

# Differentiable Learning-to-Normalize via Switchable Normalization

Ping Luo\*

The Chinese University of Hong Kong  
pluo@ie.cuhk.edu.hk

Jiamin Ren\*

Zhanglin Peng  
SenseTime Research  
{renjiamin, pengzhanglin}@sensetime.com

## Abstract

We address a learning-to-normalize problem by proposing *Switchable Normalization (SN)*, which learns to select different operations for different normalization layers of a deep neural network (DNN). SN switches among three distinct scopes to compute statistics (means and variances) including a channel, a layer, and a minibatch, by learning their importance weights in an end-to-end manner. SN has several good properties. First, it adapts to various network architectures and tasks (see Fig.1). Second, it is robust to a wide range of batch sizes, maintaining high performance when small minibatch is presented (e.g. 2 images/GPU). Third, SN treats all channels as a group, unlike group normalization that searches the number of groups as a hyper-parameter. Without bells and whistles, SN outperforms its counterparts on various challenging problems, such as image classification in ImageNet, object detection and segmentation in COCO, artistic image stylization, and neural architecture search. We hope SN will help ease the usages and understand the effects of normalization techniques in deep learning. The code of SN will be made available in <https://github.com/switchablenorms/>.

## 1. Introduction

Normalization techniques have been recognized as very effective components in deep learning, largely pushing the frontier in computer vision [2, 6] and machine learning [25]. In recent years, many normalization layers such as batch normalization [12], instance normalization [31], and layer normalization [1] have been developed to address different problems with different network architectures. Despite their great successes, the operations in these normalization layers are manually designed, and existing practices often employed the same operation in all normalization layers of an entire deep network.

For example, **batch normalization (BN)** [12] was first

\*The first two authors contribute equally.

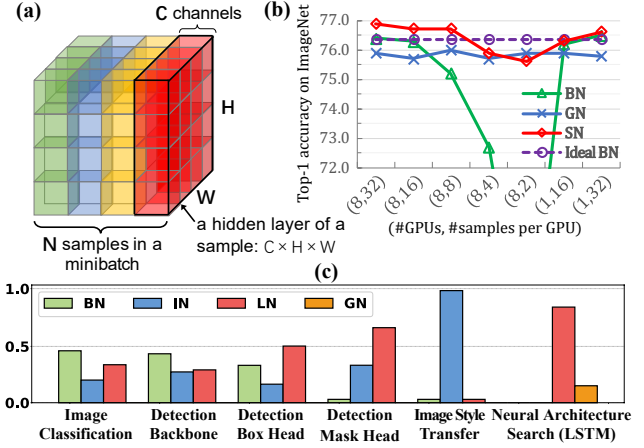


Figure 1: (a) In a CNN, the feature maps of a hidden convolutional layer is of size  $N \times C \times H \times W$ , which represents  $N$  samples in a minibatch,  $C$  channels of a layer, height and width of a channel respectively. We have  $N = 4$  in (a) as an illustrative example. Different normalization methods compute statistics (means and variances) along different axes of  $N, C, H, W$ . (b) The top-1 accuracies of ResNet50 trained with SN on ImageNet are compared with BN and GN in different batch sizes. For instance, all methods are compared to an ideal case, ‘ideal BN’, whose accuracies are 76.4%, which is the result of  $(8,32)$  of BN. This ideal case isn’t obtainable in practice. In fact, when the minibatch size decreases, BN’s accuracies drop significantly, while SN and GN both maintain reasonably good performance. SN surpasses or is comparable to both BN and GN in all settings. (c) shows that SN adapts to various network architectures and tasks, by learning an importance weight to select each normalization method (i.e. a ratio between 0 and 1; all the weights of each task sum to 1).

demonstrated in the task of image classification [27, 8, 7, 16] by normalizing the hidden feature maps of convolutional neural networks (CNNs). In particular, the feature maps of a convolutional layer in a CNN can be represented by a 4D tensor with 4 axes,  $(N, C, H, W)$ , representing minibatch size<sup>1</sup>, number of channels, height and width of

<sup>1</sup>This work represents a batch as a 2-tuple,  $(\#GPUs, \#samples \text{ per GPU})$ , and mainly investigates a regime where the gradients in training are averaged over all GPUs and the statistics of all normalization methods are

a channel respectively, as shown in Fig.1 (a). For each channel, BN computes a mean and a variance by averaging the pixel values over  $H$  and  $W$ , as well as over the corresponding channels of the other samples in the minibatch. The BN layer is often stacked after a convolutional layer and before an activation function, to normalize each channel to have zero mean and unit variance.

Another example is **instance normalization (IN)** [31], which was established in the task of artistic image style transfer [14, 10]. Different from BN that performs normalization across samples in the minibatch, IN normalizes each sample *independently* by computing the statistics along the region of  $H$  and  $W$  in a *channel-wise* manner.

Furthermore, **layer normalization (LN)** [1] was proposed to ease optimization of recurrent neural networks (RNNs). Similar to IN, the statistics of LN are not computed across the  $N$  samples in a minibatch. LN’s statistics are estimated in a *layer-wise* manner for each sample independently, by averaging the pixel values not only along  $H$  and  $W$ , but also across the  $C$  channels.

Despite the above successes, different normalization methods are proposed to train deep neural networks to solve different tasks, making model design cumbersome. Also, existing practices often employed the same operation in all normalization layers of an entire network, rendering suboptimal results.

To address these issues, we propose *Switchable Normalization (SN)*, which combines three types of statistics computed channel-wise, layer-wise, and minibatch-wise by using IN, LN, and BN respectively as introduced above. SN selects among them by learning their importance weights.

By design, *SN is adaptive to various deep networks and tasks*. As shown in Fig.1 (c), the ratios of IN, LN, and BN in SN are compared by using multiple tasks. We see that using one normalization method uniformly is not optimal for these tasks. In general, image classification and object detection prefer the combination of all three methods. In particular, SN chooses BN more than IN and LN in image classification and the backbone network of object detection, while LN has larger weights in the box and mask detection heads. For artistic image style transfer [14], SN learns to select IN. For LSTM, SN prefers LN more than group normalization (GN) [34], which is a variant of IN by dividing channels into groups.

The selectivity of normalization methods makes *SN robust to minibatch size*. As shown in Fig.1 (b), when training ResNet50 [7] on ImageNet [2] with different batch sizes, SN is close to the “ideal case” more than BN and GN. For (8, 32) as an example, ResNet50 trained with SN is able to achieve 76.9% top-1 accuracy, surpassing BN and GN by 0.5% and 1.0% respectively. In general, SN obtains better

estimated in each GPU. The *minibatch size* refers to the number of samples per GPU, while the *batch size* is “#GPUs” times “#samples per GPU”.

or comparable results than both BN and GN in all batch sizes.

Overall, this work has three key **contributions**. (1) We introduce Switchable Normalization (SN), a learning-to-normalize framework to unify various normalization operations, adapting them to different networks and tasks without choosing them manually. *SN enables each normalization layer in a DNN to have its own operation*.

(2) SN is applicable in both CNNs and RNNs/LSTMs and improves the other normalization techniques on many challenging benchmarks, including image classification in ImageNet [27], object detection and segmentation in COCO [18], artistic image stylization [14], and neural architecture search using LSTMs [25].

(3) SN is simple yet effective and easy to implement. We will make its code available. SN helps ease the selection and usage of normalization techniques in deep learning, and also pushes the frontier of architecture design in deep networks. We recommend SN as an alternative of existing handcrafted normalization approaches.

In the following, we will first present SN in Sec.2 and then discuss its relationships with previous work in Sec.3. We compare SN to related work by experiments in Sec.4

## 2. Switchable Normalization (SN)

We first describe a general formulation of a normalization layer, and then present SN.

**A General Form.** We take CNN as an illustrative example. Let  $h$  be the input data of an arbitrary normalization layer, and it is represented by a 4D tensor of  $(N, C, H, W)$ . Let  $h_{ncij}$  and  $\hat{h}_{ncij}$  be a pixel before and after normalization, where  $n \in [1, N]$ ,  $c \in [1, C]$ ,  $i \in [1, H]$ , and  $j \in [1, W]$  denote the indexes of a pixel. Let  $\mu$  and  $\sigma$  be a mean and a standard deviation. We have

$$\hat{h}_{ncij} = \gamma \frac{h_{ncij} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \quad (1)$$

where  $\gamma$  and  $\beta$  are a scale and a shift parameter respectively.  $\epsilon$  is a small constant to prevent numerical instability. Eqn.(1) shows that each pixel is normalized by using  $\mu$  and  $\sigma$ , and then re-scale and re-shift to preserve its representation capacity.

