weight decay of 0.0001. The initial learning rate is set to 0.1 and divided by 10 at 30, 60 and 90 epochs. Our baseline is the 50-layer ResNet (ResNet-50) trained with BN [20].

Effects of group number. We investigate the effects of group number for GW/GN, which we use to replace the BN of ResNet-50. We vary the group number g ranging in $\{8, 16, 32, 64, 128\}$ (we use the channel number if it is less than the group number in a given layer), and report the training and validation accuracies in Figure 3. We can see GW has consistent improvement over GN in training accuracy, across all values of g, which indicates the advantage of the whitening operation

Table 2: Effects of position when applying GW on ResNet-50 for ImageNet classification. We evaluate the top-1 validation accuracy on five architectures (**S1**, **S1-B1**, **S1-B2**, **S1-B3** and **S1-B12**).

	S1	S1-B1	S1-B2	S1-B3	S1-B12
Baseline (BN)	76.23	76.23	76.23	76.23	76.23
BW [18]	76.58 (↑0.35)	76.68 (↑0.45)	76.86 (↑0.63)	76.53 (↑0.30)	76.60 (↑0.37)
BW_Σ [17]	76.63 (↑0.40)	76.80 (↑0.57)	76.76 (↑0.53)	76.52 (↑0.29)	76.71 (↑0.48)
GW	**76.76** (↑0.53)	**77.62** (↑1.39)	**77.72** (↑1.49)	**77.47** (↑1.24)	77.45 (↑1.22)

Table 3: Comparison of validation accuracy on ResNets [13] and ResNeXts [52] for ImageNet. Note that we use an additional layer for BW_Σ to learn the decorrelated feature, as recommended in [17].

Method	ResNet-50	ResNet-101	ResNeXt-50	ResNeXt-101
Baseline (BN) [20]	76.23	77.69	77.01	79.29
GN [51]	75.71 (↓0.52)	77.20 (↓0.49)	75.69 (↓1.32)	78.00 (↓1.29)
BW_Σ [17]	77.21 (↑0.98)	78.27 (↑0.58)	77.29 (↑0.28)	79.43 (↑0.14)
GW	**77.72** (↑1.49)	**78.71** (↑1.02)	**78.43** (↑1.42)	**80.43** (↑1.14)

over standardization in optimization. Besides, GW also has better validation accuracy than GN. We believe this may be because the stronger constraints of GW contribute to generalization. We also observe that both GN and GW have significantly reduced training accuracy when the group number is too large (*e.g.*, g=128), which is consistent with the previous results in Figure 2.

Positions of GW. Although GW (g=64) provides slight improvement over the BN baseline (76.32% *vs.* 76.23%), it has a 90% additional time cost[3] on ResNet-50. Based on the analysis in Section C, it is reasonable to only partially replace BN with GW in networks, because 1) normalization within a batch or a group of channels both have their advantages in improving the generalization; 2) whitening can achieve better optimization efficiency and generalization than standardization [16], but at a higher computational cost [16, 18, 42].

Here, we investigate the position at which to apply GW (g=64) on ResNet-50. ResNet and ResNeXt are both mainly composed of a stem layer and multiple bottleneck blocks [13]. We consider: 1) replacing the BN in the stem layer with GW (referred to as 'S1'); 2) replacing the 1^{st}, 2^{nd}, 1^{st} & 2^{nd}, and 3^{rd} BNs in all the bottleneck blocks, which are referred to as 'B1', 'B2', 'B12' and 'B3', respectively. We investigate five architectures, **S1**, **S1-B1**, **S1-B2**, **S1-B3** and **S1-B12**, which have 1, 17, 17, 17 and 33 GW modules, respectively. We also perform experiments using BW [18] and BW_Σ [17] (employing a covariance matrix to estimate the population statistics of BW) for contrast.

We report the results in Table 2. BW/BW_Σ improve the BN counterpart on all architectures by a clear margin, which demonstrates the advantage of the whitening operation over standardization [18]. GW provides significant improvements over BW/BW_Σ on **S1-B1**, **S1-B2**, **S1-B3** and **S1-B12** (an absolute improvement of 0.9% on average). We attribute this to the advantage of GW in avoiding the estimation of population statistics. We also observe that GW has a slightly worse performance on **S1-B12** than on **S1-B1/S1-B2**. We believe there is a trade-off between GW and BN, in terms of affecting the model's representational capacity, optimization efficiency and generalization.

