International Journal of Computer Vision (2020) 128:742–755
https://doi.org/10.1007/s11263-019-01198-w

Group Normalization

Yuxin Wu[1] · **Kaiming He**[1]

Received: 1 February 2019 / Accepted: 8 July 2019 / Published online: 22 July 2019

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 (https://github.com/facebookresearch/Detectron/blob/master/projects/GN), 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 in modern libraries.

Keywords Normalization · Image recognition · Object detection · Batch size

1 Introduction

Batch Normalization (Batch Norm or BN; Ioffe and Szegedy 2015) has been established as a very effective component in deep learning, largely helping push the frontier in computer vision (Szegedy et al. 2016b; He et al. 2016) and beyond (Silver et al. 2017). 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 computer vision 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] Ioffe and Szegedy 2015; Szegedy et al. 2016b; He et al. 2016). A small batch leads to inaccurate estimation of the batch statistics, and *reducing BN's batch size increases the model error dramatically* (Fig. 1). As a result, many recent models (Szegedy et al. 2016b; He et al. 2016; Szegedy et al. 2016a; Huang et al. 2017; Xie et al. 2017) 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 (Girshick 2015; Ren et al. 2015; He et al. 2017), segmentation (Long et al. 2015; He et al. 2017), video recognition (Tran et al. 2015; Carreira and Zisserman 2017), and other high-level systems built on them. For example, the Fast/er and Mask R-CNN frame-

Communicated by M. Hebert.

✉ Yuxin Wu
yuxinwu@fb.com

Kaiming He
kaiminghe@fb.com

[1] Facebook AI Research, Menlo Park, USA

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

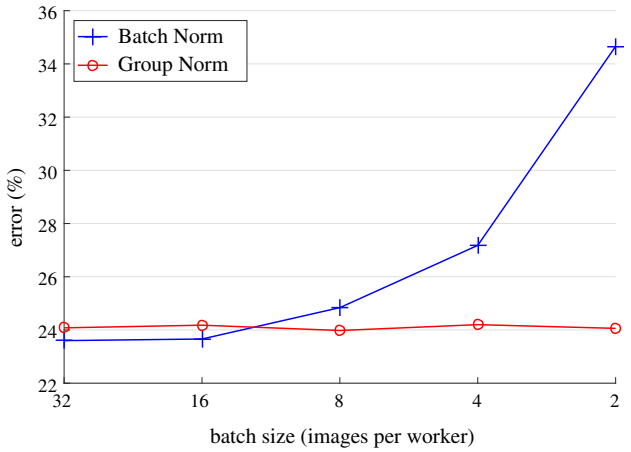


Fig. 1 ImageNet classification error versus batch sizes. This is a ResNet-50 model trained in the ImageNet training set using 8 workers (GPUs), evaluated in the validation set

works (Girshick 2015; Ren et al. 2015; He et al. 2017) use a batch size of 1 or 2 images because of higher resolution, where BN is "frozen" by transforming to a linear layer (He et al. 2016); in video classification with 3D convolutions (Tran et al. 2015; Carreira and Zisserman 2017), 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 (Lowe 2004) and HOG (Dalal and Triggs 2005) 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 (Fig. 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 (Fig. 1). With a batch size of 2 samples, GN has 10.6% lower error than its BN counterpart for ResNet-50 (He et al. 2016) in ImageNet (Russakovsky et al. 2015). With a regular batch size, GN is comparably good as BN (with a gap of ~0.5%) and outperforms other normalization variants (Ba et al. 2016; Ulyanov et al. 2016; Salimans and Kingma 2016). Moreover, although the batch size may change, GN can naturally transfer from pre-training to fine-tuning. GN shows improved results versus its BN counterpart on Mask R-CNN for COCO object detection and segmentation (Lin et al. 2014), and on 3D convolutional networks for Kinetics video classification (Kay et al. 2017). 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) (Ba et al. 2016) and Instance Normalization (IN) (Ulyanov et al. 2016) (Fig. 2), that also avoid normalizing along the batch dimension. These methods are effective for training sequential models (RNN/LSTM Rumelhart et al. 1986; Hochreiter and Schmidhuber 1997) or generative models (GANs; Goodfellow et al. 2014; Isola et al. 2017). 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.

A preliminary version of this manuscript has been published in ECCV (Wu and He 2018). After that, GN has created new research opportunities that would be blocked by the limitations of BN. For example, He et al. (2018) demonstrate that with the help of GN one can train object detectors from scratch without sacrificing accuracy, questioning the common wisdom on the role of ImageNet pre-training. GN also facilitates training joint speech and video networks (Shillingford et al. 2018) that are challenged by small batch sizes and variable lengths. We believe that the introduction of GN will provide more room for researchers to explore the uncharted areas. This manuscript also provides additional results of large-batch distributed training, showing one limitation of GN.

2 Related Work

Normalization It is well-known that normalizing the input data makes training faster (LeCun et al. 1998). To normalize hidden features, initialization methods (LeCun et al. 1998; Glorot and Bengio 2010; He et al. 2015) have been derived based on strong assumptions of feature distributions, which can become invalid when training evolves.

