Weight Standardization

Siyuan Qiao Huiyu Wang Chenxi Liu Wei Shen Alan Yuille
Johns Hopkins University
{siyuan.qiao, hwang157, cxliu}@jhu.edu {shenwei1231, alan.l.yuille}@gmail.com

Abstract

In this paper, we propose Weight Standardization (WS) to accelerate deep network training. WS is targeted at the micro-batch training setting where each GPU typically has only 1-2 images for training. The micro-batch training setting is hard because small batch sizes are not enough for training networks with Batch Normalization (BN), while other normalization methods that do not rely on batch knowledge still have difficulty matching the performances of BN in large-batch training. Our WS ends this problem because when used with Group Normalization and trained with 1 image/GPU, WS is able to match or outperform the performances of BN trained with large batch sizes with only **2 more lines of code**. *In micro-batch training, WS significantly outperforms other normalization methods. WS achieves these superior results by standardizing the weights in the convolutional layers, which we show is able to smooth the loss landscape by reducing the Lipschitz constants of the loss and the gradients. The effectiveness of WS is verified on many tasks, including image classification, object detection, instance segmentation, video recognition, semantic segmentation, and point cloud recognition. The code is available here:* https://github.com/joe-siyuan-qiao/WeightStandardization.

1. Introduction

Deep learning has advanced the state-of-the-arts in many vision tasks [4, 12]. Many deep networks use Batch Normalization (BN) [17] in their architectures because BN in most cases is able to accelerate training and help the models to converge to better solutions. BN stabilizes the training by controlling the first two moments of the distributions of the layer inputs in each mini-batch during training and is especially helpful for training very deep networks that have hundreds of layers [13, 14]. Despite its practical success, BN has several shortcomings that draw a lot of attentions from researchers. For example, (1) we lack good understandings of the reasons of BN's success, and (2) BN works well only when the batch size is sufficiently large, which prohibits it from being used in micro-batch training. We argue that these two drawbacks are related: a good understanding of BN may lead us to other normalization techniques that train deep networks faster without relying on any batch knowledge, hence micro-batch training can be possible. Although some normalization methods are specifically designed for micro-batch training such as Group Normalization (GN) [42], they still have difficulty matching the performances of BN in large-batch training (Fig. 1).

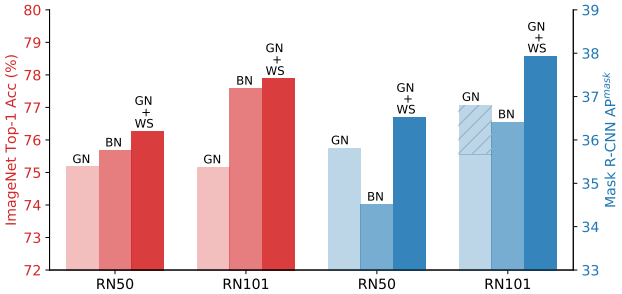

Figure 1. Comparing BN [17], GN [42], and our WS used with GN on ImageNet and COCO. On ImageNet, BN is trained with large batch sizes while GN and GN+WS are trained with **1 image/GPU**. On COCO, BN is frozen for micro-batch training. The shaded area is the difference between the reimplemented and the reported APs. Despite that the GN implementation we use has worse performances on COCO (mainly due to hardware constraints), GN+WS still outperforms both BN and GN comfortably.

The widely accepted explanation was related to internal covariate shift until [37] shows that the performance gain of BN has little to do with the reduction of internal covariate shift. Instead, [37] proves that BN makes the landscape of the corresponding optimization problem significantly smoother. We follow this idea and aim to propose another normalization technique that further smooths the landscape. Our goal is to accelerate the training of deep networks like BN but without relying on large batch sizes during training.