IN, LN, and BN share the formulation of Eqn.(1), but they use different sets of pixels to estimate  $\mu$  and  $\sigma$ . Therefore, the numbers of their statistics are different. In general, we have

$$\mu_k = \frac{1}{|I_k|} \sum_{(n,c,i,j) \in I_k} h_{ncij}, \quad \sigma_k^2 = \frac{1}{|I_k|} \sum_{(n,c,i,j) \in I_k} (h_{ncij} - \mu_k)^2, \quad (2)$$

where the subscript  $k \in \{\text{in}, \text{ln}, \text{bn}\}$  is used to distinguish different methods.  $I_k$  is a set pixels and  $|I_k|$  denotes the

number of pixels. Specifically,  $I_{\text{in}}$ ,  $I_{\text{ln}}$ , and  $I_{\text{bn}}$  are the sets of pixels corresponding to different normalization methods.

In IN [31], we have  $\mu_{\text{in}}$  and  $\sigma_{\text{in}}^2 \in \mathbb{R}^{N \times C}$ , and  $I_{\text{in}} = \{(i, j) | i \in [1, H], j \in [1, W]\}$ , meaning that IN has  $2NC$  elements of statistics, where a mean value and a variance value are computed along  $(H, W)$  for each one of the  $C$  channels in each one of the  $N$  samples.

In LN [1], we have  $\mu_{\text{ln}}, \sigma_{\text{ln}}^2 \in \mathbb{R}^{N \times 1}$  and  $I_{\text{ln}} = \{(c, i, j) | c \in [1, C], i \in [1, H], j \in [1, W]\}$ , implying that LN has  $2N$  statistical values, where a mean and a variance are computed along  $(C, H, W)$  for each one of the  $N$  samples.

In BN [12], we have  $\mu_{\text{bn}}, \sigma_{\text{bn}}^2 \in \mathbb{R}^{C \times 1}$  and  $I_{\text{bn}} = \{(n, i, j) | n \in [1, N], i \in [1, H], j \in [1, W]\}$ , in the sense that BN does not normalize across  $C$  channels, but across  $H, W$ , and the  $N$  samples in a minibatch, leading to  $2C$  elements of statistics.

## 2.1. Formulation of SN

SN has an intuitive expression

$$\hat{h}_{ncij} = \gamma \frac{h_{ncij} - \sum_{k \in \Omega} w_k \mu_k}{\sqrt{\sum_{k \in \Omega} w'_k \sigma_k^2 + \epsilon}} + \beta, \quad (3)$$

where  $\Omega$  is a set of different kinds of statistics. In this work, we define  $\Omega = \{\text{in}, \text{ln}, \text{bn}\}$  the same as in Eqn.(2).  $w_k$  and  $w'_k$  are importance weights used to weighted average the means and variances respectively. Each weight in  $w_k$  and  $w'_k$  is a scalar variable, which is shared across all channels. There are  $3 \times 2 = 6$  importance weights in SN. We have  $\sum_{k \in \Omega} w_k = 1$ ,  $\sum_{k \in \Omega} w'_k = 1$ , and  $\forall w_k, w'_k \in [0, 1]$ , and define

$$w_k = \frac{e^{\lambda_k}}{\sum_{z \in \{\text{in}, \text{ln}, \text{bn}\}} e^{\lambda_z}}, \quad k \in \{\text{in}, \text{ln}, \text{bn}\}, \quad (4)$$

where each  $w_k$  is computed by using a softmax function with  $\lambda_{\text{in}}$ ,  $\lambda_{\text{ln}}$ , and  $\lambda_{\text{bn}}$  as the parameters, which can be learned by back-propagation (BP).  $w'_k$  are defined similarly by using another three parameters  $\lambda'_{\text{in}}$ ,  $\lambda'_{\text{ln}}$ , and  $\lambda'_{\text{bn}}$ .

Furthermore,  $\mu_k$  and  $\sigma_k^2$  in Eqn.(3) can be simply calculated by following Eqn.(2). However, this strategy leads to large redundant computations. In fact, the three kinds of statistics of SN depend on each other, we could reduce redundancy by reusing computations. We have

$$\begin{aligned} \mu_{\text{in}} &= \frac{1}{HW} \sum_{i,j}^{H,W} h_{ncij}, \quad \sigma_{\text{in}}^2 = \frac{1}{HW} \sum_{i,j}^{H,W} (h_{ncij} - \mu_{\text{in}})^2, \\ \mu_{\text{ln}} &= \frac{1}{C} \sum_{c=1}^C \mu_{\text{in}}, \quad \sigma_{\text{ln}}^2 = \frac{1}{C} \sum_{c=1}^C (\sigma_{\text{in}}^2 + \mu_{\text{in}}^2) - \mu_{\text{ln}}^2, \\ \mu_{\text{bn}} &= \frac{1}{N} \sum_{n=1}^N \mu_{\text{in}}, \quad \sigma_{\text{bn}}^2 = \frac{1}{N} \sum_{n=1}^N (\sigma_{\text{in}}^2 + \mu_{\text{in}}^2) - \mu_{\text{bn}}^2, \end{aligned} \quad (5)$$

which show that the means and variances of LN and BN can be computed based on IN. Using Eqn.(5), the computational complexity of SN is  $\mathcal{O}(NCHW)$ , which is comparable to previous work.

**Discussion.** To select among different normalization approaches, one straight-forward solution is to treat them separately as different layers and then learns to aggregate their normalized features. However, this solution has high computations and large memory footprints. Instead, as shown in Eqn.(3), SN is designed to reduce complexity by combining statistics rather than features. Their relationships can be understood by rewriting Eqn.(3) as  $\gamma(w_{\text{bn}}(\frac{h_{ncij} - \mu_{\text{bn}}}{\sqrt{\sum_k w'_k \sigma_k^2 + \epsilon}}) + w_{\text{in}}(\frac{h_{ncij} - \mu_{\text{in}}}{\sqrt{\sum_k w'_k \sigma_k^2 + \epsilon}}) + w_{\text{ln}}(\frac{h_{ncij} - \mu_{\text{ln}}}{\sqrt{\sum_k w'_k \sigma_k^2 + \epsilon}})) + \beta$ , which switches among normalization methods. For example, when  $w_{\text{bn}} = w'_{\text{bn}} = 1$ , this equation reduces to BN,  $\gamma \frac{h_{ncij} - \mu_{\text{bn}}}{\sqrt{\sigma_{\text{bn}}^2 + \epsilon}} + \beta$ .

**Inference.** When applying SN in test, the statistics of IN and LN are computed independently for each testing sample, while BN uses *batch average* instead of the moving average as used in the ordinary BN. Here the *batch average* is obtained by two steps. First, we freeze the parameters of the network and all the SN layers, and then feed the network with a certain number of minibatches randomly chosen from the training set. Second, we average the means and variances produced by all these minibatches in each SN layer. The averaged statistics are used by BN of SN.

The above procedure provides a stable estimation more than the moving average does. A small amount of samples is sufficient to perform batch average after training, without computing moving average in each forward-propagation step during training. To see this, the top-1 accuracies of ResNet50 on ImageNet by using moving average, 50k, and all training samples to estimate the statistics are 76.89%, 76.90%, and 76.92% respectively. More comparisons between moving average and batch average are provided in the Appendix.

**Implementation.** SN can be easily implemented in existing platforms such as PyTorch and TensorFlow. A simple implementation is shown in Fig.7 in the Appendix. The backward computation of SN can be obtained by automatic differentiation (AD) in these platforms. In the platform implemented without AD (e.g. CUDA), we need to implement backward gradient propagation of SN, where the errors are propagated through  $\mu_k$  and  $\sigma_k^2$ . We provide the deviations in the Appendix.