We also 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 [18]. This can slightly improve the performance (0.10% on average) when using GW, though the net gain is smaller than using BW (0.22%) or BW_Σ (0.43%). Please refer to the *Appendix D* for details.

5.1.2 Validation on Larger Models

In this section, we further validate the effectiveness of GW on ResNet-101 [13], ResNeXt-50 and ResNeXt-101 [52]. We apply GW (g=64) in these models following the **S1-B2** architecture, which achieves the best performance (Table 2) without significantly increasing the computational cost (it is only increased by roughly 23%). For comparison, we also apply BW_Σ following the 'S1-B2' architecture, combining the learning of decorrelated features [17] (BW_Σ has a slightly improved performance compared to BW [17]). Our baselines are the original networks trained with BN, and we also provide the results trained with GN.

The results are shown in Table 3. We can see that 1) our method improves the baseline (BN) by a significant margin (between 1.02% and 1.49%); 2) BW_Σ has consistently better performance than BN,

[3]Note that our implementations are based on the APIs provided by PyTorch and are not finely optimized. For more discussion on time costs, please refer to the *Appendix E*.

Table 4: Detection results (%) on COCO using the Faster R-CNN framework implemented in [34]. We use ResNet-50 as the backbone, combined with FPN. All models are trained by 1x lr scheduling (90k iterations), with a batch size of 16 on eight GPUs.

	2fc head box			4conv1fc head box		
Method	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}
$BN^\dagger$	36.31	58.39	38.83	36.39	57.22	39.56
GN	$36.62_{(\uparrow 0.31)}$	$58.91_{(\uparrow 0.52)}$	$39.32_{(\uparrow 0.49)}$	$37.86_{(\uparrow 1.47)}$	$58.96_{(\uparrow 1.74)}$	$40.76_{(\uparrow 1.20)}$
GW	$38.13_{(\uparrow 1.82)}$	$60.63_{(\uparrow 2.24)}$	$41.08_{(\uparrow 2.25)}$	$39.60_{(\uparrow 3.21)}$	$61.12_{(\uparrow 3.90)}$	$43.25_{(\uparrow 3.69)}$

Table 5: Detection and segmentation results (%) on COCO using the Mask R-CNN framework implemented in [34]. We use ResNeXt-101 as the backbone, combined with FPN. All models are trained by 2x lr scheduling (180k iterations), with a batch size of 8 on eight GPUs.

Method	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
$BN^\dagger$	42.24	63.00	46.19	37.53	59.82	39.96
GN	$42.18_{(\downarrow 0.06)}$	$63.22_{(\uparrow 0.22)}$	$46.00_{(\downarrow 0.19)}$	$37.54_{(\uparrow 0.01)}$	$60.18_{(\uparrow 0.36)}$	$39.99_{(\uparrow 0.03)}$
GW	$44.41_{(\uparrow 2.17)}$	$65.36_{(\uparrow 2.36)}$	$48.67_{(\uparrow 2.48)}$	$39.17_{(\uparrow 1.64)}$	$62.13_{(\uparrow 2.31)}$	$41.95_{(\uparrow 1.99)}$

but the net gain is reduced on wider networks (RexNeXt-50 and ResNeXt-101), which is probably caused by the difficulty in estimating the population statistics.

5.2 Object Detection and Segmentation on COCO

We fine-tune the models trained on ImageNet for object detection and segmentation on the COCO benchmark [31]. We experiment on the Faster R-CNN [38] and Mask R-CNN [12] frameworks using the publicly available codebase 'maskrcnn-benchmark' [34]. We train the models on the COCO $train2017$ set and evaluate on the COCO $val2017$ set. We report the standard COCO metrics of average precision (AP), AP_{50}, and AP_{75}, for bounding box detection (AP^{bbox}) and instance segmentation (AP^m) [31]. For BN, we use its frozen version (indicated by $BN^\dagger$) when fine-tuning for object detection [51].