In this paper, we propose Weight Standardization (WS), which smooths the loss landscape by standardizing the weights in convolutional layers. Different from the previous normalization methods that focus on *activations* [3, 17, 40,

42], WS considers the smoothing effects of *weights* more than just length-direction decoupling [36]. To show its effectiveness, we study WS from both theoretical and experimental viewpoints. The highlights of our contributions are:

1. Theoretically, we prove that WS reduces the Lipschitz constants of the loss and the gradients. Hence, WS smooths the loss landscape and improves training.
2. Experiments show that on tasks where large-batch training is available (*e.g.* ImageNet [35]), GN [42] + WS with batch size 1 is able to match or outperform the performances of BN with large batch sizes (Fig. 1).
3. For tasks where only micro-batch training is available (*e.g.* COCO [22]), GN + WS will significantly improve the performances (Fig. 1).

To show that our WS is applicable to many vision tasks, we conduct comprehensive experiments, including image classification on ImageNet dataset [35], object detection and instance segmentation on COCO dataset [22], video recognition on Something-SomethingV1 dataset [8], semantic image segmentation on PASCAL VOC [6], and point cloud classification on ModelNet40 dataset [43]. The results show that our WS is able to accelerate training and improve performances on all of them.

2. Weight Standardization

It has been demonstrated that BN influences network training in a fundamental way: it makes the landscape of the optimization problem significantly smoother [37]. Specifically, [37] shows that BN reduces the Lipschitz constants of the loss function, and makes the gradients more Lipschitz, too, *i.e.*, the loss will have a better β-smoothness [29]. These results are done on the activations, which BN standardizes to have zero mean and unit variance.

We notice that BN considers the Lipschitz constants with respect to *activations*, not the *weights* that the optimizer is directly optimizing. Therefore, we argue that we can also standardize the *weights* in the convolutional layers to further smooth the landscape. By doing so, we do not have to worry about transferring smoothing effects from activations to weights, and the smoothing effects on activations and weights are also additive. Based on these motivations, we propose Weight Standardization.

2.1. Weight Standardization

Here, we show the detailed modeling of Weight Standardization (WS) (Fig. 2). Consider a standard convolutional layer with its bias term set to 0:

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x} \tag{1}$$

where $\hat{\boldsymbol{W}} \in \mathbb{R}^{O \times I}$ denotes the weights in the layer and $*$ denotes the convolution operation. For $\hat{\boldsymbol{W}} \in \mathbb{R}^{O \times I}$, O is the number of the output channels, I corresponds to the number of input channels within the kernel region of each output channel. Taking Fig. 2 as an example, $O = C_{\text{out}}$ and $I = C_{\text{in}} \times \text{Kernel\_Size}$. In Weight Standardization, instead of directly optimizing the loss $\mathcal{L}$ on the original weights $\hat{\boldsymbol{W}}$, we reparameterize the weights $\hat{\boldsymbol{W}}$ as a function of $\boldsymbol{W}$, *i.e.*, $\hat{\boldsymbol{W}} = \text{WS}(\boldsymbol{W})$, and optimize the loss $\mathcal{L}$ on $\boldsymbol{W}$ by SGD:

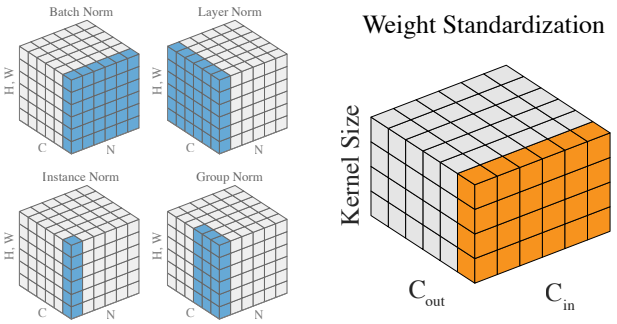

Figure 2. Comparing normalization methods on activations (blue) and Weight Standardization (orange).

$$\hat{\boldsymbol{W}} = \left[\hat{\boldsymbol{W}}_{i,j} \,\middle|\, \hat{\boldsymbol{W}}_{i,j} = \frac{\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}}}{\sigma_{\boldsymbol{W}_{i,\cdot}} + \epsilon} \right] \tag{2}$$