## 3. Relations to Previous Work

**Normalization.** In Table 1, we compare SN to BN, IN, LN, and GN, as well as three variants of BN including Batch Renormalization (BRN) [11], Batch Kalman Normalization (BKN) [32], and Weight Normalization (WN) [28]. In general, we see that SN possesses comparable numbers

	Parameter			Statistical Estimation			Network		Task				
	params	#params	hyper-params	statistics	computation complexity	#statistics	CNN	RNN/LSTM	classification	detection	segmentation	image style	NAS
BN [12]	$\gamma, \beta$	$2C$	$p, \epsilon$	$\mu, \sigma, \mu', \sigma'$	$\mathcal{O}(NCHW)$	$2C$	✓		✓	✓			
IN [31]	$\gamma, \beta$	$2C$	$\epsilon$	$\mu, \sigma$	$\mathcal{O}(NCHW)$	$2CN$	✓					✓	
LN [1]	$\gamma, \beta$	$2C$	$\epsilon$	$\mu, \sigma$	$\mathcal{O}(NCHW)$	$2N$		✓					✓
GN [34]	$\gamma, \beta$	$2C$	$g, \epsilon$	$\mu, \sigma$	$\mathcal{O}(NCHW)$	$2gN$	✓		✓	✓	✓		
BRN [11]	$\gamma, \beta$	$2C$	$p, \epsilon, r, d$	$\mu, \sigma, \mu', \sigma'$	$\mathcal{O}(NCHW)$	$2C$	✓		✓				
BKN [32]	$A$	$C^2$	$p, \epsilon$	$\mu, \Sigma, \mu', \Sigma'$	$\mathcal{O}(NC^2HW)$	$C + C^2$	✓		✓				
WN [28]	$\gamma$	$C$	—	—	—	—		✓					✓
SN	$\gamma, \beta, \{w_k\}_{k \in \Omega}$	$2C + 6$	$p, \epsilon$	$\{\mu_k, \sigma_k\}_{k \in \Omega}$	$\mathcal{O}(NCHW)$	$2C + 2N + 2CN$	✓	✓	✓	✓	✓	✓	✓

Table 1: **Comparisons of normalization methods** in *four* aspects including parameters, statistics, networks, and tasks. First, we compare their types of parameters, numbers of parameters (#params), and hyper-parameters. Second, we compare their types of statistics, computational complexity to estimate statistics, and numbers of statistics (#statistics). Finally, for networks and tasks, we consider whether they work well in CNNs and RNNs/LSTMs, as well as five different challenging problems including image classification in ImageNet [2], object detection and segmentation in COCO [18], artistic image stylization, and neural architecture search (NAS) using LSTMs. Specifically,  $\gamma, \beta$  denote the scale and shift parameters.  $\mu, \sigma, \Sigma$  are a vector of means, a vector of standard deviations, and a covariance matrix.  $\mu'$  represents the moving average. Moreover,  $p$  is the momentum of moving average,  $g$  in GN is the number of groups,  $\epsilon$  is a small value for numerical stability, and  $r, d$  are used in BRN. In SN,  $k \in \Omega$  indicates a set of different kinds of statistics,  $\Omega = \{\text{in, ln, bn}\}$ , and  $w_k$  is an importance weight of each kind.

of parameters and computations, as well as rich statistics. Details are presented as below.

- First, SN has similar number of parameters compared to previous methods, as shown in the first portion of Table 1. Most of the approaches learn a scale parameter  $\gamma$  and a bias  $\beta$  for each one of the  $C$  channels, resulting in  $2C$  parameters. SN learns 6 importance weights as the additional parameters. In Table 1, we see that BKN has the maximum number of parameters  $C^2$ , as it learns an affine transformation matrix  $A$  of the means and variances. WN has  $C$  scale parameters without the biases.

Furthermore, many methods have  $p$  and  $\epsilon$  as hyper-parameters, whose values are not sensitive because they are often fixed in different tasks. In contrast, GN and BRN have to search the number of groups  $g$  or the renormalization parameters  $r, d$ , which may have different values in different networks. Moreover, WN does not have hyper-parameters and statistics, since it performs normalization in the space of network parameters rather than feature space. Salimans *et al.* [28] showed that WN is a special case of BN.

- Second, although SN has richer statistics, the computational complexity to estimate them is comparable to previous methods, as shown in the second portion of Table 1. As introduced in Sec.2, IN, LN, and BN estimate the means and variances along axes  $(H, W)$ ,  $(C, H, W)$ , and  $(N, H, W)$  respectively, leading to  $2CN$ ,  $2N$ , and  $2C$  numbers of statistics. Therefore, SN has  $2CN + 2N + 2C$  statistics by combining them. Although BKN has the largest number of statistics  $C + C^2$ , as it estimates the covariance matrix other than a vector of variances, it also has the highest computations. Also, approximating the covariance matrix in a minibatch is nontrivial as discussed in [3, 21, 20]. Besides  $\mu$  and  $\sigma$ , BN, BRN, and BKN also compute their moving averages.

- Third, SN is demonstrated in both CNNs and LSTMs, as well as five different challenging tasks, including image classification in ImageNet [2], object detection and segmentation in COCO [18], artistic image stylization, and neural architecture search (NAS) by using LSTMs [25].

**Switchable Architecture.** The concept of switchable operation in SN is related to previous work. An early classic work is the mixture of experts (MoE) [13], which trains multiple learners to divide and solve a complicated problem. MoE has been extended in multiple ways. For example, Nair and Hinton [23] proposed mixture of RBMs by including a gate variable to determine which RBM should be activated. Luo *et al.* [22] introduced switchable RBMs for pedestrian detection by incorporating switch, mask, and gate variables to model different human parts, as well as foreground and background of images. Recently, Shazeer *et al.* [29] inserted an outrageously-large MoE layer into a DNN, which switches among thousands of feed-forward sub-branches to increase model capacity. To our knowledge, SN is the first framework to learn different operations for different normalization layers in a DNN. It pushes the frontier of network architecture design in deep learning.

## 4. Experiments

We evaluate SN on multiple challenging problems, including image classification in ImageNet [27], object detection and segmentation in COCO [18], artistic image stylization [14], and neural architecture search by using LSTMs [19].

### 4.1. Image Classification in ImageNet

SN is first compared with BN and GN on the ImageNet classification dataset of 1k categories. All the normalization



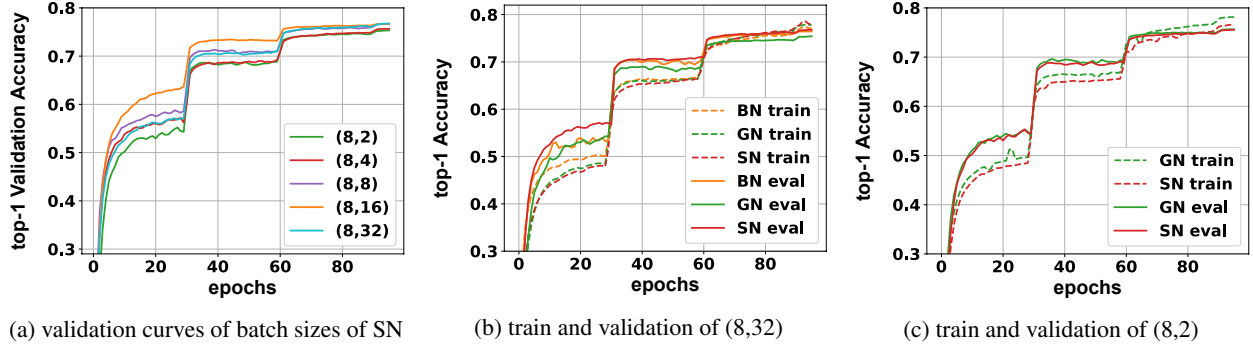


Figure 2: **Comparisons of learning curves.** (a) visualizes the validation curves of SN with different settings of batch size. The bracket  $(\cdot, \cdot)$  denotes  $(\#GPUs, \#samples \text{ per GPU})$ . (b) compares the top-1 train and validation curves on ImageNet of SN, BN, and GN in the batch size of (8,32). (c) compares the train and validation curves of SN and GN in the batch size of (8,2).

methods adopt ResNet50 as backbone network.