Results on Faster R-CNN. For the Faster R-CNN framework, we use the ResNet-50 models pre-trained on ImageNet (Table 3) as the backbones, combined with the Feature Pyramid Network (FPN) [30]. We consider two setups: 1) we use the box head consisting of two fully-connected layers ('2fc') without a normalization layer, as proposed in [30]; 2) following [51], we replace the '2fc' box head with '4conv1fc', which can better leverage GN, and apply GN/GW to the FPN and box head. We use the default hypeparameter configurations from the training scripts provided by the codebase [34] for Faster R-CNN. The results are reported in Table 4. The GW pre-trained model improves $BN^\dagger$ and GN by 1.82% and 1.51% AP, respectively. By adding GW/GN to the FPN and '4conv1fc' head box, GW improves $BN^\dagger$ and GN by 3.21% and 1.74% AP, respectively.

Results on Mask R-CNN. For the Mask R-CNN framework, we use the ResNeXt-101 [52] models pre-trained on ImageNets (Table 3) as the backbones combined with FPN. We use the '4conv1fc' box head, and apply GN/GW to the FPN, box head and mask head. We again use the default hypeparameter configurations from the training scripts provided by the codebase for Mask R-CNN [34]. The results are shown in Table 5. GW achieves 44.41% in box AP and 39.17% in mask AP, an improvement over $BN^\dagger$ of 2.17% and 1.64%, respectively.

6 Conclusion and Further Work

In this paper, we proposed group whitening (GW), which combines the advantages of normalization within a group of channels and the whitening operation. The effectiveness of GW was validated on large-scale visual recognition tasks. Furthermore, we also provided a constraint analysis for normalization methods, enabling further understanding on how the batch size (group number) affects the performance of batch (group) normalized networks from the perspective of representational capacity. This constraint analysis can provide theoretical guidance for applying GW and other normalization methods in practice.

In the future, it would be interesting to investigate normalization along other dimensions (*e.g.*, positional normalization [27] and divisive normalization [37]) or other normalization operations (*e.g.*, scaling only [7, 57]), using our constraint analysis. We hope our analysis will provide a new means of understanding the behaviors of normalization methods.

Broader Impact

The proposed group whitening method could be applied to better train deep neural networks (DNNs), since it achieves high performance on visual recognition tasks, as validated by our experiments. Further, the proposed constraint analysis provides a new tool for analyzing and understanding normalization methods, which may contribute to the design of new DNN architectures and better understanding of DNN behaviors. As commonly acknowledged, DNNs are essential tools in a wide range of applications, *e.g.*, computer vision tasks and natural language process tasks. We thus believe the work in this paper will have a broad impact for researchers and engineers in these fields in particular. Finally, while our work may bring many important benefits, we also recognize that it could be leveraged by someone to carry out the research/projects that violate ethical standards, *e.g.*, generating biased or offensive texts and images using deep generative models, and detecting and tracking people without permission using deep discriminative models.

References

[1] Sanjeev Arora, Zhiyuan Li, and Kaifeng Lyu. Theoretical analysis of auto rate-tuning by batch normalization. In *ICLR*, 2019.

[2] Andrei Atanov, Arsenii Ashukha, Dmitry Molchanov, Kirill Neklyudov, and Dmitry Vetrov. Uncertainty estimation via stochastic batch normalization. In *ICLR Workshop*, 2018.

[3] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Dario A. Bini, Nicholas J. Higham, and Beatrice Meini. Algorithms for the matrix pth root. *Numerical Algorithms*, 39(4):349–378, Aug 2005.

[5] Johan Bjorck, Carla Gomes, and Bart Selman. Understanding batch normalization. In *NeurIPS*, 2018.

[6] Yongqiang Cai, Qianxiao Li, and Zuowei Shen. A quantitative analysis of the effect of batch normalization on gradient descent. In *ICML*, 2019.

[7] Vitaliy Chiley, Ilya Sharapov, Atli Kosson, Urs Koster, Ryan Reece, Sofia Samaniego de la Fuente, Vishal Subbiah, and Michael James. Online normalization for training neural networks. In *NeurIPS*. 2019.

[8] Minhyung Cho and Jaehyung Lee. Riemannian approach to batch normalization. In *NeurIPS*, 2017.

[9] Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron C. Courville. Recurrent batch normalization. In *ICLR*, 2017.