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x} \tag{3}$$

where

$$\mu_{\boldsymbol{W}_{i,\cdot}} = \frac{1}{I} \sum_{j=1}^{I} \boldsymbol{W}_{i,j}, \quad \sigma_{\boldsymbol{W}_{i,\cdot}} = \sqrt{\frac{1}{I} \sum_{i=1}^{I} (\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}})^2} \tag{4}$$

Similar to BN, WS controls the first and second moments of the weights of each output channel individually in convolutional layers. Note that many initialization methods also initialize the weights in some similar ways. Different from those methods, WS standardizes the weights in a differentiable way which aims to normalize gradients during back-propagation. Note that we do not have any affine transformation on $\hat{\boldsymbol{W}}$. This is because we assume that normalization layers such as BN or GN will normalize this convolutional layer again, and having affine transformation will harm training as we will show in the experiments. In the following, we first discuss the normalization effects of WS to the gradients.

2.2. WS normalizes gradients

For convenience, we set $\epsilon = 0$ (in Eq. 2). We first focus on one output channel c. Let $\boldsymbol{y}_c \in \mathbb{R}^b$ be all the outputs of channel c during one pass of feedforwarding and back-propagation, and $\boldsymbol{x}_c \in \mathbb{R}^{b,I}$ be the corresponding inputs.

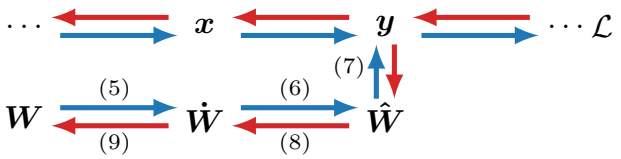

Figure 3. Computation graph for WS in feed-forwarding and back-propagation. The numbers are equation numbers.

Then, we can rewrite Eq. 2 and 3 as

$$\dot{\boldsymbol{W}}_{c,\cdot} = \boldsymbol{W}_{c,\cdot} - \frac{1}{I}\mathbf{1}\langle \mathbf{1}, \boldsymbol{W}_{c,\cdot}\rangle \tag{5}$$

$$\hat{\boldsymbol{W}}_{c,\cdot} = \dot{\boldsymbol{W}}_{c,\cdot} / \left(\sqrt{\frac{1}{I}\langle \mathbf{1}, \dot{\boldsymbol{W}}^{\circ 2}_{c,\cdot}\rangle}\right) \tag{6}$$

$$\boldsymbol{y}_c = \boldsymbol{x}_c \hat{\boldsymbol{W}}_{c,\cdot} \tag{7}$$

where $\langle\ ,\ \rangle$ denotes dot product and $^{\circ 2}$ denotes Hadamard power. Then, the gradients are

$$\nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L} = \frac{1}{\sigma_{\boldsymbol{W}_{c,\cdot}}} \left(\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L} - \frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle \hat{\boldsymbol{W}}_{c,\cdot}\right) \tag{8}$$

$$\nabla_{\boldsymbol{W}_{c,\cdot}} \mathcal{L} = \nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L} - \frac{1}{I}\mathbf{1}\langle \mathbf{1}, \nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle \tag{9}$$

Fig. 3 shows the computation graph. Based on the equations, we observe that different from the original gradients $\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}$ which is back-propagated through Eq. 7, the gradients are normalized by Eq. 8 & 9.

In Eq. 8, to compute $\nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}$, $\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}$ is first subtracted by a weighted average of $\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}$ and then divided by $\sigma_{\hat{\boldsymbol{W}}_{c,\cdot}}$. Note that when BN is used to normalize this convolutional layer, as BN will compute again the scaling factor $\sigma_{\boldsymbol{u}}$, the effects of dividing the gradients by $\sigma_{\hat{\boldsymbol{W}}_{c,\cdot}}$ will be canceled in both feedforwarding and back-propagation. As for the additive term, its effect will depend on the statistics of $\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}$ and $\hat{\boldsymbol{W}}_{c,\cdot}$. We will later show that this term will reduce the gradient norm regardless of the statistics. As for Eq. 9, it zero-centers the gradients from $\dot{\boldsymbol{W}}_{c,\cdot}$. When the mean gradient is large, zero-centering will significantly affect the gradients passed to $\boldsymbol{W}_{c,\cdot}$.