**Protocol.** All models are trained on 1.2M images and evaluated on 50K validation images. They are trained by using SGD with different settings of batch sizes, which are denoted as a 2-tuple,  $(number \text{ of GPUs}, number \text{ of samples per GPU})$ . For each setting, the gradients are aggregated over all GPUs, and the means and variances of the normalization methods are computed in each GPU. The network parameters are initialized by following [7]. For all normalization methods, all  $\gamma$ 's<sup>2</sup> are initialized as 1 and all  $\beta$ 's as 0. The parameters of SN ( $\lambda_k$  and  $\lambda'_k$ ) are initialized as 1. We use a weight decay of  $10^{-4}$  for all parameters including  $\gamma$  and  $\beta$ . All models are trained for 100 epoches with a initial learning rate of 0.1, which is decreased by  $10\times$  after 30, 60, and 90 epoches. During training, we employ data augmentation the same as [7]. The top-1 classification accuracy on the  $224\times 224$  center crop is reported.

**Performance Comparison.** The top-1 accuracies of SN are compared to BN and GN as shown in the upper part of Table 2, where the first five columns are the results trained on 8 GPUs and the minibatch size is decreased from 32 to 2. The middle two columns show the results of minibatch sizes of 16 and 32 on a single GPU, and the last two columns have the same batch size of 8, where (8, 1) is an extreme case whose minibatch only has a single sample.

In the first five columns of Table 2, we see that GN obtains around 75.9% in all cases, while SN outperforms BN in all cases and surpasses GN in 4 cases, rendering its robustness to different batch sizes. However, the accuracy of BN reduces by 1.1% from (8, 16) to (8, 8) and declines to 65.3% of (8, 2), implying that BN is unsuitable in small

	(8,32)	(8,16)	(8,8)	(8,4)	(8,2)	(1,16)	(1,32)	(8,1)	(1,8)
BN [12]	76.4	76.3	75.2	72.7	65.3	76.2	76.5	–	75.4
GN [34]	75.9	75.8	76.0	75.8	<b>75.9</b>	75.9	75.8	<b>75.5</b>	75.5
SN	<b>76.9</b>	<b>76.7</b>	<b>76.7</b>	<b>75.9</b>	75.6	<b>76.3</b>	<b>76.6</b>	75.0*	<b>75.9</b>
GN–BN	-0.5	-0.5	0.8	3.1	10.6	-0.3	-0.7	–	0.1
SN–BN	0.5	0.4	1.5	3.2	10.3	0.1	0.1	–	0.5
SN–GN	1.0	0.9	0.7	0.1	-0.3	0.4	0.8	-0.5	0.4

Table 2: **Comparisons of top-1 accuracies** on the validation set of ImageNet, by using ResNet50 trained with SN, BN, and GN in different batch size settings. The bracket  $(\cdot, \cdot)$  denotes  $(\#GPUs, \#samples \text{ per GPU})$ . In the bottom part, “GN–BN” indicates the difference between the accuracies of GN and BN. The “–” in (8, 1) of BN indicates it does not converge.

\* For (8, 1), SN contains IN and SN without BN, as BN is the same as IN in training.

minibatch, where the random noise from the statistics is too heavy. Fig. 2 (a) plots the validation curves of SN. Fig. 2 (b) and (c) compare the training and validation curves of SN to BN and GN in (8, 32) and (8, 2) respectively. From all these curves, we see that SN enables faster convergence while maintains higher or comparable accuracies than those of BN and GN.

In the middle two columns of Table 2, the settings of (1, 16) and (1, 32) average the gradients in a single GPU by using only 16 and 32 samples respectively. Their batch sizes are the same as (8, 2) and (8, 4). SN again performs best in these single-GPU settings, while BN outperforms GN. For example, unlike (8, 4) that uses distributed training on 8 GPUs, BN achieves 76.5% in (1, 32), which is the best-performing result of BN, although the batch size to compute the gradients is as small as 32. From the above results, we see that BN’s performance are sensitive to the statistics more than the gradients, while SN are robust to both of them.

The two settings in the last two columns of Table 2 have the same batch size of 8. The (1, 8) has a minibatch size of 8, while (8, 1) is an extreme case with a single sample in a minibatch. For (1, 8), SN performs best. In the extreme case of (8, 1), SN consists of IN and LN but no BN, because

<sup>2</sup>In [34], the  $\gamma$  of each residual block’s last normalization layer is initialized as 0 following [5], to make the forward/backward signal initially propagated through an identity shortcut. We found that this provides 0.15 improvement to the top-1 accuracy when minibatch size is 32, but it doesn’t have significant impact in small minibatch. For direct comparisons of the normalization approaches, we set all  $\gamma$ ’s to 1 without using this heuristic strategy.

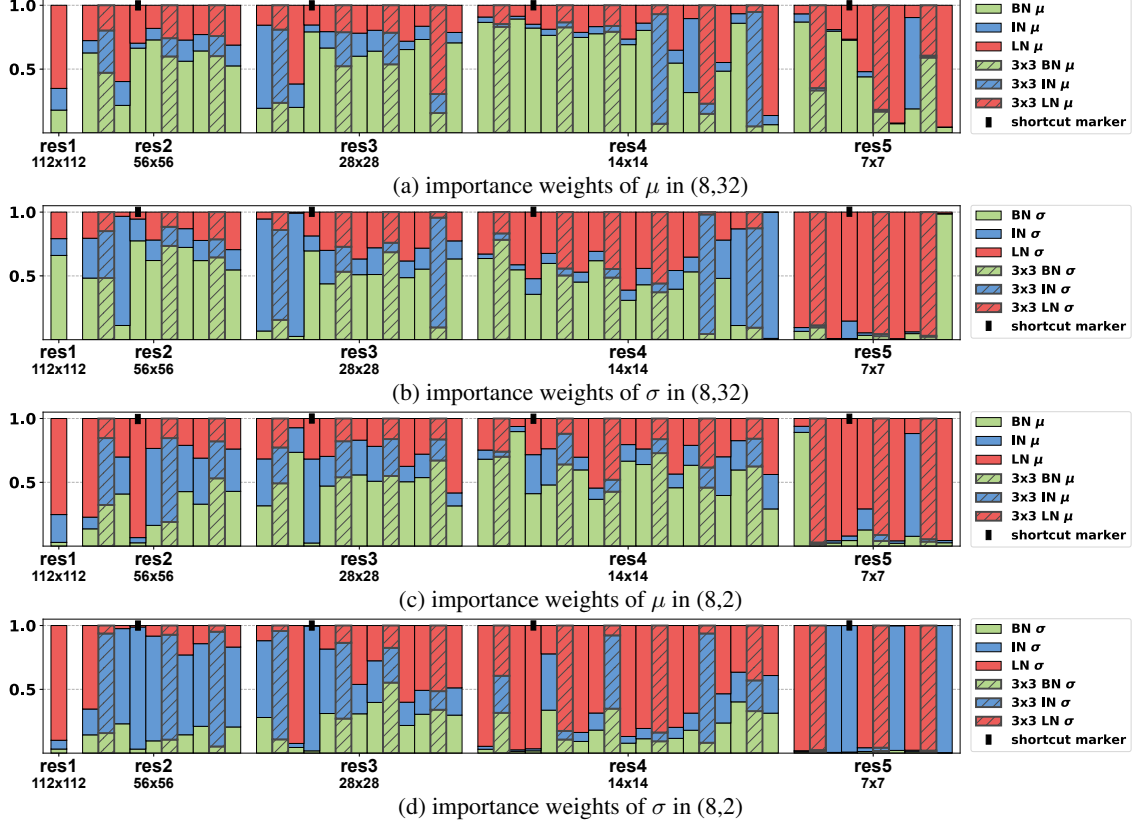


Figure 3: **Selected operations of each SN layer in ResNet50.** There are 53 SN layers. (a,b) show the importance weights for  $\mu$  and  $\sigma$  of (8, 32), while (c,d) show those of (8, 2). The  $y$ -axis represents the importance weights that sum to 1, while the  $x$ -axis shows different residual blocks of ResNet50. The SN layers in different places are highlighted differently. For example, the SN layers follow the  $3 \times 3$  conv layers are outlined by shaded color, those in the shortcuts are marked with “■”, while those follow the  $1 \times 1$  conv layers are in flat color. The first SN layer follows a  $7 \times 7$  conv layer. We see that SN learns distinct importance weights for different normalization methods as well as  $\mu$  and  $\sigma$ , adapting to different batch sizes, places, and depths of a deep network.