[10] Hadi Daneshmand, Jonas Kohler, Francis Bach, Thomas Hofmann, and Aurelien Lucchi. Theoretical understanding of batch-normalization: A markov chain perspective. *arXiv preprint arXiv:2003.01652*, 2020.

[11] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *ICML*, 2019.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017.

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

[14] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *NeurIPS*, 2018.

[15] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

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

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

[18] Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. In *CVPR*, 2019.

[19] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *NeurIPS*, 2017.

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

[21] Arthur Jacot, Franck Gabriel, and Clément Hongler. Freeze and chaos for dnns: an ntk view of batch normalization, checkerboard and boundary effects. *arXiv preprint arXiv:1907.05715*, 2019.

[22] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. The normalization method for alleviating pathological sharpness in wide neural networks. In *NeurIPS*. 2019.

[23] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314, 2018.

[24] Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In *AISTATS*, 2019.

[25] César Laurent, Gabriel Pereyra, Philemon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *ICASSP*, 2016.

[26] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50, 1998.

[27] Boyi Li, Felix Wu, Kilian Q Weinberger, and Serge Belongie. Positional normalization. In *NeurIPS*, 2019.

[28] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for batch normalized networks. In *ICLR*, 2020.

[29] Xiangru Lian and Ji Liu. Revisit batch normalization: New understanding and refinement via composition optimization. In *AISTATS*, 2019.

[30] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[32] Ping Luo, Jiamin Ren, and Zhanglin Peng. Differentiable learning-to-normalize via switchable normalization. *arXiv preprint arXiv:1806.10779*, 2018.

[33] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. In *ICLR*, 2019.

[34] Francisco Massa and Ross Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018. Accessed: 09-26-2019.

[35] Xingang Pan, Xiaohang Zhan, Jianping Shi, Xiaoou Tang, and Ping Luo. Switchable whitening for deep representation learning. In *ICCV*, 2019.

[36] 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.

[37] Mengye Ren, Renjie Liao, Raquel Urtasun, Fabian H. Sinz, and Richard S. Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. In *ICLR*, 2017.

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[40] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *NeurIPS*, 2018.

[41] Alexander Shekhovtsov and Boris Flach. Stochastic normalizations as bayesian learning. In *ACCV*, 2018.

[42] Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening and coloring batch transform for gans. In *ICLR*, 2019.

[43] Saurabh Singh and Abhinav Shrivastava. Evalnorm: Estimating batch normalization statistics for evaluation. In *ICCV*, 2019.

[44] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[46] Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *ICML*, 2018.

[47] 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.

[48] Guangrun Wang, Jiefeng Peng, Ping Luo, Xinjiang Wang, and Liang Lin. Kalman normalization: Normalizing internal representations across network layers. In *NeurIPS*, 2018.

[49] Mingwei Wei, James Stokes, and David J. Schwab. Mean-field analysis of batch normalization. *arXiv preprint arXiv:1903.02606*, 2019.

[50] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. In *NeurIPS*, 2011.

[51] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

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

[53] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. In *NeurIPS*, 2019.

[54] Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of batch normalization. In *ICLR*, 2019.

[55] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. In *ICLR*, 2018.

[56] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[57] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *NeurIPS*, 2019.

[58] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

[59] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger B. Grosse. Three mechanisms of weight decay regularization. In *ICLR*, 2019.

Algorithm 1 The forward pass of group whitening.

1: **Input**: a input sample $\mathbf{x} \in \mathbb{R}^d$.
2: **Hyperparameters**: ϵ, 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: $\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \Pi^{-1}(\frac{\partial \mathcal{L}}{\partial \mathbf{X}_G}) \in \mathbb{R}^d$.

A 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[4]. Note that we need to specify the method to calculate 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') [18]. Here, we provide the details.

