Group Normalization

Yuxin Wu Kaiming He

Facebook AI Research (FAIR)

Abstract Batch Normalization (BN) is a milestone technique in the development of deep learning, enabling various networks to train. However, normalizing along the batch dimension introduces problems — BN's error increases rapidly when the batch size becomes smaller, caused by inaccurate batch statistics estimation. This limits BN's usage for training larger models and transferring features to computer vision tasks including detection, segmentation, and video, which require small batches constrained by memory consumption. In this paper, we present Group Normalization (GN) as a simple alternative to BN. GN divides the channels into groups and computes within each group the mean and variance for normalization. GN's computation is independent of batch sizes, and its accuracy is stable in a wide range of batch sizes. On ResNet-50 trained in ImageNet, GN has 10.6% lower error than its BN counterpart when using a batch size of 2; when using typical batch sizes, GN is comparably good with BN and outperforms other normalization variants. Moreover, GN can be naturally transferred from pre-training to fine-tuning. GN can outperform its BN-based counterparts for object detection and segmentation in COCO, and for video classification in Kinetics, showing that GN can effectively replace the powerful BN in a variety of tasks. GN can be easily implemented by a few lines of code.

1 Introduction

Batch Normalization (Batch Norm or BN) [1] has been established as a very effective component in deep learning, largely helping push the frontier in computer vision [2,3] and beyond [4]. BN normalizes the features by the mean and variance computed within a (mini-)batch. This has been shown by many practices to ease optimization and enable very deep networks to converge. The stochastic uncertainty of the batch statistics also acts as a regularizer that can benefit generalization. BN has been a foundation of many state-of-the-art algorithms.

Despite its great success, BN exhibits drawbacks that are also caused by its distinct behavior of normalizing along the batch dimension. In particular, it is required for BN to work with a *sufficiently large batch size* (*e.g.*, 32 per worker[1] [1,2,3]). A small batch leads to inaccurate estimation of the batch statistics, and *reducing BN's batch size increases the model error dramatically* (Figure 1).

[1] In the context of this paper, we use "batch size" to refer to the number of samples *per worker* (*e.g.*, GPU). BN's statistics are computed for each worker, but *not* broadcast across workers, as is standard in many libraries.

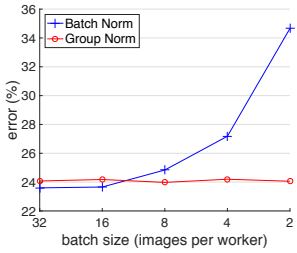


Figure 1. ImageNet classification error *vs*. batch sizes. The model is ResNet-50 trained in the ImageNet training set using 8 workers (GPUs) and evaluated in the validation set. BN's error increases rapidly when reducing the batch size. GN's computation is independent of batch sizes, and its error rate is stable despite the batch size changes. GN has substantially lower error (by 10%) than BN with a batch size of 2.

As a result, many recent models [2,3,5,6,7] are trained with non-trivial batch sizes that are memory-consuming. The heavy reliance on BN's effectiveness to train models in turn prohibits people from exploring higher-capacity models that would be limited by memory.

The restriction on batch sizes is more demanding in computer vision tasks including detection [8,9,10], segmentation [11,10], video recognition [12,13], and other high-level systems built on them. E.g., the Fast/er and Mask R-CNN frameworks [8,9,10] use a batch size of 1 or 2 images because of higher resolution, where BN is "frozen" by transforming to a linear layer [3]; in video classification with 3D convolutions [12,13], the presence of spatial-temporal features introduces a trade-off between the temporal length and batch size. The usage of BN often requires these systems to compromise between the model design and batch sizes.

This paper presents Group Normalization (GN) as a simple alternative to BN. We notice that many classical features like SIFT [14] and HOG [15] are *group-wise* features and involve *group-wise normalization*. For example, a HOG vector is the outcome of several spatial cells where each cell is represented by a normalized orientation histogram. Analogously, we propose GN as a layer that divides channels into groups and normalizes the features within each group (Figure 2). GN does not exploit the batch dimension, and its computation is independent of batch sizes.

GN behaves very stably over a wide range of batch sizes (Figure 1). With a batch size of 2 samples, GN has 10.6% lower error than its BN counterpart for ResNet-50 [3] in ImageNet [16]. With a regular batch size, GN is comparably good as BN (with a gap of ~0.5%) and outperforms other normalization variants [17,18,19]. Moreover, although the batch size may change, GN can naturally transfer from pre-training to fine-tuning. GN shows improved results *vs*. its BN counterpart on Mask R-CNN for COCO object detection and segmentation [20], and on 3D convolutional networks for Kinetics video classification [21]. The effectiveness of GN in ImageNet, COCO, and Kinetics demonstrates that GN is a competitive alternative to BN that has been dominant in these tasks.

There have been existing methods, such as Layer Normalization (LN) [17] and Instance Normalization (IN) [18] (Figure 2), that also avoid normalizing along the batch dimension. These methods are effective for training sequential models (RNN/LSTM [22,23]) or generative models (GANs [24,25]). But as we will show by experiments, both LN and IN have limited success in visual recognition, for

which GN presents better results. Conversely, GN could be used in place of LN and IN and thus is applicable for sequential or generative models. This is beyond the focus of this paper, but it is suggestive for future research.