2.3. WS smooths landscape

We will show that WS is able to make the loss landscape smoother. Specifically, we show that optimizing $\mathcal{L}$ on $\boldsymbol{W}$ has smaller Lipschitz constants on both the loss and the gradients than optimizing $\mathcal{L}$ on $\hat{\boldsymbol{W}}$. Lipschitz constant of a function f is the value of L if f satisfies $|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|$, $\forall x_1, x_2$. For the loss and gradients, f will be $\mathcal{L}$ and $\nabla_{\boldsymbol{W}}\mathcal{L}$, and x will be $\boldsymbol{W}$. Smaller Lipschitz constants on the loss and gradients mean that the changes of the loss and the gradients during training will be bounded more. They will provide more confidence when the optimizer takes a big step in the gradient direction as the gradient direction will vary less within the range of the step. In other words, the optimizer can take longer steps without worrying about sudden changes of the loss landscape and gradients. Therefore, WS is able to accelerate training.

Effects of WS on the Lipschitz constant of the loss Here, we show that both Eq. 8 and Eq. 9 are able to reduce the Lipschitz constant of the loss. We first study Eq. 8:

$$\left\|\nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\right\|^2 = \frac{1}{\sigma^2_{\boldsymbol{W}_{c,\cdot}}} \Big(\left\|\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\right\|^2 + \frac{1}{I^2}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2 \left(\langle \hat{\boldsymbol{W}}_{c,\cdot}, \hat{\boldsymbol{W}}_{c,\cdot}\rangle - 2I\right)\Big) \tag{10}$$

By Eq. 6, we know that $\|\hat{\boldsymbol{W}}_{c,\cdot}\|^2 = I$. Then,

$$\left\|\nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\right\|^2 = \frac{1}{\sigma^2_{\boldsymbol{W}_{c,\cdot}}} \Big(\left\|\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\right\|^2 - \tag{11}$$

$$\frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2\Big) \tag{12}$$

Since we assume that this convolutional layer is followed by a normalization layer such as BN or GN, the effect of $1/\sigma^2_{\boldsymbol{W}_{c,\cdot}}$ will be canceled. Therefore, the real effect on the gradient norm is the reduction $\frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2$, which reduces the Lipschitz constant of the loss.

Next, we study the effect of Eq. 9. By definition,

$$\left\|\nabla_{\boldsymbol{W}_{c,\cdot}} \mathcal{L}\right\|^2 = \left\|\nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\right\|^2 - \frac{1}{I}\langle \mathbf{1}, \nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2 \tag{13}$$

By Eq. 8, we rewrite the second term:

$$\frac{1}{I}\langle \mathbf{1}, \nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2 = \frac{1}{I \cdot \sigma^2_{\boldsymbol{W}_{c,\cdot}}} \Big(\langle \mathbf{1}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle - \frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle \cdot \langle \mathbf{1},\ \hat{\boldsymbol{W}}_{c,\cdot}\rangle\Big)^2 \tag{14}$$

Since $\langle \mathbf{1},\ \hat{\boldsymbol{W}}_{c,\cdot}\rangle = 0$, we have

$$\left\|\nabla_{\boldsymbol{W}_{c,\cdot}} \mathcal{L}\right\|^2 = \left\|\nabla_{\dot{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\right\|^2 - \frac{1}{I \cdot \sigma^2_{\boldsymbol{W}_{c,\cdot}}}\langle \mathbf{1}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2 \tag{15}$$

Summarizing the effects of Eq. 8 and 9 on the Lipschitz constant of the loss: ignoring $1/\sigma^2_{\boldsymbol{W}_{c,\cdot}}$, Eq. 8 reduces it by $\frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2$, and Eq. 9 reduces $\frac{1}{I}\langle \mathbf{1}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}} \mathcal{L}\rangle^2$.