ZCA whitening. ZCA whitening [16] 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 $\Lambda = \text{diag}(\sigma_1, \ldots, \sigma_d)$ and $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_d]$ are the eigenvalues and associated eigenvectors of Σ, *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:

$$\frac{\partial \mathcal{L}}{\partial \Lambda} = \mathbf{D}^T(\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})\mathbf{D}(-\frac{1}{2}\Lambda^{-3/2}) \tag{13}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{D}} = (\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}} + (\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})^T)\mathbf{D}\Lambda^{-1/2} \tag{14}$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma} = \mathbf{D}\{(\mathbf{K}^T \odot (\mathbf{D}^T\frac{\partial \mathcal{L}}{\partial \mathbf{D}})) + (\frac{\partial \mathcal{L}}{\partial \Lambda})_{diag}\}\mathbf{D}^T, \tag{15}$$

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 [18] 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 Σ_d and $\mathbf{P}_T$ is calculated iteratively as:

$$\begin{cases}\mathbf{P}_0 = \mathbf{I} \\ \mathbf{P}_k = \frac{1}{2}(3\mathbf{P}_{k-1} - \mathbf{P}^3_{k-1}\Sigma^N_d), \ \ k = 1, 2, ..., T.\end{cases} \tag{16}$$

[4]For GW, we also use the extra learnable dimension-wise scale and shift parameters, like BN [20]. We omit this in the algorithms for simplicity.

```
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 4: Python code of GW using ItN whitening, based on PyTorch.

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

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

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{P}_T} =& \frac{1}{\sqrt{tr(\Sigma)}} \frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}} \\
\frac{\partial \mathcal{L}}{\partial \Sigma_N} =& -\frac{1}{2} \sum_{k=1}^{T} (\mathbf{P}_{k-1}^3)^T \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k} \\
\frac{\partial \mathcal{L}}{\partial \Sigma} =& \frac{1}{tr(\Sigma)} \frac{\partial \mathcal{L}}{\partial \Sigma_N} - \frac{1}{(tr(\Sigma))^2} tr(\frac{\partial \mathcal{L}}{\partial \Sigma_N}^T \Sigma) \mathbf{I} \\
& - \frac{1}{2(tr(\Sigma))^{3/2}} tr((\frac{\partial \mathcal{L}}{\partial \Sigma^{-\frac{1}{2}}})^T \mathbf{P}_T) \mathbf{I}.
\end{aligned} \tag{17}$$

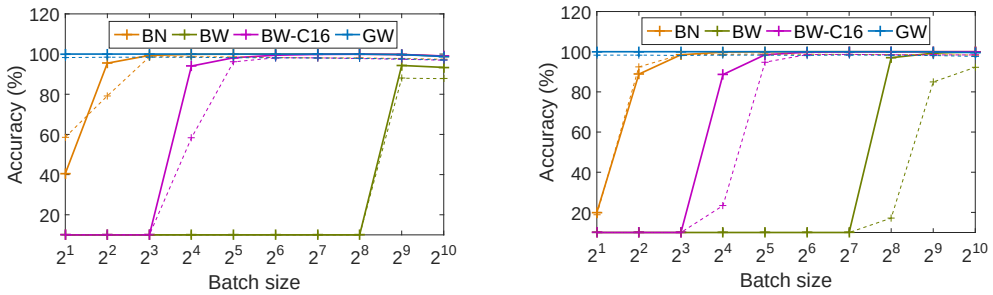
Here, $\frac{\partial \mathcal{L}}{\partial \mathbf{P}_k}$ can be calculated by the following iterations:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{P}_{k-1}} =& \frac{3}{2} \frac{\partial L}{\partial \mathbf{P}_k} - \frac{1}{2} \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k} (\mathbf{P}_{k-1}^2 \Sigma_N)^T - \frac{1}{2} (\mathbf{P}_{k-1}^2)^T \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k} \Sigma_N^T \\
& - \frac{1}{2} (\mathbf{P}_{k-1})^T \frac{\partial \mathcal{L}}{\partial \mathbf{P}_k} (\mathbf{P}_{k-1} \Sigma_N)^T, \ \ k = T, ..., 1.
\end{aligned} \tag{18}$$

We also provide the python code of GW using ItN whitening, based on PyTorch [36], in Figure 4.

B More Results on Effects of Batch Size

In Figure 1 of the submitted paper, we show the effects of batch size for different normalization methods, where the results are obtained with a learning rate of 0.1. Here, we provide more results using different learning rates, shown in Figure 5. We obtain similar observations.



(a) Learning rate=0.01 (b) Learning rate=0.5

Figure 5: Effects of batch size for different normalization methods. We train a four-layer multilayer perceptron (MLP) with 256 neurons in each layer, for MNIST classification. We compare BN, BW, group-based BW with 16 neurons/channels in each group ('BW-C16'), and GW (we use a group number of 16). We vary the batch size and evaluate the training (thick 'plus' with solid line) and validation (thin 'plus' with dashed line) accuracy at the end of 50 training epochs. These results are obtained using a learning rate of (a) 0.01 and (b) 0.5.