2 Related Work

Normalization. Normalization layers in deep networks had been widely used before the development of BN. Local Response Normalization (LRN) [26,27,28] was a component in AlexNet [28] and following models [29,30,31]. LRN computes the statistics in a small neighborhood for each pixel.

Batch Normalization [1] performs more global normalization along the batch dimension (and as importantly, it suggests to do this for all layers). But the concept of "batch" is not always present, or it may change from time to time. For example, batch-wise normalization is not legitimate at inference time, so the mean and variance are pre-computed from the training set [1], often by running average; consequently, there is no normalization performed when testing. The pre-computed statistics may also change when the target data distribution changes [32]. These issues lead to inconsistency at training, transferring, and testing time. In addition, as aforementioned, reducing the batch size can have dramatic impact on the estimated batch statistics.

Several normalization methods [17,18,19,33,34] have been proposed to avoid exploiting the batch dimension. Layer Normalization (LN) [17] operates along the channel dimension, and Instance Normalization (IN) [18] performs BN-like computation but only for each sample (Figure 2). Instead of operating on features, Weight Normalization (WN) [19] proposes to normalize the filter weights. These methods do not suffer from the issues caused by the batch dimension, but they have not been able to approach BN's accuracy in many visual recognition tasks. We compare with these methods in context of the remaining sections.

Addressing small batches. Ioffe [35] proposes Batch Renormalization (BR) that alleviates BN's issue involving small batches. BR introduces two extra parameters that constrain the estimated mean and variance of BN within a certain range, reducing their drift when the batch size is small. BR has better accuracy than BN in the small-batch regime. But BR is also batch-dependent, and when the batch size decreases its accuracy still degrades [35].

There are also attempts to *avoid* using small batches. The object detector in [36] performs synchronized BN whose mean and variance are computed across multiple GPUs. However, this method does not solve the problem of small batches; instead, it migrates the algorithm problem to engineering and hardware demands, using a number of GPUs proportional to BN's requirements. Moreover, the synchronized BN computation prevents using *asynchronous* solvers (ASGD [37]), a practical solution to large-scale training widely used in industry. These issues can limit the scope of using synchronized BN.

Instead of addressing the batch statistics computation (*e.g.*, [35,36]), our normalization method inherently avoids this computation.

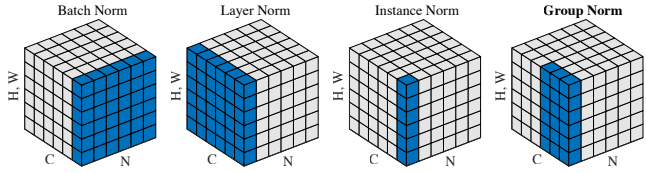


Figure 2. Normalization methods. Each subplot shows a feature map tensor. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. Group Norm is illustrated using a group number of 2.

Group-wise computation. *Group convolutions* have been presented by AlexNet [28] for distributing a model into two GPUs. The concept of *groups* as a dimension for model design has been more widely studied recently. The work of ResNeXt [7] investigates the trade-off between depth, width, and groups, and it suggests that a larger number of groups can improve accuracy under similar computational cost. MobileNet [38] and Xception [39] exploit *channel-wise* (also called "depth-wise") convolutions, which are group convolutions with a group number equal to the channel number. ShuffleNet [40] proposes a channel shuffle operation that permutes the axes of grouped features. These methods all involve dividing the channel dimension into groups. Despite the relation to these methods, GN does *not* require group convolutions. GN is a generic layer, as we evaluate in standard ResNets [3].

3 Group Normalization

The channels of visual representations are not entirely independent. Classical features of SIFT [14], HOG [15], and GIST [41] are *group-wise* representations by design, where each group of channels is constructed by some kind of histogram. These features are often processed by *group-wise normalization* over each histogram or each orientation. Higher-level features such as VLAD [42] and Fisher Vectors (FV) [43] are also group-wise features where a group can be thought of as the sub-vector computed with respect to a cluster.

Analogously, it is not necessary to think of deep neural network features as unstructured vectors. For example, for conv_1 (the first convolutional layer) of a network, it is reasonable to expect a filter and its horizontal flipping to exhibit similar distributions of filter responses on natural images. If conv_1 happens to approximately learn this pair of filters, or if the horizontal flipping (or other transformations) is made into the architectures by design [44,45], then the corresponding channels of these filters can be normalized together.

The higher-level layers are more abstract and their behaviors are not as intuitive. However, in addition to orientations (SIFT [14], HOG [15], or [44,45]), there are many factors that could lead to grouping, *e.g.*, frequency, shapes, illumination, textures. Their coefficients can be interdependent. In fact, a well-accepted computational model in neuroscience is to normalize across the cell

responses [46,47,48,49], "with various receptive-field centers (covering the visual field) and with various spatiotemporal frequency tunings" (p183, [46]); this can happen not only in the primary visual cortex, but also "throughout the visual system" [49]. Motivated by these works, we propose new generic group-wise normalization for deep neural networks.