Although both Eq. 8 and 9 reduce the Lipschitz constant, their real effects depend the statistics of the weights and the gradients. For example, the reduction effect of Eq. 9 depends on the average gradients on $\hat{\boldsymbol{W}}$. As for Eq. 8, note that $\langle \mathbf{1},\ \hat{\boldsymbol{W}}_{c,\cdot}\rangle = 0$, its effect might be limited when $\hat{\boldsymbol{W}}_{c,\cdot}$ is evenly distributed around 0. To understand their real effects, we conduct a case study on ResNet-50 trained on ImageNet to see which one of Eq. 5 and 6 has bigger effects or they contribute similarly to smoothing the landscape.

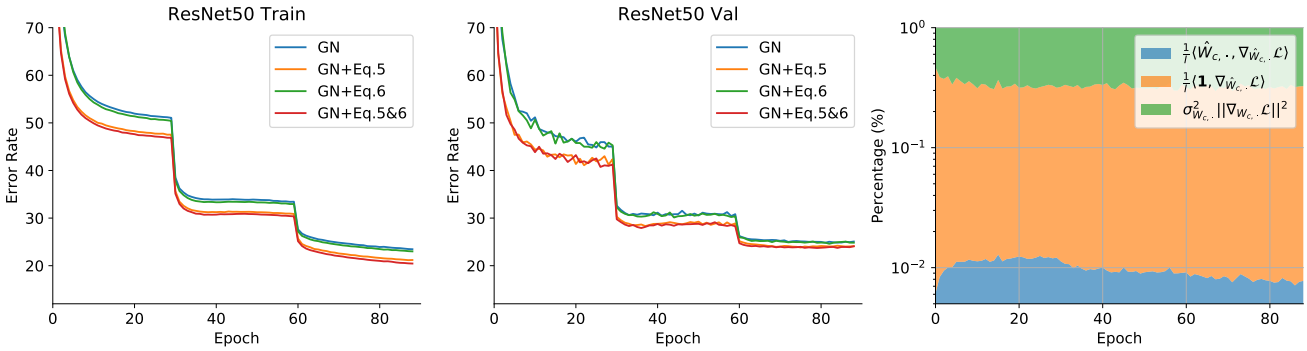

Figure 4. Training ResNet-50 on ImageNet with GN, Eq. 5 and 6. The left and the middle figures show the training dynamics. The right figure shows the reduction percentages on the Lipschitz constant. Note that the y-axis of the right figure is in **log** scale.

Case Study on ResNet-50 Before the Lipschitzness study on the gradients, we first show a case study where we train ResNet-50 models on ImageNet following the conventional training procedure [12]. In total, we train four models, including ResNet-50 with GN, ResNet-50 with GN+Eq. 5, ResNet-50 with GN+Eq. 6 and ResNet-50 with GN+Eq. 5&6. The training dynamics are shown in Fig. 4, from which we observe that Eq. 6 slightly improves the training speed and performances of models with or without Eq. 5 while the major improvements are from Eq. 5. This observation motivates us to study the real effects of Eq. 5 and 6. To investigate this, we take a look at the values of $\frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}\rangle^2$, and $\frac{1}{I}\langle \mathbf{1}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}\rangle^2$ during training.

To compute the two values above, we gather and save the intermediate gradients $\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}$, and the weights for the convolution $\hat{\boldsymbol{W}}_{c,\cdot}$. In total, we train ResNet-50 with GN, Eq. 5 and 6 for 90 epochs, and we save the gradients and the weights of the first training iteration of each epoch. The right figure of Fig. 4 shows the average percentages of $\frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}\rangle^2$, $\frac{1}{I}\langle \mathbf{1}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}\rangle^2$, and $\sigma^2_{\boldsymbol{W}_{c,\cdot}}\|\nabla_{\boldsymbol{W}_{c,\cdot}}\mathcal{L}\|^2$. From the right figure we can see that $\frac{1}{I}\langle \hat{\boldsymbol{W}}_{c,\cdot}, \nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}\rangle^2$ is small compared with other two components (< 0.02). In other words, although Eq. 6 decreases the gradient norm regardless of the statistics of the weights and gradients, its real effect is limited due to the distribution of $\hat{\boldsymbol{W}}_{c,\cdot}$ and $\nabla_{\hat{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}$. Nevertheless, from the left figures we can see that Eq. 6 still improves the training. Since Eq. 6 requires very little computations, we will keep it in WS.