Normalization layers in deep networks had been widely used before the development of BN. Local Response Normalization (LRN) (Lyu and Simoncelli 2008; Jarrett et al. 2009; Krizhevsky et al. 2012) was a component in AlexNet (Krizhevsky et al. 2012) and following models (Zeiler and Fergus 2014; Sermanet et al. 2014; Szegedy et al. 2015). Unlike recent methods (Ioffe and Szegedy 2015; Ba et al. 2016; Ulyanov et al. 2016), LRN computes the statistics in a small neighborhood for each pixel.

Batch Normalization (Ioffe and Szegedy 2015) 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

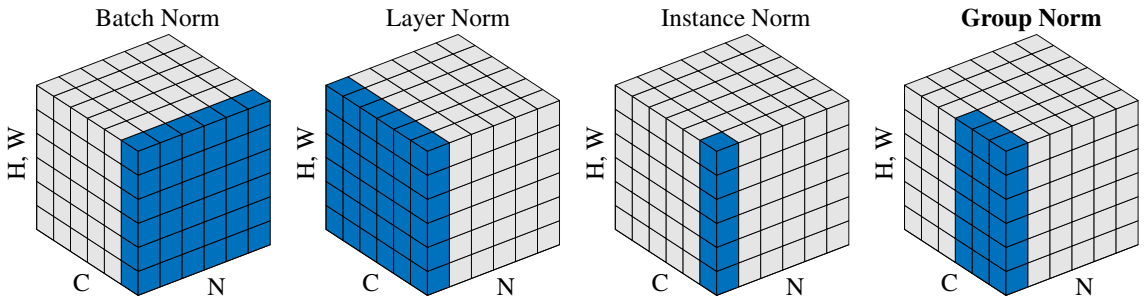


Fig. 2 Normalization methods. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels

not legitimate at inference time, so the mean and variance are pre-computed from the training set (Ioffe and Szegedy 2015), 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 (Rebuffi et al. 2017). 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 (Ba et al. 2016; Ulyanov et al. 2016; Salimans and Kingma 2016; Arpit et al. 2016; Ren et al. 2017a) have been proposed to avoid exploiting the batch dimension. Layer Normalization (LN) (Ba et al. 2016) operates along the channel dimension, and Instance Normalization (IN) (Ulyanov et al. 2016) performs BN-like computation but only for each sample (Fig. 2). Instead of operating on features, Weight Normalization (WN) (Salimans and Kingma 2016) 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 provide comparisons with these methods in context of the remaining sections.

Addressing Small Batches Ioffe (2017) 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 (Ioffe 2017).

There are also attempts to *avoid* using small batches. The object detector in Peng et al. (2018) 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; Dean et al. 2012), 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., Ioffe 2017; Peng et al. 2018), our normalization method inherently avoids this computation.

Group-Wise Computation *Group convolutions* have been presented by AlexNet (Krizhevsky et al. 2012) 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 (Xie et al. 2017) 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 (Howard et al. 2017) and Xception (Chollet 2017) exploit *channel-wise* (also called "depth-wise") convolutions, which are group convolutions with a group number equal to the channel number. ShuffleNet (Zhang et al. 2018) 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 (He et al. 2016).

3 Group Normalization

The channels of visual representations are not entirely independent. Classical features of SIFT (Lowe 2004), HOG (Dalal and Triggs 2005), and GIST (Oliva and Torralba 2001) 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 (Jegou et al. 2010) and Fisher Vectors (FV) (Perronnin and Dance 2007) 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 (Dieleman et al. 2016; Cohen and Welling 2016), 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 Lowe 2004, HOG Dalal and Triggs 2005, or Dieleman et al. 2016; Cohen and Welling 2016), 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 (Heeger et al. 1992; Schwartz and Simoncelli 2001; Simoncelli and Olshausen 2001; Carandini and Heeger 2012), "with various receptive-field centers (covering the visual field) and with various spatiotemporal frequency tunings" (p183, Heeger et al. 1992); this can happen not only in the primary visual cortex, but also "throughout the visual system" (Carandini and Heeger 2012). Motivated by these works, we propose new generic group-wise normalization for deep neural networks.

3.1 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 (Fig. 2), discussed as follows.

In Batch Norm (Ioffe and Szegedy 2015), 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 (Ba et al. 2016), 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 (Ulyanov et al. 2016), 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 Fig. 2.

As in Ioffe and Szegedy (2015), 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).

Group Norm Formally, a Group Norm layer computes μ and σ in a set $\mathcal{S}_i$ 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 Fig. 2 (rightmost), which is a simple case of 2 groups ($G = 2$) each having 3 channels.