Formulation. We first describe a general formulation of feature normalization, and then present GN in this formulation. A family of feature normalization methods, including BN, LN, IN, and GN, perform the following computation:

$$\hat{x}_i = \frac{1}{\sigma_i}(x_i - \mu_i). \tag{1}$$

Here x is the feature computed by a layer, and i is an index. In the case of 2D images, $i = (i_N, i_C, i_H, i_W)$ is a 4D vector indexing the features in (N, C, H, W) order, where N is the batch axis, C is the channel axis, and H and W are the spatial height and width axes.

μ and σ in (1) are the mean and standard deviation (std) computed by:

$$\mu_i = \frac{1}{m}\sum_{k\in\mathcal{S}_i} x_k, \quad \sigma_i = \sqrt{\frac{1}{m}\sum_{k\in\mathcal{S}_i}(x_k - \mu_i)^2 + \epsilon}, \tag{2}$$

with ϵ as a small constant. $\mathcal{S}_i$ is the set of pixels in which the mean and std are computed, and m is the size of this set. Many types of feature normalization methods mainly differ in how the set $\mathcal{S}_i$ is defined (Figure 2), discussed as follows.

In **Batch Norm** [1], the set $\mathcal{S}_i$ is defined as:

$$\mathcal{S}_i = \{k \mid k_C = i_C\}, \tag{3}$$

where i_C (and k_C) denotes the sub-index of i (and k) along the C axis. This means that the pixels sharing the same channel index are normalized together, *i.e.*, for each channel, BN computes μ and σ along the (N, H, W) axes. In **Layer Norm** [17], the set is:

$$\mathcal{S}_i = \{k \mid k_N = i_N\}, \tag{4}$$

meaning that LN computes μ and σ along the (C, H, W) axes for each sample. In **Instance Norm** [18], the set is:

$$\mathcal{S}_i = \{k \mid k_N = i_N, k_C = i_C\}. \tag{5}$$

meaning that IN computes μ and σ along the (H, W) axes for each sample and each channel. The relations among BN, LN, and IN are in Figure 2.

As in [1], all methods of BN, LN, and IN learn a per-channel linear transform to compensate for the possible lost of representational ability:

$$y_i = \gamma\hat{x}_i + \beta, \tag{6}$$

where γ and β are trainable scale and shift (indexed by i_C in all case, which we omit for simplifying notations).

```
def GroupNorm(x, gamma, beta, G, eps=1e−5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: learnable scale and offset, with shape [1,C,1,1]
    # G: number of groups for GN

    N, C, H, W = x.shape
    x = tf.reshape(x, [N, G, C // G, H, W])

    mean, var = tf.nn.moments(x, [2, 3, 4], keep_dims=True)
    x = (x − mean) / tf.sqrt(var + eps)

    x = tf.reshape(x, [N, C, H, W])

    return x ∗ gamma + beta
```

Figure 3. Python code of Group Norm based on TensorFlow.

Group Norm. A Group Norm layer computes μ and σ in $\mathcal{S}_i$ defined as:

$$\mathcal{S}_i = \{k \mid k_N = i_N, \lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor\}. \tag{7}$$

Here G is the number of groups, which is a pre-defined hyper-parameter ($G = 32$ by default). C/G is the number of channels per group. $\lfloor \cdot \rfloor$ is the floor operation, and "$\lfloor \frac{k_C}{C/G} \rfloor = \lfloor \frac{i_C}{C/G} \rfloor$" means that the indexes i and k are in the same group of channels, assuming each group of channels are stored in a sequential order along the C axis. GN computes μ and σ along the (H, W) axes and along a group of $\frac{C}{G}$ channels. The computation of GN is illustrated in Figure 2 (rightmost), which is a simple case of 2 groups ($G = 2$) each having 3 channels.

Given $\mathcal{S}_i$ in Eqn.(7), a GN layer is defined by Eqn.(1), (2), and (6). Specifically, the pixels in the same group are normalized together by the same μ and σ. GN also learns the per-channel γ and β.

Relation to Prior Work. LN, IN, and GN all perform independent computations along the batch axis. The two extreme cases of GN are equivalent to LN and IN (Figure 2).

GN becomes LN [17] if we set the group number as $G = 1$. LN assumes *all* channels in a layer make "similar contributions" [17]. Unlike the case of fully-connected layers studied in [17], this assumption can be less valid with the presence of convolutions, as discussed in [17]. GN is less restricted than LN, because each group of channels (instead of all of them) are assumed to subject to the shared mean and variance; the model still has flexibility of learning a different distribution for each group. This leads to improved representational power of GN over LN, as shown by the lower training and validation error in experiments (Figure 4).

GN becomes IN [18] if we set the group number as $G = C$ (*i.e.*, one channel per group). But IN can only rely on the spatial dimension for computing the mean and variance and it misses the opportunity of exploiting the channel dependence.