Effects of WS on the Lipschitz constant of the gradients From the experiments above, we observe that the training speed boost is mainly due to Eq. 5. As the effect of Eq. 6 is limited, in this section, we only study the effect of Eq. 5 on the Hessian of $\boldsymbol{W}_{c,\cdot}$ and $\dot{\boldsymbol{W}}_{c,\cdot}$. Here, we will show that Eq. 5 decreases the Frobenius norm of the Hessian matrix of the weights, *i.e.*, $\|\nabla^2_{\boldsymbol{W}_{c,\cdot}}\mathcal{L}\|_F \leq \|\nabla^2_{\dot{\boldsymbol{W}}_{c,\cdot}}\mathcal{L}\|_F$. With smaller Frobenius norm, the gradients of $\dot{\boldsymbol{W}}_{c,\cdot}$ are more predictable, thus the loss is smoother and easier to optimize.

We use $\boldsymbol{H}$ and $\dot{\boldsymbol{H}}$ to denote the Hessian matrices of $\boldsymbol{W}_{c,\cdot}$ and $\dot{\boldsymbol{W}}_{c,\cdot}$, respectively, *i.e.*,

$$\boldsymbol{H}_{i,j} = \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{W}_{c,i}\partial \boldsymbol{W}_{c,j}}, \quad \dot{\boldsymbol{H}}_{i,j} = \frac{\partial^2 \mathcal{L}}{\partial \dot{\boldsymbol{W}}_{c,i}\partial \dot{\boldsymbol{W}}_{c,j}} \tag{16}$$

We first derive the relationship between $\boldsymbol{H}_{i,j}$ and $\dot{\boldsymbol{H}}_{i,j}$:

$$\boldsymbol{H}_{i,j} = \dot{\boldsymbol{H}}_{i,j} - \frac{1}{I}\sum_{k=1}^{I}(\dot{\boldsymbol{H}}_{i,k} + \dot{\boldsymbol{H}}_{k,j}) + \frac{1}{I^2}\sum_{p=1}^{I}\sum_{q=1}^{I}\dot{\boldsymbol{H}}_{p,q} \tag{17}$$

Note that

$$\sum_{i=1}^{I}\sum_{j=1}^{I}\boldsymbol{H}_{i,j} = 0 \tag{18}$$

Therefore, Eq. 5 not only zero-centers the feedforwarding outputs and the back-propagated gradients, but also the Hessian matrix. Next, we compute its Frobenius norm:

$$\begin{aligned}
\|\boldsymbol{H}\|_F =& \sum_{i=1}^{I}\sum_{j=1}^{I}\boldsymbol{H}_{i,j}^2 \\
=& \|\dot{\boldsymbol{H}}\|_F + \frac{1}{I^2}\Big(\sum_{i=1}^{I}\sum_{j=1}^{I}\dot{\boldsymbol{H}}_{i,j}\Big)^2 \\
&- \frac{1}{I}\sum_{i=1}^{I}\Big(\sum_{j=1}^{I}\dot{\boldsymbol{H}}_{i,j}\Big)^2 - \frac{1}{I}\sum_{j=1}^{I}\Big(\sum_{i=1}^{I}\dot{\boldsymbol{H}}_{i,j}\Big)^2 \\
\leq& \|\dot{\boldsymbol{H}}\|_F - \frac{1}{I^2}\Big(\sum_{i=1}^{I}\sum_{j=1}^{I}\dot{\boldsymbol{H}}_{i,j}\Big)^2
\end{aligned} \tag{19}$$

As shown in Eq. 19, Eq. 5 reduces the Frobenius norm of the Hessian matrix by at least $\big(\sum_{i=1}^{I}\sum_{j=1}^{I}\dot{\boldsymbol{H}}_{i,j}\big)^2/I^2$, which makes the gradients more predictable than directly optimizing on the weights of the convolutional layer.