IN and BN are the same in training when the minibatch size is 1. In this case, both SN and GN still perform reasonably well, while BN fails to converge.

**SN vs. IN and LN.** IN and LN are not optimal in image classification as reported in [31] and [1]. With a regular setting of (8, 32), ResNet50 trained with IN and LN achieve 71.6% and 74.7% respectively, which reduce 5.3% and 2.2% compared to 76.9% of SN.

**SN vs. BRN and BKN.** BRN has two extra hyper-parameters,  $r_{max}$  and  $d_{max}$ , which renormalize the means and variances. We choose their values as  $r_{max} = 1.5$  and  $d_{max} = 0.5$ , which work best for ResNet50 in the setting of (8, 4) following [11], where 73.7% of BRN surpasses 72.7% of BN by 1%, but it reduces 2.2% compared to 75.9% of SN.

BKN [32] estimates the statistics in the current layer by combining those computed in the preceding layers. It estimates the covariance matrix rather than the variance vector. In particular, how to combine the layers and estimate the

statistics requires careful design for every specific network. For ResNet50 in (8, 32), BKN achieved 76.8%, which is comparable to 76.9% of SN. However, for small minibatch, BKN reported 76.1%, which was evaluated in a micro-batch setting where 256 samples are used to compute the gradients and 4 samples to estimate the statistics. This setting is easier than (8, 4), which uses 32 samples to compute the gradients. Furthermore, it is unclear how to apply BRN and BKN in the other tasks such as object detection and segmentation.

**Ablation study of importance weights.** Fig. 1 (c) and Fig. 4 plot histograms to compare the importance weights of SN with respect to different tasks and batch sizes respectively. These histograms are computed by averaging the importance weights of all SN layers in a network. They show that SN adapts to various scenarios by changing its importance weights. For example, SN prefers BN when the minibatch is sufficiently large, while it selects LN instead when small minibatch is presented, as shown in the green and red bars of Fig. 4.

In particular, the selected operations of each SN layer are shown in Fig.3. We have several observations. First, for the same batch size, the weights of  $\mu$  and  $\sigma$  could have notable differences, especially when comparing ‘res1,4,5’ of (a,b) and ‘res2,4,5’ of (c,d). For example,  $\sigma$  of BN (green) in ‘res5’ in (b,d) are mostly reduced compared to  $\mu$  of BN in (a,c). As discussed in [12, 28], this is because the variance estimated in a minibatch produces larger noise than the mean, leading to instable training. SN is able to restrain the noisy statistics.

Second, the SN layers in different places of a network may select distinct operations. In other words, when comparing the adjacent SN layers after the  $3 \times 3$  conv layer, shortcut, and the  $1 \times 1$  conv layer, we see that they may choose different importance weights, *e.g.* ‘res2,3’. The selectivity of operations in different places (normalization layers) of a deep network has not been observed in previous work.

Third, deeper layers prefer LN and IN more than BN, as illustrated in ‘res5’, which tells us that putting BN in an appropriate place is crucial in network architecture design. Although the stochastic uncertainty in BN (*i.e.* the minibatch statistics) acts as a regularizer that might benefit generalization, using BN uniformly in all normalization layers may impede performance.

Overall, studying the importance weights in the SN layers discloses interesting characteristics and impacts of the normalization techniques in deep learning, and sheds light on network architecture design in future research.

## 4.2. Object Detection and Segmentation in COCO

Next we evaluate SN in the tasks of object detection and segmentation in COCO [18]. Unlike image classification, these two tasks benefit from large size of input images, making large memory footprint and therefore leading to small minibatch size, such as 2 samples per GPU [26, 17]. In this case, as BN is not applicable in small minibatch, previous work [26, 17, 6] often freeze BN and turns it into a constant linear transformation layer, which actually performs no normalization. Overall, SN selects different operations in different components of a detection system (see Fig.1 (c)), showing much more superiority than both BN and GN.

SN is easily plugged into different detection frameworks implemented by using different codebases. To see this, we implement it on existing detection codebases of PyTorch and Caffe2-Detectron [4] using Python and CUDA respectively. We conduct 3 settings, including **setting-1**: Faster R-CNN [26] on PyTorch; **setting-2**: Faster R-CNN+FPN [17] on Caffe2; and **setting-3**: Mask R-CNN [6]+FPN on Caffe2. For all the settings, we choose ResNet50 as the backbone network. In each setting, the experimental configurations of all the models are the same, while only

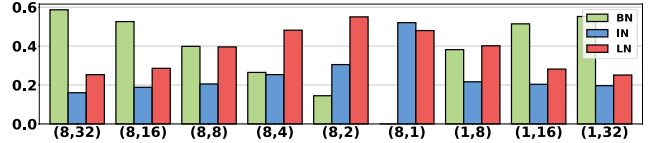


Figure 4: **Importance weights of SN.** The importance weights of BN, IN, and LN of different batch sizes are compared. The bracket  $(\cdot, \cdot)$  indicates (#GPUs, #samples per GPU). SN doesn’t have BN in (8, 1).

backbone	head	AP	AP <sub>.5</sub>	AP <sub>.75</sub>	AP <sub>l</sub>	AP <sub>m</sub>	AP <sub>s</sub>
BN <sup>†</sup>	BN <sup>†</sup>	29.6	47.8	31.9	45.5	33.0	11.5
BN	BN	19.3	33.0	20.0	32.3	21.3	7.4
GN	GN	32.7	52.4	35.1	<b>49.1</b>	36.1	14.9
SN	SN	<b>33.0</b>	<b>52.9</b>	<b>35.7</b>	48.7	<b>37.2</b>	<b>15.6</b>
BN <sup>‡</sup>	BN	20.0	33.5	21.1	32.1	21.9	7.3
GN <sup>‡</sup>	GN	28.3	46.3	30.1	41.2	30.0	12.7
SN <sup>‡</sup>	SN	<b>29.5</b>	<b>47.8</b>	<b>31.6</b>	<b>44.2</b>	<b>32.6</b>	<b>13.0</b>

Table 3: **Faster R-CNN for detection in COCO** using ResNet50 and RPN. BN<sup>†</sup> represents BN is frozen without finetuning. The superscript <sup>‡</sup> indicates the backbones are trained from scratch without pretraining on ImageNet. The best results in the upper and lower parts are bold.

the normalization layers are replaced.

**Protocol.** For **setting-1**, we employ a fast implementation [35] of Faster R-CNN in PyTorch and follow its protocol. Specifically, we train all models on 4 GPUs and 3 images per GPU. Each image is re-scaled such that its shorter side is 600 pixels. All models are trained for 80k iterations with a learning rate of 0.01 and then for another 40k iterations with 0.001. For **setting-2** and **setting-3**, we employ the configurations of the Caffe2-Detectron [4]. We train all models on 8 GPUs and 2 images per GPU. Each image is re-scaled to its shorter side of 800 pixels. The train-time proposals for FPN are 2000, while the test-time proposals are 1000. In particular, for setting-2, the learning rate (LR) is initialized as 0.02 and is decreased by a factor of 0.1 after 60k and 80k iterations and finally terminates at 90k iterations. This is referred as the 1x schedule in Detectron. In setting-3, the LR schedule is twice as long as the 1x schedule with the LR decay points scaled twofold proportionally, referred as 2x schedule. For all settings, we set weight decay to 0 for both  $\gamma$  and  $\beta$  following [34].

All the above models are trained in the 2017 train set of COCO by using SGD with a momentum of 0.9 and weight decay of  $10^{-4}$ , and tested in the 2017 val set. We report the standard metrics of COCO, including average precisions at IoU=0.5:0.05:0.75 (AP), IoU=0.5 (AP<sub>.5</sub>), and IoU=0.75 (AP<sub>.75</sub>) for both bounding box (AP<sup>b</sup>) and segmentation mask (AP<sup>m</sup>). Also, we report average precisions for small (AP<sub>s</sub>), medium (AP<sub>m</sub>), and large (AP<sub>l</sub>) objects.

**Results of Setting-1.** As shown in Table 3, SN is compared with both BN and GN in the Faster R-CNN. In

this setting, the layers up to conv4 of ResNet50 are used as *backbone* to extract features, and the layers of conv5 are used as the Region-of-Interest *head* for classification and regression. As the layers are inherited from the pretrained model, both the backbone and head involve normalization layers. Different results of Table 3 use different normalization methods in the backbone and head. Its upper part shows results of finetuning the ResNet50 models pretrained on ImageNet. The lower part compares training COCO from scratch without pretraining on ImageNet.