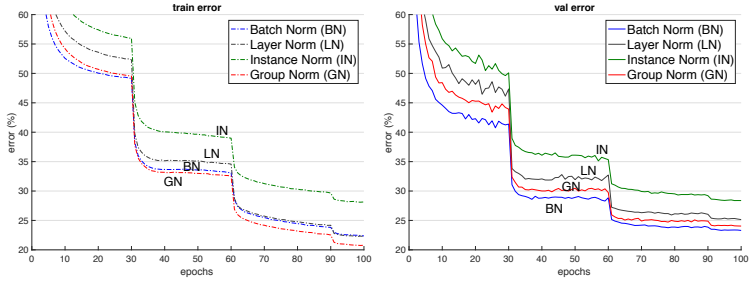


Figure 4. Comparison of error curves with a batch size of **32 images**/GPU. We show the ImageNet training error (left) and validation error (right) *vs.* numbers of training epochs. The model is ResNet-50.

	BN	LN	IN	GN
val error (%)	**23.6**	25.3	28.4	24.1
△ (*vs.* BN)	-	*1.7*	*4.8*	***0.5***

Table 1. Comparison of error rates with a batch size of **32 images**/GPU, on ResNet-50 in the ImageNet validation set. The error curves are in Figure 4.

Implementation. GN can be easily implemented by a few lines of code in PyTorch [50] and TensorFlow [51] where automatic differentiation is supported. Figure 3 shows the code based on TensorFlow. In fact, we only need to specify how the mean and variance ("moments") are computed, along the appropriate axes as defined by the normalization method.

4 Experiments

4.1 Image Classification in ImageNet

Implementation details. As standard practice [3,52], we use 8 GPUs to train all models, and the batch mean and variance of BN are computed *within* each GPU. We use the method of [53] to initialize all convolutions for all models. We use 1 to initialize all γ parameters, except for each residual block's last normalization layer where we initialize γ by 0 following [54] (such that the initial state of a residual block is identity). We use a weight decay of 0.0001 for all weight layers, including γ and β (following [52] but unlike [3,54]). We train 100 epochs for all models, and decrease the learning rate by 10× at 30, 60, and 90 epochs. During training, we adopt the data augmentation of [31] as implemented by [52]. We evaluate the top-1 classification error on the center crops of 224×224 pixels in the validation set. To reduce random variations, we report the median error rate of the final 5 epochs [54]. Other implementation details follow [52].

Our baseline is the ResNet trained with BN [3]. To compare with LN, IN, and GN, we replace BN with the specific variant. We use the same hyper-parameters for all models. We set $G = 32$ for GN by default.

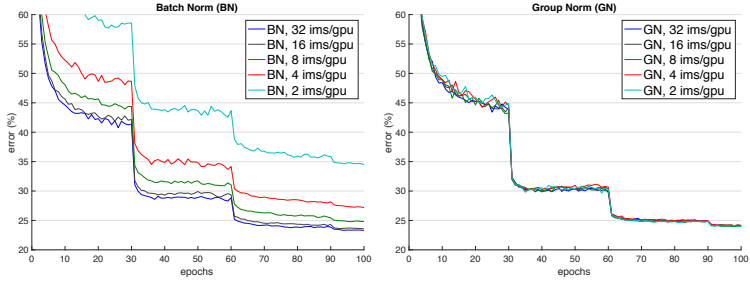


Figure 5. Sensitivity to batch sizes. We show ResNet-50's validation error of BN (left) and LN (right) using a batch size of 32, 16, 8, 4, and 2 images/GPU.

batch size	32	16	8	4	2
BN	**23.6**	**23.7**	24.8	27.3	34.7
GN	24.1	24.2	**24.0**	**24.2**	**24.1**
△	*0.5*	*0.5*	*-0.8*	*-3.1*	*-10.6*

Table 2. Sensitivity to batch sizes. We show ResNet-50's validation error (%) in ImageNet. The last row shows the differences between BN and GN. The error curves are in Figure 5. This table is visualized in Figure 1.

Comparison of feature normalization methods. We first experiment with a regular batch size of **32 images** (per GPU) [1,3]. BN works successfully in this regime, so this is a strong baseline to compare with. Figure 4 shows the error curves, and Table 1 shows the final results. *All* of these normalization methods are able to converge. LN has a small degradation of 1.7% comparing with BN. This is an encouraging result, as it suggests that normalizing along *all* channels (as done by LN) of a *convolutional* network is reasonably good. IN also makes the model converge, but is 4.8% worse than BN.

In this regime where BN works well, GN is able to approach BN's accuracy, with a decent degradation of 0.5% in the validation set. Actually, Figure 4 (left) shows that GN has *lower training error* than BN, indicating that GN is effective for easing *optimization*. The slightly higher validation error of GN implies that GN loses some regularization ability of BN. This is understandable, because BN's mean and variance computation introduces uncertainty caused by the stochastic batch sampling, which helps regularization [1]. This uncertainty is missing in GN (and LN/IN). But it is possible that GN combined with a suitable regularizer will improve results. This can be a future research topic.

Small batch sizes. Although BN benefits from the stochasticity under some situations, its error increases when the batch size becomes smaller and the uncertainty gets bigger. We show this in Figure 1, Figure 5, and Table 2.

We evaluate batch sizes of 32, 16, 8, 4, 2 images per GPU. In all cases, the BN mean and variance are computed within each GPU and not synchronized.