C 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 at the submitted paper. Here, we provide the details for deriving the constraint number of batch whitening (BW), group normalization(GN) [51] and our proposed GW, for the mini-batch input $\mathbf{X} \in \mathbb{R}^{d\times m}$.

Constraint number of BW. BW [16] ensures that the normalized output is centered and whitened, which has the constraints $\Upsilon_{\phi_{BW}}(\widehat{\mathbf{X}})$ as:

$$\widehat{\mathbf{X}}\mathbf{1} = \mathbf{0}_d, \quad and \tag{19}$$

$$\widehat{\mathbf{X}}\widehat{\mathbf{X}}^T - m\mathbf{I} = \mathbf{0}_{d\times d}, \tag{20}$$

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:

$$\widehat{\mathbf{Z}} = \Lambda_g^{-\frac{1}{2}}(\mathbf{Z} - \mu_g\mathbf{1}^T), \tag{21}$$

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

$$\sum_{j=1}^{c}\widehat{\mathbf{Z}}_{ij} = 0 \; and \; \sum_{j=1}^{c}\widehat{\mathbf{Z}}_{ij}^2 = c, \; for \; each \; group \; i = 1, ..., g. \tag{22}$$

In the system of equations 22, the number of independent equations is $2g$. Therefore, the constraint number of GN is $2dm$, when given m samples.

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

$$Group \; division: \quad \mathbf{X}_G = \Pi(\mathbf{x}; g) \in \mathbb{R}^{g\times c}, \tag{23}$$

$$Whitening: \quad \widehat{\mathbf{X}}_G = \phi^W(\mathbf{X}_G) = \Sigma_g^{-\frac{1}{2}}(\mathbf{X}_G - \mu_g\mathbf{1}^T), \tag{24}$$

$$Inverse \; group \; division: \quad \hat{\mathbf{x}} = \Pi^{-1}(\widehat{\mathbf{X}}_G) \in \mathbb{R}^d. \tag{25}$$

Table 6: Effects of inserting a GW/BW/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)	76.24	76.24	76.24	76.24	76.24
BW [18]	76.91 (↑0.67)	76.94 (↑0.70)	76.93 (↑0.69)	76.78 (↑0.54)	76.79 (↑0.55)
BW$_\Sigma$ [17]	**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 7: 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.

	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%)

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

$$\widehat{\mathbf{X}}_G\mathbf{1} = \mathbf{0}, \quad and \tag{26}$$

$$\widehat{\mathbf{X}}_G\widehat{\mathbf{X}}_G^T - c\mathbf{I} = \mathbf{0}. \tag{27}$$

Following the analysis for BW, the number of independent equations is $g(g+3)/2$ from Eqns. 26 and 27. Therefore, the constraint number of GW is $mg(g+3)/2$, when given m samples.

D 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 [18]. We provide the results in Table 6. This can slightly improve the performance (0.10% on average) when using GW (comparing Table 6 to Table 2 of the submitted paper). We note that BW_Σ benefits the most from this kinds of architecture.

E 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) 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 7 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 $4\times$ larger than that of the first/second normalization layer, in the bottleneck blocks [13].

Table 8 shows the time costs of ResNets [13] and ResNeXts [52] (the corresponding models in Table 3 of the submitted paper) for ImageNet classification.

Table 8: Time costs (ms) of ResNets [13] and ResNeXts [52] for ImageNet classification. Note that $\Delta\ x\%$ indicates the additional time cost is $x\%$, compared to the baselines.

Method	ResNet-50	ResNet-101	ResNeXt-50	ResNeXt-101
Baseline (BN) [20]	419	672	574	912
BW$_\Sigma$ [18]	550 (Δ31.3%)	882 (Δ30.9%)	798 (Δ39.0%)	1587 (Δ74.0%)
GW	514 (Δ22.7%)	810 (Δ20.5%)	738 (Δ28.6%)	1180 (Δ29.3%)