In the upper part of Table 3, the baseline is denoted as  $\text{BN}^\dagger$ , where the BN layers are frozen. We see that freezing BN performs significantly better than finetuning BN (29.6 vs. 19.3). SN and GN enable finetuning the normalization layers, where SN obtains the best-performing AP of 33.0 in this setting. Fig.5 (a) compares their AP curves.

As reported in the lower part of Table 3, SN and GN allow us to train COCO from scratch without pretraining on ImageNet, and they still achieve competitive results. For instance, 29.5 of  $\text{SN}^\ddagger$  outperforms  $\text{BN}^\ddagger$  by a large margin of 9.5 AP and  $\text{GN}^\ddagger$  by 1.2 AP. Their learning curves are compared in Fig.5 (b).

**Results of Setting-2.** Next, Table 4 compares SN with GN with Faster R-CNN by using ResNet50 and the Feature Pyramid Network (FPN) [17] as the backbone, which utilizes all pretrained layers to construct a pyramid, and appends randomly initialized layers as the head. Following [34], we employ a head with 4 convolutional layers and 1 fully-connected layer.

As shown in Table 4, a baseline  $\text{BN}^\dagger$  achieves an AP of 36.7 without using normalization in the detection head. When using SN and GN in the head and  $\text{BN}^\dagger$  in the backbone,  $\text{BN}^\dagger$ +SN improves the AP of  $\text{BN}^\dagger$ +GN by 0.8 (from 37.2 to 38.0). We further investigate using SN and GN in both the backbone and head. In this case, we find that GN improves  $\text{BN}^\dagger$ +SN by only a small margin of 0.2 AP (38.2 vs. 38.0), although the backbone is pretrained and finetuned by using GN. When finetuning the SN backbone, SN obtains a significant improvement of 1.1 AP over GN (39.3 vs. 38.2). Furthermore, the 39.3 AP of SN and 38.2 of GN both outperform 37.8 in [24], which synchronized the statistics of the BN layers in the backbone (*i.e.* BN layers are not frozen).

**Results of Setting-3.** We further evaluate SN in the Mask R-CNN [6] with FPN framework. Table 5 reports the results. In the upper part, SN is compared to a detection head with no normalization and a head with GN, while the backbone is pretrained with BN, which is then frozen in finetuning (*i.e.* the ImageNet pretrained features are the same). We see that the baseline  $\text{BN}^\dagger$  achieves a box AP of 38.6 and a mask AP of 34.2. SN improves GN by 0.5 box AP and 0.4 mask AP, when finetuning the same  $\text{BN}^\dagger$  backbone.

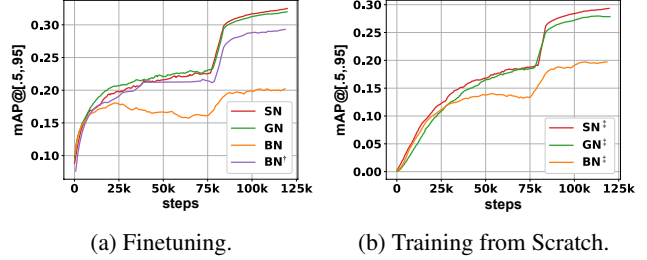


Figure 5: Average precision (AP) curves of Faster R-CNN on the 2017 val set of COCO. (a) plots the results of finetuning pretrained networks. (b) shows training the models from scratch.

backbone	head	AP	AP <sub>.5</sub>	AP <sub>.75</sub>	AP <sub>l</sub>	AP <sub>m</sub>	AP <sub>s</sub>
$\text{BN}^\dagger$	–	36.7	58.4	39.6	48.1	39.8	21.1
$\text{BN}^\dagger$	GN	37.2	58.0	40.4	48.6	40.3	21.6
$\text{BN}^\dagger$	SN	38.0	59.4	41.5	48.9	41.3	22.7
GN	GN	38.2	58.7	41.3	49.6	41.0	22.4
SN	SN	<b>39.3</b>	<b>60.9</b>	<b>42.8</b>	<b>50.3</b>	<b>42.7</b>	<b>23.5</b>

Table 4: **Faster R-CNN+FPN for detection in COCO** using ResNet50 and FPN [17] with 1x LR schedule.  $\text{BN}^\dagger$  represents BN is frozen without finetuning. The best results are bold.

backbone	head	AP <sup>b</sup>	AP <sup>b</sup> <sub>.5</sub>	AP <sup>b</sup> <sub>.75</sub>	AP <sup>m</sup>	AP <sup>m</sup> <sub>.5</sub>	AP <sup>m</sup> <sub>.75</sub>
$\text{BN}^\dagger$	–	38.6	59.5	41.9	34.2	56.2	36.1
$\text{BN}^\dagger$	GN	39.5	60.0	43.2	34.4	56.4	36.3
$\text{BN}^\dagger$	SN	40.0	61.0	43.3	34.8	57.3	36.3
GN	GN	40.2	60.9	43.8	35.7	57.8	38.0
GN	SN	40.4	61.4	44.2	36.0	58.4	38.1
SN	SN	<b>41.0</b>	<b>62.3</b>	<b>45.1</b>	<b>36.5</b>	<b>58.9</b>	<b>38.7</b>

Table 5: **Mask R-CNN for detection and segmentation in COCO** using ResNet50 and FPN [17] with 2x LR schedule.  $\text{BN}^\dagger$  represents BN is frozen without finetuning. The best results are bold.

For more direct comparison with GN as shown in the lower part of Table 5, we apply SN in the head and finetune the same backbone network pretrained with GN. In this case, SN outperforms GN by 0.2 and 0.3 box and mask APs respectively. Moreover, when finetuning the SN backbone, SN surpasses GN by a large margin of both box and mask APs (41.0 vs. 40.2 and 36.5 vs. 35.7). Note that the performance of SN even outperforms 40.9 and 36.4 of the 101-layered ResNet with the BN layers frozen [4].

### 4.3. Artistic Image Stylization

Next we evaluate SN in the task of artistic image stylization. We adopt a recent advanced approach [14], which jointly minimizes two loss functions. Specifically, one is a feature reconstruction loss that penalizes an output image when its content is deviated from a target image, and the other is a style reconstruction loss that penalizes differences in style (*e.g.* color, texture, exact boundary). [14, 10] show that IN works better than BN in this task.

We compare SN with IN and BN using VGG16 [30] as



backbone network. All models are trained on the COCO dataset [18]. For each model in training, we resize each image to  $256 \times 256$  and train for 40,000 iterations with a batch size setting of (1, 4). We do not employ weight decay or dropout. The other training protocols are the same as [14]. In test, we evaluate the trained models on  $512 \times 512$  images selected following [14].

Fig.6 (a) compares the style and feature reconstruction losses. We see that SN enables faster convergence than both IN and BN. As shown in Fig.1 (c), SN automatically selects IN in image stylization. Fig.9 visualizes some stylization results.

#### 4.4. Neural Architecture Search

Here we investigate SN in LSTM for efficient neural architecture search (ENAS) [25], which is designed to search the structures of convolutional cells. In ENAS, a convolutional neural network (CNN) is constructed by stacking multiple convolutional cells. It consists of two steps, training controllers and training child models. A controller is a LSTM whose parameters are trained by using the REINFORCE [33] algorithm to sample a cell architecture, while a child model is a CNN that stacks many sampled cell architectures and its parameters are trained by back-propagation with SGD. In [25], the LSTM controller is learned to produce an architecture with high reward, which is the classification accuracy on the validation set of CIFAR-10 [15]. Higher accuracy indicates the controller produces better architecture.

We compare SN with LN and GN by using them in the LSTM controller to improve architecture search. As BN is not applicable in LSTM and IN is equivalent to LN in fully-connected layer (*i.e.* both compute the statistics across neurons), SN combines LN and GN in this experiment. Fig.6 (b) shows the validation accuracy of CIFAR10. We see that SN obtains better accuracy than both LN and GN.