# groups (G)						
64	32	16	8	4	2	1 (=LN)
24.6	**24.1**	24.6	24.4	24.6	24.7	25.3
0.5	-	*0.5*	*0.3*	*0.5*	*0.6*	*1.2*

channels per group						
64	32	16	8	4	2	1 (=IN)
24.4	24.5	**24.2**	24.3	24.8	25.6	28.4
0.2	*0.3*	-	*0.1*	*0.6*	*1.4*	*4.2*

Table 3. Group division. We show ResNet-50's validation error (%) in ImageNet, trained with 32 images/GPU. (Left): a given number of groups. (Right): a given number of channels per group. The last rows show the differences with the best number.

All models are trained in 8 GPUs. In this set of experiments, we adopt the linear learning rate scaling rule [55,56,54] to adapt to batch size changes — we use a learning rate of 0.1 [3] for the batch size of 32, and $0.1N/32$ for a batch size of N. This linear scaling rule works well for BN if the total batch size changes (by changing the number of GPUs) but the per-GPU batch size does not change [54]. We keep the same number of training epochs for all cases (Figure 5, x-axis). All other hyper-parameters are unchanged.

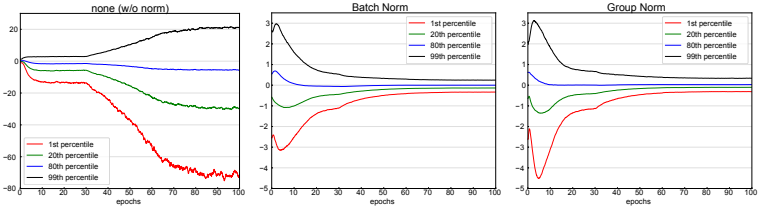
Figure 5 (left) shows that BN's error becomes considerably higher with small batch sizes. GN's behavior is more stable and insensitive to the batch size. Actually, Figure 5 (right) shows that GN has very similar curves (subject to random variations) across a wide range of batch sizes from 32 to 2. For a batch size of 2, GN has **10.6%** lower error rate than its BN counterpart (24.1% *vs*. 34.7%).

These results indicate that the batch mean and variance estimation can be overly stochastic and inaccurate, especially when they are computed over 4 or 2 images. However, this stochasticity disappears if the statistics are computed from 1 image, in which case BN becomes similar to IN at training time. We see that IN has a better result (28.4%) than BN with a batch size of 2 (34.7%).

The robust results of GN in Table 2 demonstrate GN's strength. It allows to remove the batch size constraint imposed by BN, which can give considerably more memory (*e.g.*, 16× or more). This will make it possible to train higher-capacity models that would be otherwise bottlenecked by memory limitation. We hope this will create new opportunities in architecture design.

Comparison with Batch Renorm (BR). BR [35] introduces two extra parameters (r and d in [35]) that constrain the estimated mean and variance of BN. Their values are controlled by $r_{\max}$ and $d_{\max}$. To apply BR to ResNet-50, we have carefully chosen these hyper-parameters, and found that $r_{\max} = 1.5$ and $d_{\max} = 0.5$ work best for ResNet-50. With a batch size of 4, ResNet-50 trained with BR has an error rate of 26.3%. This is better than BN's 27.3%, but still 2.1% higher than GN's 24.2%.

Group division. Thus far all presented GN models are trained with a group number of $G = 32$. Next we evaluate different ways of dividing into groups. With a given fixed group number, GN performs reasonably well for all values of G we



	err.
none	29.2
BN	28.0
GN	**27.6**

Figure 6. Evolution of feature distributions of conv_{5_3}'s output features (before normalization and ReLU) from VGG-16, shown as the {1, 20, 80, 99} percentile of responses, with no normalization, BN, and GN. The table on the right shows the classification error (%) in ImageNet validation. Models are trained with 32 images/GPU.

studied (Table 3, right panel). In the extreme case of $G = 1$, GN is equivalent to LN, and its error rate is higher than all cases of $G > 1$ studied.

We also evaluate fixing the number of channels per group (Table 3, left panel). Note that because the layers can have different channel numbers, the group number G can change across layers in this setting. In the extreme case of 1 channel per group, GN is equivalent to IN. Even if using as few as 2 channels per group, GN has substantially lower error than IN (25.6% *vs*. 28.4%). This result shows the effect of grouping channels when performing normalization.

Deeper models. We have also compared GN with BN on ResNet-101 [3]. With a batch size of 32, our BN baseline of ResNet-101 has 22.0% validation error, and the GN counterpart has 22.4%, slightly worse by 0.4%. With a batch size of 2, GN ResNet-101's error is 23.0%. This is still a decently stable result considering the very small batch size, and it is 8.9% better than the BN counterpart's 31.9%.

Results and analysis of VGG models. To study GN/BN compared to *no normalization*, we consider VGG-16 [57] that can be healthily trained without normalization layers. We apply BN or GN right after each convolutional layer. Figure 6 shows the evolution of the feature distributions of conv_{5_3} (the last convolutional layer). GN and BN behave *qualitatively similar*, while being substantially different with the variant that uses no normalization; this phenomenon is also observed for all other convolutional layers. This comparison suggests that performing normalization is essential for controlling the distribution of features.

For VGG-16, GN is *better* than BN by 0.4% (Figure 6, right). This possibly implies that VGG-16 benefits less from BN's regularization effect, and GN (that leads to lower training error) is superior to BN in this case.