Given $\mathcal{S}_i$ in Eq. (7), a GN layer is defined by Eqs. (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.

```
def GroupNorm(x, gamma, beta, G, eps=1e−5):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: 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
```

Fig. 3 Python code of Group Norm based on TensorFlow. Here the function `tf.nn.moments` computes the mean and variance by aggregating along the specified axes

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

    N, C, H, W = x.shape
    x = x.view(N ∗ G, −1) # reshape to [N∗G, C//G∗H∗W]

    mean = x.mean(1, keepdim=True)
    var = x.var(1, keepdim=True)
    x = (x − mean) / (var + eps).sqrt()

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

    return gamma ∗ x + beta
```

Fig. 4 Python code of Group Norm based on PyTorch. Here the function `x.mean` and `x.std` computes the mean and std by aggregating along the specified axes

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 (Fig. 2).

Relation to Layer Normalization (Ba et al. 2016). GN becomes LN if we set the group number as $G = 1$. LN assumes *all* channels in a layer make "similar contributions" (Ba et al. 2016). Unlike the case of fully-connected layers studied in Ba et al. (2016), this assumption can be less valid with the presence of convolutions, as discussed in Ba et al. (2016). 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 (Fig. 5).

Relation to Instance Normalization (Ulyanov et al. 2016). GN becomes IN 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.

3.2 Implementation

GN can be easily implemented by a few lines of code in PyTorch and TensorFlow (Abadi et al. 2016) where automatic differentiation is supported. Figures 3 and 4 show the code based on TensorFlow and PyTorch. 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.

The implementation in Figs. 3 and 4 is convenient for prototyping. Considering the frequent usage of normalization layers, we recommend to implement GN as a stand-alone backend operation (op) written in C and CUDA, similar to the common practice of BN. This can reduce memory usage and increase running speed. We have made our implementation available online, for both Caffe2[2] and PyTorch.[3]

4 Experiments

4.1 Image Classification in ImageNet

We experiment in the ImageNet classification dataset (Russakovsky et al. 2015) with 1000 classes. We train on the ∼1.28 M training images and evaluate on the 50,000 validation images, using the ResNet models (He et al. 2016).

Implementation Details As standard practice (He et al. 2016; Gross and Wilber 2016), 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 He et al. (2015) 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 Goyal et al. (2017) (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 Gross and Wilber 2016 but unlike He et al. 2016; Goyal et al. 2017). We train 100 epochs for all models, and decrease the learning rate by $10\times$ at 30, 60, and 90 epochs. During training, we adopt the data augmentation of Szegedy et al. (2015) as implemented by Gross and Wilber (2016). 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 (Goyal et al. 2017). Other implementation details follow Gross and Wilber (2016).

[2] https://github.com/pytorch/pytorch/blob/master/caffe2/operators/group_norm_op.h.

[3] https://github.com/pytorch/pytorch/blob/master/aten/src/ATen/native/Normalization.cpp.

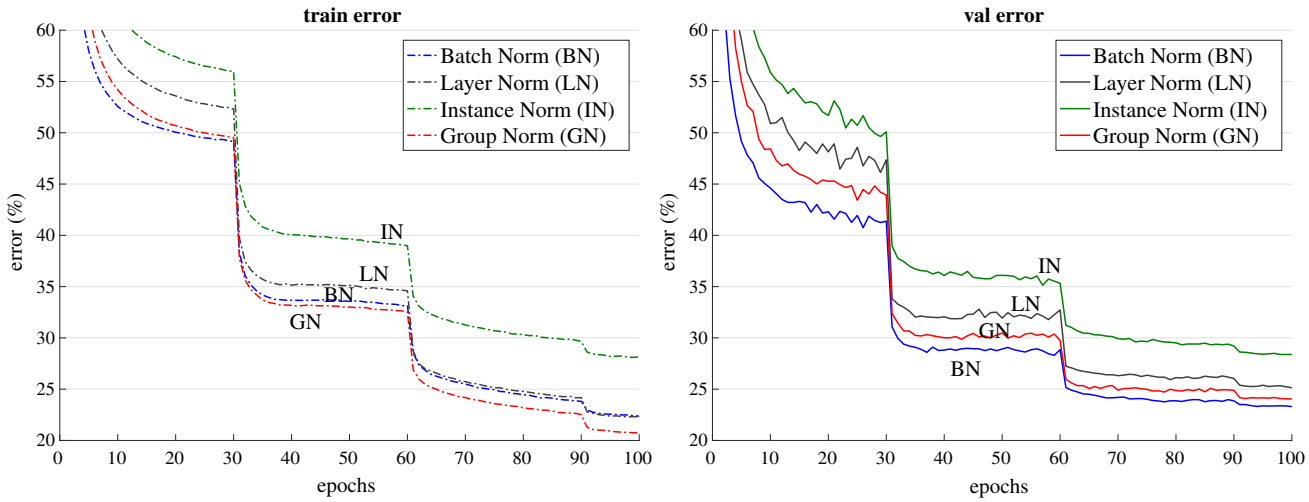


Fig. 5 Comparison of error curves with a batch size of 32 images/GPU. We show the ImageNet training error (left) and validation error (right) versus numbers of training epochs. The model is ResNet-50

Table 1 Comparison of error rates (%) of ResNet-50 in the ImageNet validation set, trained with a batch size of 32 images/GPU. The error curves are in Fig. 5

	BN	LN	IN	GN
Val error	**23.6**	25.3	28.4	24.1
△ (versus BN)	–	*1.7*	*4.8*	***0.5***

Bold values indicate best results in each row

Our baseline is the ResNet trained with BN (He et al. 2016). 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.

Comparison of Feature Normalization Methods We first experiment with a regular batch size of 32 images (per GPU) (Ioffe and Szegedy 2015; He et al. 2016). BN works successfully in this regime, so this is a strong baseline to compare with. Figure 5 shows the error curves, and Table 1 shows the final results.

Figure 5 shows that *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.[4]

