Group Normalization

Yuxin Wu Kaiming He

Facebook AI Research (FAIR)

{yuxinwu,kaiminghe}

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,[1] 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.

1. Introduction

Batch Normalization (Batch Norm or BN) [26] has been established as a very effective component in deep learning, largely helping push the frontier in computer vision [59, 20] and beyond [54]. 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 ben- efit generalization. BN has been a foundation of many state- of-the-art computer vision algorithms.

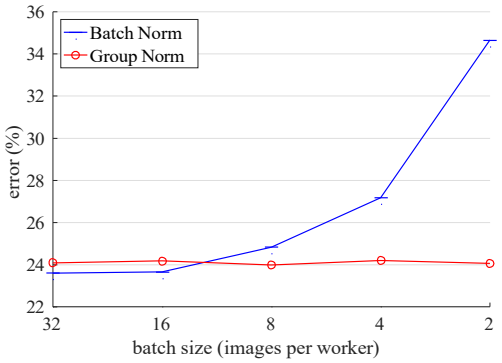

Figure 1. **ImageNet classification error *vs.* batch sizes**. This is a ResNet-50 model trained in the ImageNet training set using 8 workers (GPUs), evaluated in the validation set.

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[2] [26, 59, 20]). A small batch leads to inaccurate estimation of the batch statistics, and *reducing BN's batch size increases the model error dramatically* (Figure 1). As a result, many recent models [59, 20, 57, 24, 63] 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 [12, 47, 18], segmentation [38, 18], video recognition [60, 6], and other high-level systems built on them. For example, the Fast/er and Mask N frameworks [12, 47, 18] use a batch size of 1 or 2 images because of higher resolution, where BN is "frozen" by transforming to a linear layer [20]; in video classification with 3D convolutions [60, 6], 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.

/facebookresearch/Detectron/blob/master/projects/GN.

[2] 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.

This paper presents Group Normalization (GN) as a simple alternative to BN. We notice that many classical features like SIFT [39] and HOG [9] 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 [20] in ImageNet [50]. With a regular batch size, GN is comparably good as BN (with a gap of ~0.5%) and outperforms other normalization variants [3, 61, 51]. 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 N for COCO object detection and segmentation [37], and on 3D convolutional networks for Kinetics video classification [30]. 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) [3] and Instance Normalization (IN) [61] (Figure 2), that also avoid normalizing along the batch dimension. These methods are effective for training sequential models (RNN/LSTM [49, 22]) or generative models (GANs [15, 27]). 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. It is well-known that normalizing the input data makes training faster [33]. To normalize hidden features, initialization methods [33, 14, 19] 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) [40, 28, 32] was a component in AlexNet [32] and following models [64, 53, 58]. Unlike recent methods [26, 3, 61], LRN computes the statistics in a small neighborhood for each pixel.

Batch Normalization [26] 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 [26], 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 [45]. 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 [3, 61, 51, 2, 46] have been proposed to avoid exploiting the batch dimension. Layer Normalization (LN) [3] operates along the channel dimension, and Instance Normalization (IN) [61] performs BN-like computation but only for each sample (Figure 2). Instead of operating on features, Weight Normalization (WN) [51] 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 [25] 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 [25].

There are also attempts to *avoid* using small batches. The object detector in [43] 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 [10]), 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.*, [25, 43]), our normalization method inherently avoids this computation.

Group-wise computation. *Group convolutions* have been presented by AlexNet [32] 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 [63] 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 [23] and Xception [7] exploit *channel-wise* (also called "depth-wise") convolutions, which are group convolutions with a group number equal to the channel

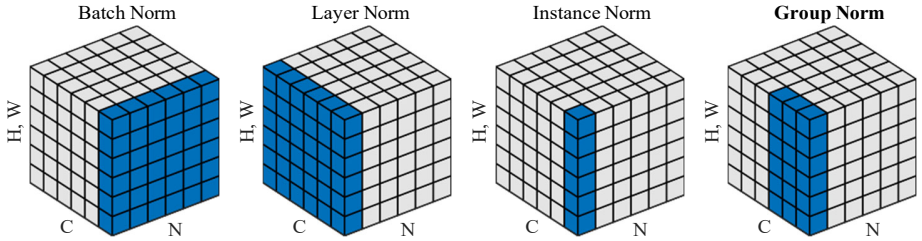

Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

number. ShuffleNet [65] 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 [20].

3. Group Normalization

The channels of visual representations are not entirely independent. Classical features of SIFT [39], HOG [9], 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 [29] and Fisher Vectors (FV) [44] 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 [11, 8], then the corresponding channels of these filters can be normalized together.

The higher-level layers are more and their behaviors are not as intuitive. However, in addition to orientations (SIFT [39], HOG [9], or [11, 8]), 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 [21, 52, 55, 5], "with various receptive-field centers (covering the visual field) and with various spatiotemporal frequency tunings" (p183, [21]); this can happen not only in the primary visual cortex, but also "throughout the visual system" [5]. 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:

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

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:

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

with ε as a small constant. 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 S_i is defined (Figure 2), discussed as follows.

In **Batch Norm** [26], the set S_i is defined as:

$$S_i = \{k \mid k_C = i_C\}, \tag{3}$$

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** [3], the set is:

$$S_i = \{k \mid k_N = i_N\}, \tag{4}$$

meaning that LN computes μ and σ along the (C, H, W) axes for each sample. In **Instance Norm** [61], the set is:

$$S_i = \{k \mid k_N = i_N, k_C = i_C\}. \tag{5}$$

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 [26], 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 S_i defined as:

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

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 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).

Relation to Layer Normalization [3]. GN becomes LN if we set the group number as $G = 1$. LN assumes *all* channels in a layer make "similar contributions" [3]. Unlike the case of fully-connected layers studied in [3], this assumption can be less valid with the presence of convolutions, as discussed in [3]. 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).

Relation to Instance Normalization [61]. 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 [42] and TensorFlow [1] 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.

```
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
```

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

4. Experiments

4.1. Image Classification in ImageNet

We experiment in the ImageNet classification dataset [50] with 1000 classes. We train on the ~1.28M training images and evaluate on the 50,000 validation images, using the ResNet models [20].

Implementation details. As standard practice [20, 17], 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 [19] 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 [16] (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 [17] but unlike [20, 16]). 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 [58] as implemented by [17]. 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 [16]. Other implementation details follow [17].

Our baseline is the ResNet trained with BN [20]. 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) [26, 20]. 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.

Figure 4 shows that *all* of these normalization methods are able to converge. LN has a small degradation of 1.7%