4.2 Object Detection and Segmentation in COCO

Next we evaluate fine-tuning the models for transferring to object detection and segmentation. These computer vision tasks in general benefit from higher-resolution input, so the batch size tends to be small in common practice (1

backbone	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
C4, BN^*	37.7	57.9	40.9	32.8	54.3	34.7
C4, GN	**38.8**	**59.2**	**42.2**	**33.6**	**55.9**	**35.4**

Table 4. Detection and segmentation results in COCO, using Mask R-CNN with the **ResNet-50 C4** backbone. BN^* means BN is frozen.

or 2 images/GPU [8,9,10,58]). As a result, BN is turned into a *linear* layer $y = \frac{\gamma}{\sigma}(x-\mu)+\beta$ where μ and σ are pre-computed from the pre-trained model and frozen [3]. We denote this as BN^*, which in fact performs no normalization during fine-tuning. We have also tried a variant that fine-tunes BN (normalization is performed and not frozen) and found it works poorly (reducing ~6 AP with a batch size of 2), so we ignore this variant.

We experiment on the Mask R-CNN baselines [10], implemented in the publicly available codebase of *Detectron* [59]. We use the end-to-end variant with the same hyper-parameters as in [59]. We replace BN^* with GN during fine-tuning, using the corresponding models pre-trained from ImageNet.[2] During fine-tuning, we use a weight decay of 0 for the γ and β parameters, which is important for good detection results when γ and β are being tuned. We fine-tune with a batch size of 1 image/GPU and 8 GPUs.

The models are trained in the COCO `train2017` set and evaluated in the COCO `val2017` set (a.k.a `minival`). 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^{mask}).

Results of C4 backbone. Table 4 shows the comparison of GN *vs*. BN^* on Mask R-CNN using a $conv_4$ backbone ("C4" [10]). This C4 variant uses ResNet's layers of up to $conv_4$ to extract feature maps, and ResNet's $conv_5$ layers as the Region-of-Interest (RoI) heads for classification and regression. As they are inherited from the pre-trained model, the backbone and head both involve normalization layers.

On this baseline, GN improves over BN^* by 1.1 box AP and 0.8 mask AP. We note that the pre-trained GN model is slightly worse than BN in ImageNet (24.1% *vs*. 23.6%), but GN still outperforms BN^* for fine-tuning. BN^* creates inconsistency between pre-training and fine-tuning (frozen), which may explain the degradation.

Results of FPN backbone. Next we compare GN and BN^* on Mask R-CNN using a Feature Pyramid Network (FPN) backbone [60], the currently state-of-the-art framework in COCO. Unlike the C4 variant, FPN exploits all pre-trained

[2] Detectron [59] uses pre-trained models provided by the authors of [3]. For fair comparisons, we instead use the models pre-trained in this paper. The object detection and segmentation accuracy is statistically similar between these pre-trained models.

backbone	box head w/	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
FPN, BN^*	-	38.6	59.5	41.9	34.2	56.2	36.1
FPN, BN^*	GN	39.5	60.0	43.2	34.4	56.4	**36.3**
FPN, GN	GN	**40.0**	**61.0**	**43.3**	**34.8**	**57.3**	**36.3**

Table 5. Detection and segmentation results in COCO, using Mask R-CNN with **ResNet-50 FPN** and a 4conv1fc bounding box head. BN^* means BN is frozen.

backbone	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
R50 BN^*	38.6	59.8	42.1	34.5	56.4	36.3
R50 GN	40.3	61.0	44.0	35.7	57.9	37.7
R50 GN, long	**40.8**	**61.6**	**44.4**	**36.1**	**58.5**	**38.2**
R101 BN^*	40.9	61.9	44.8	36.4	58.5	38.7
R101 GN	41.8	62.5	45.4	36.8	59.2	39.0
R101 GN, long	**42.3**	**62.8**	**46.2**	**37.2**	**59.7**	**39.5**

Table 6. Detection and segmentation results in COCO using Mask R-CNN and FPN. Here BN^* is the default Detectron baseline [59], and GN is applied to the backbone, box head, and mask head. "long" means training with more iterations.

layers to construct a pyramid, and appends randomly initialized layers as the head. In [60], the box head consists of two hidden fully-connected layers (2fc). We find that replacing the 2fc box head with 4conv1fc (similar to [61]) can better leverage GN. The resulting comparisons are in Table 5.

As a baseline, BN^* has 38.6 box AP using the 4conv1fc head, on par with its 2fc counterpart using the same pre-trained model (38.5 AP). By adding GN to all convolutional layers of the box head (but still using the BN^* backbone), we increase the box AP by 0.9 to 39.5 (2nd row, Table 5). This ablation shows that a substantial portion of GN's improvement for detection is from *normalization in the head* (which is also done by the C4 variant). On the contrary, applying BN to the box head (that has 512 RoIs per image) does not provide satisfactory result and is ~9 AP worse — in detection, the batch of RoIs are sampled from the same image and their distribution is not *i.i.d.*, and the *non-i.i.d.* distribution is also an issue that degrades BN's batch statistics estimation [35]. GN does not suffer from this problem.