### 5. Conclusion and Discussion

We proposed SN, which is able to learn different operations in different normalization layers of a deep network. It is simple, effective, adaptable to both CNNs and LSTMs, and applicable in many important problems such as image classification in ImageNet, object detection and segmentation in COCO, artistic image stylization, and modeling long sequences. SN can be naturally extended to incorporate more normalization operations. We release its implementations and recommend it as an alternative of existing handcrafted normalization methods.

Future work may focus on several aspects. First, we will investigate SN's capability in more scenarios, such as synthesizing high-quality images by using generative adversarial networks (GANs). Second, SN in whitening is also an inspired direction. For example, SN could switch

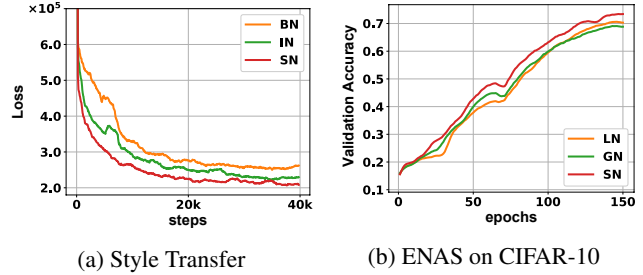


Figure 6: (a) shows the losses of BN, IN, and SN in the task of image stylization. SN converges faster than IN and BN. As shown in Fig.1 and 9, SN adapts its importance weight to IN while producing comparable stylization results. (b) plots the validation accuracy on CIFAR-10 when searching network architectures.

between standardization and whitening to improve a wide range of whitened neural networks [3, 21, 20, 9], which decorrelate the hidden representation [9, 21] or the back-propagated errors [20], enabling SGD to mimic natural gradient descent. SN could reduce the computations of the whitening networks, while maintaining fast convergence. Third, we will also study the selectivity of normalization operations in different places of more deep neural networks. We believe this study may help understand the impact of normalization methods in deep learning and shed light on new architecture design in future research.

#### Acknowledgement

The authors would like to thank Wenqi Shao for verifying the derivations of SN, as well as Xinjiang Wang, Lingyun Wu, Jingyu Li, and Ruimao Zhang for helpful discussions.

#### References

- [1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016. 1, 2, 3, 4, 6
- [2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 2, 4
- [3] G. Desjardins, K. Simonyan, R. Pascanu, and K. Kavukcuoglu. Natural neural networks. *NIPS*, 2015. 4, 9
- [4] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. <https://github.com/facebookresearch/detectron>, 2018. 7, 8
- [5] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677*, 2017. 5
- [6] K. He, G. Gkioxari, P. Dollr, and R. Girshick. Mask r-cnn. *ICCV*, 2017. 1, 7, 8
- [7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 5

- [8] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. 2017. 1
- [9] L. Huang, D. Yang, B. Lang, and J. Deng. Decorrelated batch normalization. *arXiv:1804.08450*, 2018. 9
- [10] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *arXiv:1703.06868*, 2017. 2, 8
- [11] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *arXiv:1702.03275*, 2017. 3, 4, 6
- [12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 1, 3, 4, 5, 7
- [13] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computing*, 1991. 4
- [14] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *arXiv:1603.08155*, 2016. 2, 4, 8, 9
- [15] A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical report*, 2009. 9
- [16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1
- [17] T.-Y. Lin, P. Dollra, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. *arXiv:1612.03144*, 2016. 7, 8
- [18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 4, 7, 9
- [19] M. Liwicki and H. Bunke. Iam-ondb - an on-line english sentence database acquired from handwritten text on a whiteboard. *8th Intl. Conf. on Document Analysis and Recognition*, 2015. 4
- [20] P. Luo. Eigennet: Towards fast and structural learning of deep neural networks. *IJCAI*, 2017. 4, 9
- [21] P. Luo. Learning deep architectures via generalized whitened neural networks. *ICML*, 2017. 4, 9
- [22] P. Luo, Y. Tian, X. Wang, and X. Tang. Switchable deep network for pedestrian detection. *CVPR*, 2014. 4
- [23] V. Nair and G. Hinton. Implicit mixtures of restricted boltzmann machines. *NIPS*, 2008. 4
- [24] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun. Megdet: A large mini-batch object detector. *arXiv:1711.07240*, 2017. 8
- [25] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv:1802.03268*, 2018. 1, 2, 4, 9
- [26] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv:1506.01497*, 2015. 7
- [27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 2, 4
- [28] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv:1602.07868*, 2016. 3, 4, 7
- [29] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, , and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ICLR*, 2017. 4
- [30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014. 8
- [31] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016. 1, 2, 3, 4, 6
- [32] G. Wang, J. Peng, P. Luo, X. Wang, and L. Lin. Batch kalman normalization: Towards training deep neural networks with micro-batches. *arXiv:1802.03133*, 2018. 3, 4, 6
- [33] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992. 9
- [34] Y. Wu and K. He. Group normalization. *arXiv:1803.08494*, 2018. 2, 4, 5, 7, 8
- [35] J. Yang, J. Lu, D. Batra, and D. Parikh. A faster pytorch implementation of faster r-cnn. <https://github.com/jwyang/faster-rcnn.pytorch>, 2017. 7

```

1  def SwitchNorm(x, gamma, beta, mean_weight, var_weight, eps):
2      # x: input features with shape [N,C,H,W]
3      # gamma, beta: scale and offset, with shape [1,C,1,1]
4      # mean_weight, var_weight: importance weights for means and variances of IN, LN, and BN in SN
5
6      mean_in, var_in = tf.nn.moments(x, [2, 3], keepdims=True)
7      mean_ln, var_ln = tf.nn.moments(x, [1, 2, 3], keepdims=True)
8      mean_bn, var_bn = tf.nn.moments(x, [0, 2, 3], keepdims=True)
9
10     mean = mean_weight[0] * mean_in + mean_weight[1] * mean_ln + mean_weight[2] * mean_bn
11     var = var_weight[0] * var_in + var_weight[1] * var_ln + var_weight[2] * var_bn
12
13     x = (x - mean) / tf.sqrt(var + eps)
14     return x * gamma + beta

```

Figure 7: A simple implementation of SN in TensorFlow is provided to help understand the rationale of SN. More details can be found in <https://github.com/switchablenorms/>.

## Appendix

### Comparisons of moving average and batch average.

In Fig.8, we compare SN with BN, where SN is evaluated with both moving average and batch average to estimate the statistics used in test. They are used to train ResNet50 on ImageNet. The two settings of SN produce similar results of 76.9% when converged, which is better than 76.4% of BN. We see that SN with batch average converges faster and more stably than BN and SN that use moving average. In this work, we find that for all batch settings, SN with batch average provides stable results better than those produced by using moving average.

**Code of SN.** A simple implementation of SN in TensorFlow is provided in Fig.7, which may help understand the rationale of SN. More sophisticated implementation can be found in <https://github.com/switchablenorms/>.

**Back-propagation of SN.** We provide the backward computations of SN as below. Let  $\hat{h}$  be the output of the SN layer represented by a 4D tensor  $(N, C, H, W)$  with index  $n, c, i, j$ . Let  $\hat{h} = \gamma \tilde{h} + \beta$  and  $\tilde{h} = \frac{h - \mu}{\sqrt{\sigma^2 + \epsilon}}$ , where  $\mu = w_{bn}\mu_{bn} + w_{in}\mu_{in} + w_{ln}\mu_{ln}$ ,  $\sigma^2 = w_{bn}\sigma_{bn}^2 + w_{in}\sigma_{in}^2 + w_{ln}\sigma_{ln}^2$ , and  $w_{bn} + w_{in} + w_{ln} = 1$ . Note that the importance weights are shared among the means and variances for clarity of notations. Suppose that each one of  $\{\mu, \mu_{bn}, \mu_{in}, \mu_{ln}, \sigma^2, \sigma_{bn}^2, \sigma_{in}^2, \sigma_{ln}^2\}$  is reshaped into a vector of  $N \times C$  entries, which are the same as the dimension of IN's statistics. Let  $\mathcal{L}$  be the loss function and  $(\frac{\partial \mathcal{L}}{\partial \mu})_n$  be the gradient with respect to the  $n$ -th entry of  $\mu$ .