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, Fig. 5 (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 (Ioffe and Szegedy 2015). 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 Figs. 1, 6, 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. All models are trained in 8 GPUs. In this set of experiments, we adopt the linear learning rate scaling rule (Krizhevsky et al. 2014; Bottou et al. 2016; Goyal et al. 2017) to adapt to batch size changes—we use a learning rate of 0.1 (He et al. 2016) 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 (Goyal et al. 2017). We keep the same number of training epochs for all cases (Fig. 6, x-axis). All other hyper-parameters are unchanged.

Figure 6 (left) shows that BN's error becomes considerably higher with small batch sizes. GN's behavior is more stable and insensitive to the small batch size. Actually, Fig. 6 (right) shows that GN has very similar curves (subject to random variations) across a wide range of batch sizes from 32 to 2. In the case of a batch size of 2, GN has **10.6**% lower error rate than its BN counterpart (24.1 vs 34.7%). We've also

[4] For completeness, we have also trained ResNet-50 with WN (Salimans and Kingma 2016), which is *filter* (instead of *feature*) normalization. WN's result is 28.2%.

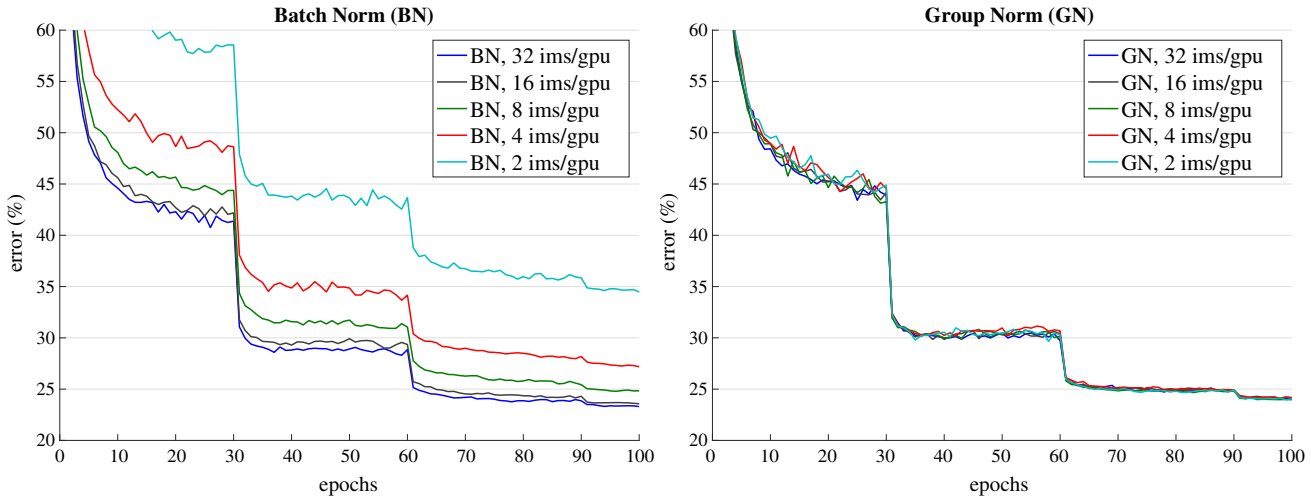


Fig. 6 Sensitivity to small batch sizes: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU

Table 2 Sensitivity to small batch sizes

Batch size per GPU	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*

We show ResNet-50's validation error (%) in ImageNet. The last row shows the differences between BN and GN. The error curves are in Fig. 6. This table is visualized in Fig. 1
Bold values indicate best results for each batch size

Table 3 Group division

# 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*

We show ResNet-50's validation error (%) in ImageNet, trained with 32 images/GPU. (Top): a given number of groups. (Bottom): a given number of channels per group. The last rows show the differences with the best
Bold values indicate best results

observed that LN and IN have similar robustness to change of batch size, since they are both batch-independent as well.

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%).

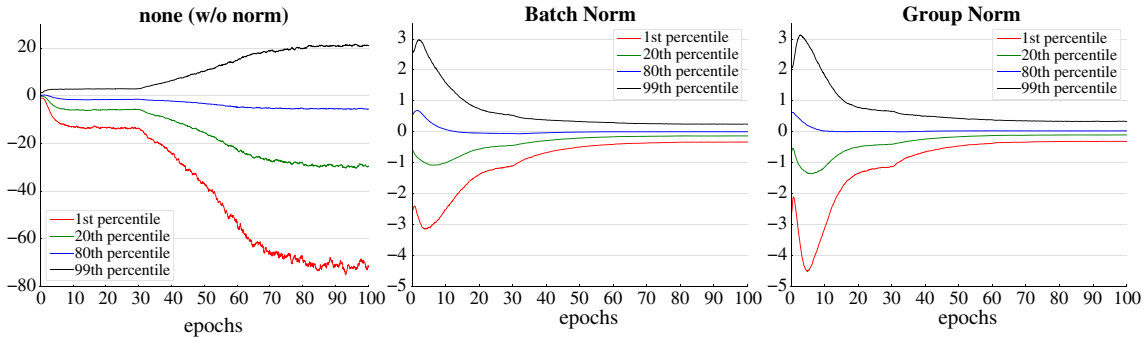
The robust results of GN in Table 2 demonstrate GN's strength. It allows to remove the batch size constraint imposed by BN, which can give considerably more memory (e.g., 16× or more). This will make it possible to train higher-capacity models that would be otherwise bottlenecked by memory limitation.

Comparison with Batch Renorm (BR) BR (Ioffe 2017) introduces two extra parameters [r and d in Ioffe (2017)] 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 studied (Table 3, top 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, bottom 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 (He et al. 2016). With a batch size of 32, our BN



	error
none	29.2
BN	28.0
GN	**27.6**

Fig. 7 Evolution of feature distributions of conv_{5_3}'s output (before normalization and ReLU) from VGG-16, shown as the $\{1, 20, 80, 99\}$ percentile of responses. The table on the right shows the ImageNet validation error (%). Models are trained with 32 images/GPU

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 (Simonyan and Zisserman 2015) that can be healthily trained without normalization layers. We apply BN or GN right after each convolutional layer. Figure 7 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% (Fig. 7, 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 or 2 images/GPU Girshick 2015; Ren et al. 2015; He et al. 2017; Lin et al. 2017b). 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 (He et al. 2016). 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 (He et al. 2017), implemented in the publicly available codebase of *Detectron* (Girshick et al. 2018). We use the end-to-end variant with the same hyper-parameters as in Girshick et al. (2018). We replace BN* with GN during fine-tuning, using the corresponding models pre-trained from ImageNet.[5] 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 versus BN* on Mask R-CNN using a conv_4 backbone ("C4" He et al. 2017). 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.

We have also experimented with the LN variant, and found it is 1.9 box AP worse than GN and 0.8 worse than BN*. Although LN is also independent of batch sizes, its representational power is weaker than GN.