Next we replace the FPN backbone with the GN-based counterpart, *i.e.*, the GN pre-trained model is used during fine-tuning (3rd row, Table 5). Applying GN to the backbone *alone* contributes a 0.5 AP gain (from 39.5 to 40.0), suggesting that GN helps when transferring features.

Table 6 shows the full results of GN (applied to the backbone, box head, and mask head), compared with the Detectron baseline [59] based on BN^*. Using the same hyper-parameters as [59], GN increases over BN^* by a healthy margin. Moreover, we found that GN is not fully trained with the default schedule in [59], so we also tried increasing the iterations from 180k to 270k (BN^* does not benefit from longer training). Our final ResNet-50 GN model ("long", Table 6) is **2.2** points box AP and **1.6** points mask AP better than its BN^* variant.

from scratch	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
R50 GN	39.5	59.8	43.6	35.2	56.9	37.6
R101 GN	41.0	61.1	44.9	36.4	58.2	38.7

Table 7. COCO models trained **from scratch** using Mask R-CNN and FPN.

Training Mask R-CNN from scratch. GN allows us to easily investigate training object detectors *from scratch* (without any pre-training). We show the results in Table 7, where the GN models are trained for 270k iterations. To our knowledge, our numbers (**41.0** box AP and **36.4** mask AP) are the best *from-scratch* results in COCO reported to date; they can even compete with the ImageNet-pretrained results in Table 6. As a reference, with synchronous BN [36], a concurrent work [62] achieves a from-scratch result of 34.5 box AP using R50 and 36.3 using a specialized backbone.

4.3 Video Classification in Kinetics

Lastly we evaluate video classification in the Kinetics dataset [21]. This task is memory-demanding and imposes constraints on the batch sizes.

We experiment with Inflated 3D (I3D) convolutional networks [13]. We use the ResNet-50 I3D *baseline* as described in [63]. The models are pre-trained from ImageNet. For both BN and GN, we extend the normalization from over (H, W) to over (T, H, W), where T is the temporal axis. We train in the 400-class Kinetics training set and evaluate in the validation set. We report the top-1 and top-5 classification accuracy, using standard 10-clip testing that averages softmax scores from 10 clips regularly sampled.

We study two different temporal lengths: 32-frame and 64-frame input clips. The 32-frame clip is regularly sampled with a frame interval of 2 from the raw video, and the 64-frame clip is sampled continuously. The model is fully convolutional in spacetime, so the 64-frame variant consumes about 2× more memory. We study a batch size of 8 or 4 clips/GPU for the 32-frame variant, and 4 clips/GPU for the 64-frame variant due to memory limitation.

Results of 32-frame inputs. Table 8 (col. 1, 2) shows the video classification accuracy in Kinetics using 32-frame clips. For the batch size of 8, GN is slightly worse than BN by 0.3% top-1 accuracy and 0.1% top-5. This shows that GN is competitive with BN when BN works well. For the smaller batch size of 4, GN's accuracy is kept similar (72.8 / 90.6 *vs*. 73.0 / 90.6), but is better than BN's 72.1 / 90.0. BN's accuracy is decreased by 1.2% when the batch size decreases from 8 to 4. Figure 7 shows the error curves. BN's error curves (left) have a noticeable gap when the batch size decreases from 8 to 4, while GN's error curves (right) are very similar.

Results of 64-frame inputs. Table 8 (col. 3) shows the results of using 64-frame clips. In this case, BN has a result of 73.3 / 90.8. These appear to be

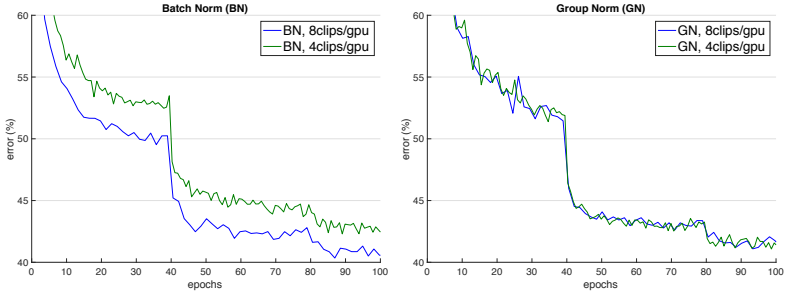


Figure 7. Error curves in Kinetics with an input length of 32 frames. We show ResNet-50 I3D's validation error of BN (left) and GN (right) using a batch size of 8 and 4 clips/GPU. The monitored validation error is the 1-clip error under the same data augmentation as the training set, while the final validation accuracy in Table 8 is 10-clip testing without data augmentation.

clip length	32	32	64
batch size	8	4	4
BN	**73.3** / **90.7**	72.1 / 90.0	73.3 / 90.8
GN	73.0 / 90.6	**72.8** / **90.6**	**74.5** / **91.7**

Table 8. Video classification in Kinetics: ResNet-50 I3D's top-1/5 accuracy (%).

acceptable numbers (*vs*. 73.3 / 90.7 of 32-frame, batch size 8), but *the trade-off between the temporal length (64* vs. *32) and batch size (4* vs. *8) could have been overlooked*. Comparing col. 3 and col. 2 in Table 8, we find that the temporal length actually has positive impact (+1.2%), but it is veiled by BN's negative effect of the smaller batch size.