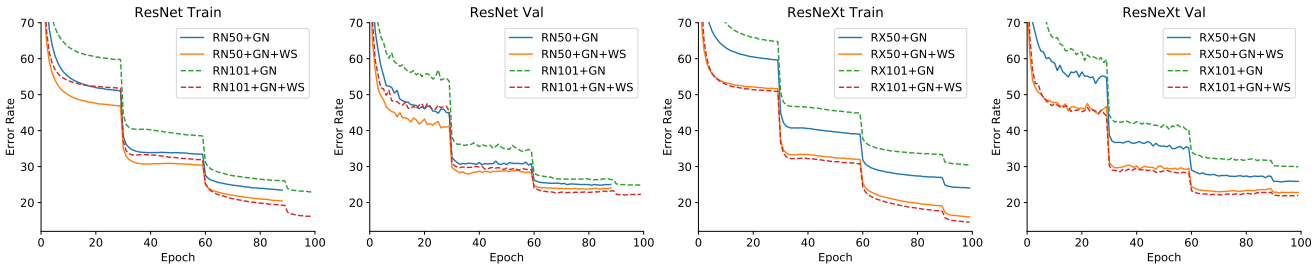

Figure 5. Training and validation error rates of ResNet and ResNeXt on ImageNet. The comparison is between GN baseline [42] and GN + WS. Our method WS not only significantly improves the training speed, it also lowers the error rates of the final models by a large margin.

Method – Batch Size	BN [17] – 64 / 32		SN [26] – 1		GN [42] – 1		BN+WS – 64 / 32		GN+WS – 1	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
ResNet-50 [12]	24.30	7.19	25.00	–	24.81	7.46	23.76	7.13	23.72	6.99
ResNet-101 [12]	22.44	6.21	–	–	24.82	7.56	21.89	6.01	22.10	6.07

Table 1. Error rates of ResNet-50 and ResNet-101 on ImageNet. ResNet-50 models with BN are trained with batch size 64 per GPU, and ResNet-101 models with BN are trained with 32 images per GPU. Other normalization methods are trained with 1 image per GPU.

Method	Top-1	Method	Top-1
LN [3]	27.22	LN+WS	24.60
IN [40]	29.49	IN+WS	28.24
GN [42]	24.81	GN+WS	23.72
BN [17]	24.30	BN+WS	23.76
GN+WN [36]	25.09	GN+WS+AF	29.43
GN+WS+MO	23.96	GN+WS+DO	24.60

Table 2. Top-1 error rates of ResNet-50 on ImageNet with different normalization methods. GN+WS+AF: GN+WS with affine transformation on weights. GN+WS+MO: GN+WS with Eq. 5. GN+WS+DO: GN+WS with Eq. 6. All models except BN are trained with batch size 1 per GPU. BN models are trained with batch size 64 per GPU.

Method	GN [42]		GN+WS [42]	
Batch Size = 1	Top-1	Top-5	Top-1	Top-5
ResNeXt-50 [44]	25.73	8.13	22.71	6.38
ResNeXt-101 [44]	29.84	10.77	21.80	6.03

Table 3. ResNeXt-50 and ResNeXt-101 on ImageNet. All models are trained with batch size 1 per GPU.

3. Experimental Results

In this section, we will present the experimental results, including image classification on ImageNet [35], object detection and instance segmentation on COCO [22], video recognition on Something-SomethingV1 dataset [8], semantic segmentation on PASCAL VOC [6], and point cloud classification on ModelNet40 [43].

3.1. Image Classification on ImageNet

ImageNet dataset is a large-scale image classification dataset. There are about 1.28 million training samples and 50K validation images. It has 1000 categories, each has roughly 1300 training images and exactly 50 validation samples. Table 2 shows the top-1 error rates of ResNet-50 on ImageNet when it is trained with different normalization methods, including Layer Normalization [3], Instance Normalization [40], Group Normalization [42] and Batch Normalization. From Table 2, we can see that when the batch size is limited to 1, GN+WS is able to achieve performances comparable to BN with large batch size. Therefore, we will use GN+WS for micro-batch training because GN shows the best results among all the normalization methods that can be trained with 1 image per GPU. Table 2 also shows WS with affine transformation, which harms the training.