[5] Detectron Girshick et al. (2018) uses pre-trained models provided by the authors of He et al. (2016). 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.

Table 4 Detection and segmentation ablation results in COCO, using Mask R-CNN with ResNet-50 C4

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

Bold values indicate best results
BN* means BN is frozen

Table 5 Detection and segmentation ablation results in COCO, using Mask R-CNN with ResNet-50 FPN and a 4conv1fc bounding box head

Backbone	Box head	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
BN*	–	38.6	59.5	41.9	34.2	56.2	36.1
BN*	GN	39.5	60.0	43.2	34.4	56.4	**36.3**
GN	GN	**40.0**	**61.0**	**43.3**	**34.8**	**57.3**	**36.3**

Bold values indicate best results
BN* means BN is frozen

Table 6 Detection and segmentation results in COCO using Mask R-CNN and FPN

	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, longer	**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, longer	**42.3**	**62.8**	**46.2**	**37.2**	**59.7**	**39.5**

Here BN* is the default Detectron baseline (Girshick et al. 2018), and GN is applied to the backbone, box head, and mask head. "longer" means training with more iterations. Code of these results are in https://github.com/facebookresearch/Detectron/blob/master/projects/GN
Bold values indicate best results

Results of FPN Backbone Next we compare GN and BN* on Mask R-CNN using a Feature Pyramid Network (FPN) backbone (Lin et al. 2017a), the currently state-of-the-art framework in COCO. Unlike the C4 variant, FPN exploits all pre-trained layers to construct a pyramid, and appends randomly initialized layers as the head. In Lin et al. (2017a), the box head consists of two hidden fully-connected layers (2fc). We find that replacing the 2fc box head with 4conv1fc [similar to Ren et al. (2017b)] 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 0.9–39.5 (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 (Ioffe 2017). 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 standard Detectron baseline (Girshick et al. 2018) based on BN*. Using the same hyper-parameters as Girshick et al. (2018), GN increases over BN* by a healthy margin. Moreover, we found that GN is not fully trained with the default schedule in Girshick et al. (2018), 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.

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 270 k iterations.[6] At the time of our preliminary publication Wu and He (2018), to our

[6] For models trained from scratch, we turn off the default StopGrad in Detectron that freezes the first few layers.

Table 7 Detection and segmentation results trained from scratch in COCO using Mask R-CNN and FPN

From scratch	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
R50 BN (Li et al. 2018)	34.5	55.2	37.7	–	–	–
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

Here the BN results are from Li et al. (2018), and BN is synced across GPUs (Peng et al. 2018) and is *not* frozen. Code of these results are in https://github.com/facebookresearch/Detectron/blob/master/projects/GN

knowledge, these 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 (Peng et al. 2018), a concurrent work (Li et al. 2018) achieves a from-scratch result of 34.5 box AP using R50 (Table 7), and 36.3 using a specialized backbone.

In fact, the results in Table 7 provide encouraging signals, suggesting that there might be no fundamental problem preventing training object detectors from scratch, if an appropriate normalization is adopted. He et al. He et al. (2018) find that the models in Table 7 can *match the accuracy* of ImageNet-pretrained counterparts if trained sufficiently long till convergence. The new discovery, enabled by our introduction of GN, demonstrates the scientific values of GN as an alternative of the formerly dominant BN in computer vision.

4.3 Video Classification in Kinetics

Lastly we evaluate video classification in the Kinetics dataset (Kay et al. 2017). Many video classification models (Tran et al. 2015; Carreira and Zisserman 2017) extend the features to 3D spatial-temporal dimensions. This is memory-demanding and imposes constraints on the batch sizes and model designs.

We experiment with Inflated 3D (I3D) convolutional networks (Carreira and Zisserman 2017). We use the ResNet-50 I3D *baseline* as described in Wang et al. (2018). 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.

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

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**

Bold values indicate best results for each setting