GN does not suffer from this trade-off. The 64-frame variant of GN has 74.5 / 91.7 accuracy, showing healthy gains over its BN counterpart and all BN variants. GN helps the model benefit from temporal length, and the longer clip boosts the top-1 accuracy by 1.7% (top-5 1.1%) with the same batch size.

The improvement of GN on detection, segmentation, and video classification demonstrates that GN is a strong alternative to the powerful and currently dominant BN technique in these tasks.

5 Discussion and Future Work

We have presented GN as an effective normalization layer without exploiting the batch dimension. We have evaluated GN's behaviors in a variety of applications. We note, however, that BN has been so influential that many state-of-the-art systems and their hyper-parameters have been designed for it, which may not be optimal for GN-based models. It is possible that re-designing the systems or searching new hyper-parameters for GN will give better results.

References

1. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. (2015)
2. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: CVPR. (2016)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
4. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of go without human knowledge. Nature (2017)
5. Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: ICLR Workshop. (2016)
6. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR. (2017)
7. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: CVPR. (2017)
8. Girshick, R.: Fast R-CNN. In: ICCV. (2015)
9. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)
10. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. In: ICCV. (2017)
11. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015)
12. Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M.: Learning spatiotemporal features with 3D convolutional networks. In: ICCV. (2015)
13. Carreira, J., Zisserman, A.: Quo vadis, action recognition? a new model and the kinetics dataset. In: CVPR. (2017)
14. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV (2004)
15. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR. (2005)
16. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. IJCV (2015)
17. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv:1607.06450 (2016)
18. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: The missing ingredient for fast stylization. arXiv:1607.08022 (2016)
19. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: NIPS. (2016)
20. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: ECCV. (2014)
21. Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., Viola, F., Green, T., Back, T., Natsev, P., et al.: The Kinetics human action video dataset. arXiv:1705.06950 (2017)
22. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature (1986)
23. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation (1997)
24. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS. (2014)

25. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: CVPR. (2017)
26. Lyu, S., Simoncelli, E.P.: Nonlinear image representation using divisive normalization. In: CVPR. (2008)
27. Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al.: What is the best multi-stage architecture for object recognition? In: ICCV. (2009)
28. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012)
29. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional neural networks. In: ECCV. (2014)
30. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. In: ICLR. (2014)
31. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: CVPR. (2015)
32. Rebuffi, S.A., Bilen, H., Vedaldi, A.: Learning multiple visual domains with residual adapters. In: NIPS. (2017)
33. Arpit, D., Zhou, Y., Kota, B., Govindaraju, V.: Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In: ICML. (2016)
34. Ren, M., Liao, R., Urtasun, R., Sinz, F.H., Zemel, R.S.: Normalizing the normalizers: Comparing and extending network normalization schemes. In: ICLR. (2017)
35. Ioffe, S.: Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In: NIPS. (2017)
36. Peng, C., Xiao, T., Li, Z., Jiang, Y., Zhang, X., Jia, K., Yu, G., Sun, J.: MegDet: A large mini-batch object detector. In: CVPR. (2018)
37. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: NIPS. (2012)
38. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 (2017)
39. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: CVPR. (2017)
40. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: CVPR. (2018)
41. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. IJCV (2001)
42. Jegou, H., Douze, M., Schmid, C., Perez, P.: Aggregating local descriptors into a compact image representation. In: CVPR. (2010)
43. Perronnin, F., Dance, C.: Fisher kernels on visual vocabularies for image categorization. In: CVPR. (2007)
44. Dieleman, S., De Fauw, J., Kavukcuoglu, K.: Exploiting cyclic symmetry in convolutional neural networks. In: ICML. (2016)
45. Cohen, T., Welling, M.: Group equivariant convolutional networks. In: ICML. (2016)
46. Heeger, D.J.: Normalization of cell responses in cat striate cortex. Visual neuroscience (1992)
47. Schwartz, O., Simoncelli, E.P.: Natural signal statistics and sensory gain control. Nature neuroscience (2001)

48. Simoncelli, E.P., Olshausen, B.A.: Natural image statistics and neural representation. Annual review of neuroscience (2001)
49. Carandini, M., Heeger, D.J.: Normalization as a canonical neural computation. Nature Reviews Neuroscience (2012)
50. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. (2017)
51. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: Operating Systems Design and Implementation (OSDI). (2016)
52. Gross, S., Wilber, M.: Training and investigating Residual Nets. https://github.com/facebook/fb.resnet.torch (2016)
53. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV. (2015)
54. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv:1706.02677 (2017)
55. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997 (2014)
56. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. arXiv:1606.04838 (2016)
57. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR. (2015)
58. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: ICCV. (2017)
59. Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., He, K.: Detectron. https://github.com/facebookresearch/detectron (2018)
60. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: CVPR. (2017)
61. Ren, S., He, K., Girshick, R., Zhang, X., Sun, J.: Object detection networks on convolutional feature maps. TPAMI (2017)
62. Li, Z., Peng, C., Yu, G., Zhang, X., Deng, Y., Sun, J.: DetNet: A backbone network for object detection. arXiv:1804.06215 (2018)
63. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: CVPR. (2018)