We have

$$\frac{\partial \mathcal{L}}{\partial \tilde{h}_{ncij}} = \frac{\partial \mathcal{L}}{\partial \hat{h}_{ncij}} \cdot \gamma_c, \quad (6)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{1}{2(\sigma^2 + \epsilon)} \sum_{i,j} \frac{\partial \mathcal{L}}{\partial \tilde{h}_{ncij}} \cdot \tilde{h}_{ncij}, \quad (7)$$

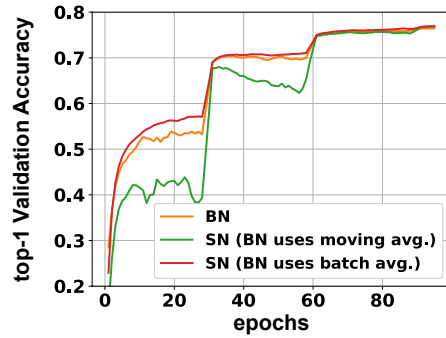


Figure 8: Comparisons of BN, SN with moving average, and SN with batch average, when training ResNet50 on ImageNet in the batch setting of (8,32). We see that SN with batch average produces faster and more stable convergence than the other methods.

$$\frac{\partial \mathcal{L}}{\partial \mu} = -\frac{1}{\sqrt{\sigma^2 + \epsilon}} \sum_{i,j} \frac{\partial \mathcal{L}}{\partial \tilde{h}_{ncij}}, \quad (8)$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial h_{ncij}} &= \frac{\partial \mathcal{L}}{\partial \tilde{h}_{ncij}} \cdot \frac{1}{\sqrt{\sigma^2 + \epsilon}} + \left[ \frac{2w_{in}(h_{ncij} - \mu_{in})}{HW} \frac{\partial \mathcal{L}}{\partial \sigma^2} \right. \\ &+ \frac{2w_{ln}(h_{ncij} - \mu_{ln})}{CHW} \sum_{c=1}^C \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_c + \frac{2w_{bn}(h_{ncij} - \mu_{bn})}{NHW} \sum_{n=1}^N \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_n \Big] \\ &+ \left[ \frac{w_{in}}{HW} \frac{\partial \mathcal{L}}{\partial \mu} + \frac{w_{ln}}{CHW} \sum_{c=1}^C \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_c + \frac{w_{bn}}{NHW} \sum_{n=1}^N \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_n \right], \end{aligned} \quad (9)$$

The gradients for  $\gamma$  and  $\beta$  are

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{n,i,j} \frac{\partial \mathcal{L}}{\partial \hat{h}_{ncij}} \cdot \tilde{h}_{ncij}, \quad (10)$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = \sum_{n,i,j} \frac{\partial \mathcal{L}}{\partial \hat{h}_{ncij}}, \quad (11)$$

and the gradients for  $\lambda_{\text{in}}$ ,  $\lambda_{\text{ln}}$ , and  $\lambda_{\text{bn}}$  are

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \lambda_{\text{in}}} = & w_{\text{in}}(1 - w_{\text{in}}) \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{in}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{in}}^2 \right) \\ & - w_{\text{in}} w_{\text{ln}} \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{ln}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{ln}}^2 \right) \\ & - w_{\text{in}} w_{\text{bn}} \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{bn}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{bn}}^2 \right). \quad (2) \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \lambda_{\text{ln}}} = & w_{\text{ln}}(1 - w_{\text{ln}}) \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{ln}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{ln}}^2 \right) \\ & - w_{\text{in}} w_{\text{ln}} \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{in}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{in}}^2 \right) \\ & - w_{\text{ln}} w_{\text{bn}} \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{bn}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{bn}}^2 \right). \quad (3) \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \lambda_{\text{bn}}} = & w_{\text{bn}}(1 - w_{\text{bn}}) \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{bn}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{bn}}^2 \right) \\ & - w_{\text{in}} w_{\text{bn}} \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{in}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{in}}^2 \right) \\ & - w_{\text{ln}} w_{\text{bn}} \sum_{n,c}^{N,C} \left( \left( \frac{\partial \mathcal{L}}{\partial \mu} \right)_{nc} \mu_{\text{ln}} + \left( \frac{\partial \mathcal{L}}{\partial \sigma^2} \right)_{nc} \sigma_{\text{ln}}^2 \right). \quad (14) \end{aligned}$$



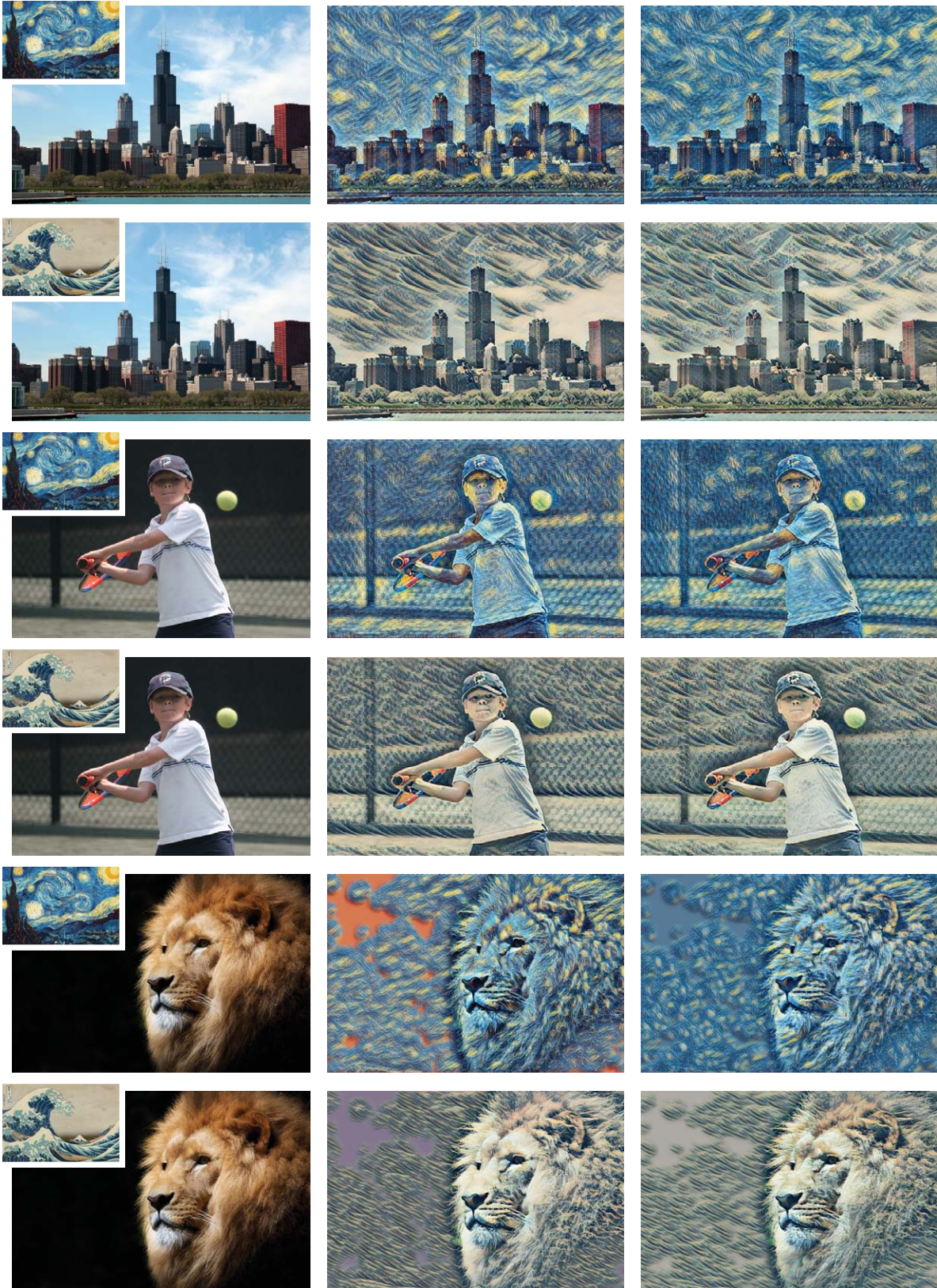


Figure 9: **Results of Image Stylization.** The first column visualizes the content and the style images. The second and third columns are the results of IN and SN respectively. SN works comparably well with IN in this task, as it adaptively selects IN as shown by the importance weights in Fig.1 (c).