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 8 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 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.

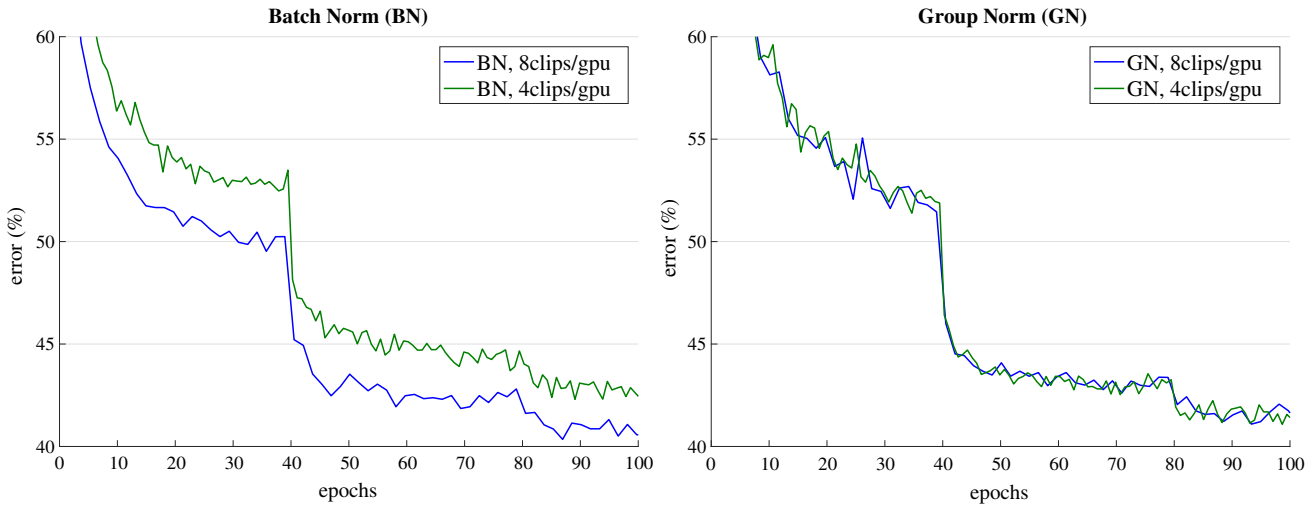


Fig. 8 Error curves in Kinetics with an input length of 32 frames. We show ResNet-50 I3D's validation error of BN (left) and GN (right) using a batch size of 8 and 4 clips/GPU. The monitored validation error is the 1-clip error under the same data augmentation as the training set, while the final validation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

5 Limitation and Discussion

GN, as an alternative to BN, also enables us to expand the research horizon on those topics involving batch sizes. By comparing the behaviors between BN and GN, we may have new evidence on the underlying factors that are hard to identify if BN is the only usable tool. Next we present our results on *large-batch distributed training*—it shows one limitation of GN, which however provides more hints about distributed training and BN.

Thus far we have focused on the *per-GPU batch size* that impacts batch statistics computation in the presence of BN. In this section we investigate another scenario called *distributed training*[7]—the *total batch size* varies, while the per-GPU batch size is kept fixed.

Experiment Setup We follow the distributed training recipe of Goyal et al. (2017). Specially, we fix the per-GPU batch size to 32, such that the BN statistics are computed in the reliable regime. The total batch size is scaled proportionally with the number of GPUs. We use the linear learning rate scaling (Goyal et al. 2017) to adapt to changes in total batch size—we use a learning rate of 0.1 for 8 GPUs (baseline), and $0.1K/8$ for K GPUs. We use learning rate warm-up (Goyal et al. 2017) in the first 5 epochs. Other implementation details are the same as above.

[7] We refer to "distributed training" as training with multiple workers (GPUs), which are often hosted in multiple machines. In our infrastructure, typical settings are 8 GPUs per machine.

Results Figure 9 and Table 9 shows the ImageNet validation error of BN versus GN when using a total batch size of 256, 512, 1024, and 2048. The BN-based model (Fig. 9, left) behaves elegantly when the total batch size increases, as demonstrated by the nicely matching curves across different numbers of GPUs. This shows the effectiveness of the large-batch training recipe in Goyal et al. (2017) (i.e., linear learning rate scaling with warm up). The nice property breaks when the total batch size increases, e.g., to 32768, see Fig. 9 (left); more details are in Goyal et al. (2017).

On the other hand, GN (Fig. 9, right) exhibits degradation when the total batch size is 1024 or more (i.e., 32 GPUs or more). The large-batch training recipe still works well when the total batch size is 512, which is consistent with our observation on GN in the small-batch regime (see Fig. 6, right).

Discussions Figure 9 suggests that GN is more sensitive than BN to a larger *total* batch size. Both methods show degradation when the total batch size is too large, but the breaking-down size for GN is smaller (between 512 and 1024). Interestingly, we have found LN and IN have a similar breaking-down size as GN.

This limitation of GN in turn provides new perspectives for understanding large-batch training. The presence of BN results in a unique property of batching—a batch is *hierarchical* when the batch statistics is computed within each GPU while the gradients are accumulated across all samples in all GPUs. When this happens, the gradient of one sample is *dependent* of all other samples in the same GPU, but independent of those in other GPUs.

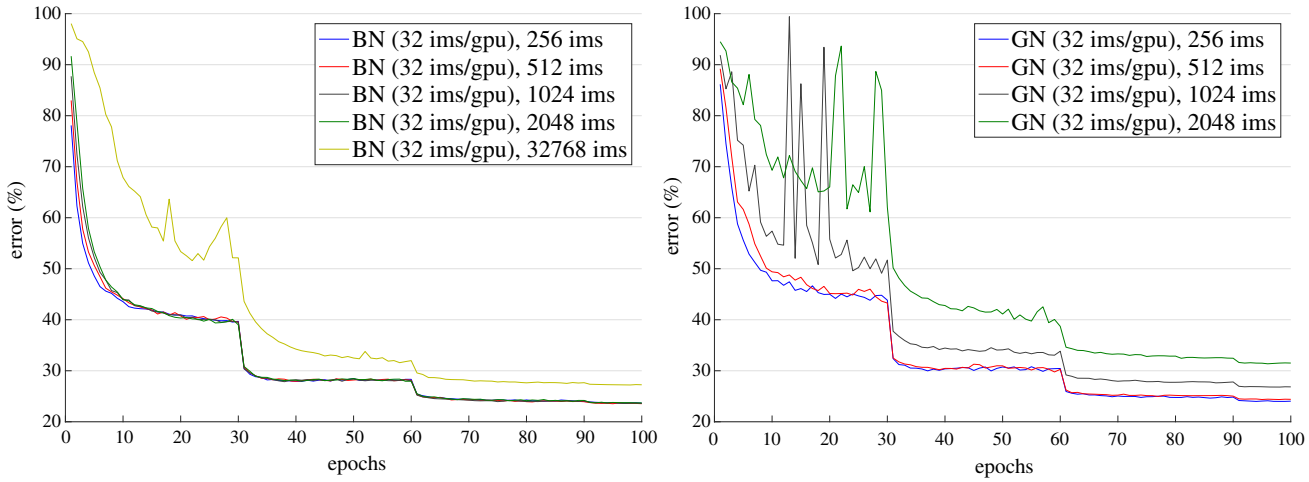


Fig. 9 Distributed training with larger total batch sizes: ResNet-50's error on ImageNet validation set of BN (left) and GN (right), trained with 32 images/GPU in 8, 16, 32, or 64 GPUs, resulting in a total batch size of 256, 512, 1024, or 2048. BN is also shown with 1024 GPUs (a total batch size of 32768), when it starts to show degradation (Goyal et al. 2017)

Table 9 Distributed training with larger total batch sizes

Total batch size	256	512	1024	2048	32768
BN	23.6	23.5	23.5	23.5	27.3
GN	24.1	24.4	26.8	31.5	–

We show ResNet-50's validation error (%) in ImageNet, corresponding to Fig. 9. The batch size per GPU is 32

The GN/LN/IN and other standard SGD counterparts do not have this property, as the gradient of one sample is always independent of all other samples regardless where they are computed. We suspect that the hierarchical batching property may be an essential factor underlying the recently prevalent large-batch distributed training (Goyal et al. 2017; Gitman and Ginsburg 2017). We hope future research will delve deeper into this topic.

Although GN shows its limitation under the large-batch distributed training scenario, introducing an alternative to BN for various scenarios is beneficial for explorative research.

6 Conclusion

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.

In addition, we have shown that GN is related to LN and IN, two normalization methods that are particularly successful in training recurrent or generative models. This suggests us to study GN in those areas in the future. We will also investigate GN's performance on learning representations for reinforcement learning (RL) tasks, e.g., (Silver et al. 2017), where BN is playing an important role for training very deep models (He et al. 2016).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *Operating systems design and implementation (OSDI)*.

Arpit, D., Zhou, Y., Kota, B., & Govindaraju, V. (2016). Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *International conference on machine learning (ICML)*.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450.

Bottou, L., Curtis, F. E., & Nocedal, J. (2016). Optimization methods for large-scale machine learning. arXiv:1606.04838.

Carandini, M., & Heeger, D. J. (2012). Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, *13*, 51.

Carreira, J., & Zisserman, A. (2017). Quo vadis, action recognition? A new model and the kinetics dataset. In *Computer vision and pattern recognition (CVPR)*.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Computer vision and pattern recognition (CVPR)*.

Cohen, T., & Welling, M. (2016). Group equivariant convolutional networks. In *International conference on machine learning (ICML)*.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer vision and pattern recognition (CVPR)*.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale dis-

tributed deep networks. In *Neural information processing systems (NeurIPS)*.
Dieleman, S., De Fauw, J., & Kavukcuoglu, K. (2016). Exploiting cyclic symmetry in convolutional neural networks. In *International conference on machine learning (ICML)*.
Girshick, R. (2015). Fast R-CNN. In *International conference on computer vision (ICCV)*.
Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., & He, K. (2018). Detectron. https://github.com/facebookresearch/detectron.
Gitman, Y. Y. I., & Ginsburg, B. (2017). Scaling SGD batch size to 32 k for imagenet training. arXiv:1708.03888.
Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics (AISTATS)*.
Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Neural information processing systems (NeurIPS)*.
Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., et al. (2017). Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv:1706.02677.
Gross, S., & Wilber, M. (2016). Training and investigating residual nets. https://github.com/facebook/fb.resnet.torch.
He, K., Girshick, R., & Dollár, P. (2018). Rethinking imagenet pre-training. arXiv:1811.08883.
He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In *International conference on computer vision (ICCV)*.
He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International conference on computer vision (ICCV)*.
He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Computer vision and pattern recognition (CVPR)*.
Heeger, D. J. (1992). Normalization of cell responses in cat striate cortex. *Visual Neuroscience*, *9*, 181–197.
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.
Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.
Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K.Q. (2017). Densely connected convolutional networks. In *Computer vision and pattern recognition (CVPR)*.
Ioffe, S. (2017). Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Neural Information processing systems (NeurIPS)*.
Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning (ICML)*.
Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Computer vision and pattern recognition (CVPR)*.
Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? In *International conference on computer vision (ICCV)*.
Jegou, H., Douze, M., Schmid, C., & Perez, P. (2010). Aggregating local descriptors into a compact image representation. In *Computer vision and pattern recognition (CVPR)*.
Kay, W., Carreira, J., Simonyan, K., Zhang, B., Hillier, C., Vijayanarasimhan, S., et al. (2017). The Kinetics human action video dataset. arXiv:1705.06950.
Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997.
Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In *Neural information processing systems (NeurIPS)*.
LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient backprop. In *Neural networks: Tricks of the trade*.
Li, Z., Peng, C., Yu, G., Zhang, X., Deng, Y., & Sun, J. (2018). DetNet: A backbone network for object detection. arXiv:1804.06215.
Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017a). Feature pyramid networks for object detection. In *Computer vision and pattern recognition (CVPR)*.
Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017b). Focal loss for dense object detection. In *International conference on computer vision (ICCV)*.
Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *European conference on computer vision (ECCV)*.
Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Computer vision and pattern recognition (CVPR)*.
Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, *60*, 91–110.
Lyu, S., & Simoncelli, E. P. (2008). Nonlinear image representation using divisive normalization. In *Computer vision and pattern recognition (CVPR)*.
Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision (IJCV)*, *42*, 145–175.
Peng, C., Xiao, T., Li, Z., Jiang, Y., Zhang, X., Jia, K., Yu, G., & Sun, J. (2018). MegDet: A large mini-batch object detector. In *Computer vision and pattern recognition (CVPR)*.
Perronnin, F., & Dance, C. (2007). Fisher kernels on visual vocabularies for image categorization. In *Computer vision and pattern recognition (CVPR)*.
Rebuffi, S. A., Bilen, H., & Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Neural information processing systems (NeurIPS)*.
Ren, M., Liao, R., Urtasun, R., Sinz, F. H., & Zemel, R. S. (2017a). Normalizing the normalizers: Comparing and extending network normalization schemes. In *International conference on learning representations (ICLR)*.
Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural information processing systems (NeurIPS)*.
Ren, S., He, K., Girshick, R., Zhang, X., & Sun, J. (2017b). Object detection networks on convolutional feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *39*, 1476–1481.
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *5*, 1.
Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV).*, *115*, 211–252.
Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Neural information processing systems (NeurIPS)*.
Schwartz, O., & Simoncelli, E. P. (2001). Natural signal statistics and sensory gain control. *Nature Neuroscience*, *4*, 819.
Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International conference on learning representations (ICLR)*.
Shillingford, B., Assael, Y., Hoffman, M. W., Paine, T., Hughes, C., Prabhu, U., et al. (2018). Large-scale visual speech recognition. arXiv:1807.05162.
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354.

Simoncelli, E. P., & Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annual Review of Neuroscience*, *24*, 1193–1216.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International conference on learning representations (ICLR)*.

Szegedy, C., Ioffe, S., & Vanhoucke, V. (2016a). Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR workshop*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *Computer vision and pattern recognition (CVPR)*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016b). Rethinking the inception architecture for computer vision. In *Computer vision and pattern recognition (CVPR)*.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning spatiotemporal features with 3D convolutional networks. In *International conference on computer vision (ICCV)*.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. arXiv:1607.08022.

Wang, X., Girshick, R., Gupta, A., & He, K. (2018). Non-local neural networks. In *Computer vision and pattern recognition (CVPR)*.

Wu, Y., & He, K. (2018). Group normalization. In *European conference on computer vision (ECCV)*.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated residual transformations for deep neural networks. In *Computer vision and pattern recognition (CVPR)*.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional neural networks. In *European conference on computer vision (ECCV)*.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Computer vision and pattern recognition (CVPR)*.