Table 1 shows our major experimental results on the ImageNet dataset [35]. Note that Table 1 only shows the error rates of ResNet-50 and ResNet-101. This is to compare with the previous work that focus on micro-batch training problem, e.g. Switchable Normalization [26] and Group Normalization [42]. We run all the experiments using the official PyTorch implementations of the layers except for SN [26] which are the performances reported in their paper. This makes sure that all the experimental results are comparable, and our improvements are reproducible.

In Table 3, we also provide the experimental results on ResNeXt [44], and the comparisons of the training curves of ResNet and ResNeXt trained with different normalization methods are shown in Fig. 5. Here, we show the performance comparisons between ResNeXt+GN and ResNeXt+GN+WS. Note that GN originally did not provide results on ResNeXt. Without tuning the hyper-parameters in the Group Normalization layers, we use 32 groups for

Method	Backbone	AP^b	$AP^b_{.5}$	$AP^b_{.75}$	AP^b_l	AP^b_m	AP^b_s	AP^m	$AP^m_{.5}$	$AP^m_{.75}$	AP^m_l	AP^m_m	AP^m_s
GN	ResNet-50	39.86	60.55	43.60	51.78	42.53	24.48	35.80	57.66	38.06	52.71	38.18	17.54
SN	ResNet-50	40.43	61.06	44.22	52.34	43.71	24.09	35.71	57.80	37.61	52.16	38.81	16.27
GN+WS	ResNet-50	40.81	61.59	44.78	52.69	43.94	23.48	36.51	58.53	38.93	53.48	39.33	16.64
GN	ResNet-101	39.68	60.35	43.01	51.45	42.75	22.63	35.66	57.36	37.95	52.51	38.02	16.21
GN+WS	ResNet-101	42.73	63.62	46.78	55.94	46.07	25.67	37.92	60.35	40.72	56.28	40.64	18.16
GN	ResNeXt-50	39.49	60.38	42.89	51.50	42.08	23.86	35.41	57.07	38.02	52.25	37.84	17.09
GN+WS	ResNeXt-50	42.00	63.01	45.65	54.10	45.25	25.69	37.28	59.93	39.91	54.35	40.01	18.76
GN	ResNeXt-101	36.36	56.78	39.01	48.38	38.67	20.84	32.68	53.74	34.07	49.48	34.10	14.95
GN+WS	ResNeXt-101	43.12	64.15	47.11	56.39	47.19	25.49	38.34	61.07	40.82	56.08	41.73	18.32

Table 4. Detection and segmentation results on COCO [22] using Mask R-CNN [10] and FPN [23] with ResNet [12] and ResNeXt (32x4d) [44] as backbone. The normalization methods, *i.e.* GN [42], SN [26], and GN+WS, are used in both the backbone and the heads.

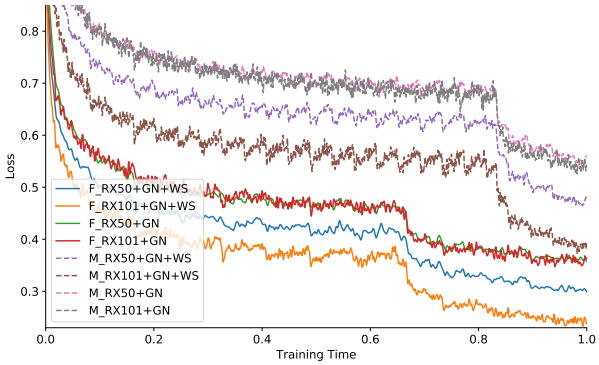

Figure 6. The training loss of Faster R-CNN (F_*) and Mask R-CNN (M_*) with different backbones and normalization methods.

Method	Backbone	AP	$AP_{.5}$	$AP_{.75}$	AP_l	AP_m	AP_s
GN	RN-50	37.98	59.05	41.27	49.47	40.87	22.42
SN	RN-50	38.04	60.68	40.87	48.41	41.50	23.46
GN+WS	RN-50	39.36	60.77	42.56	51.09	42.31	23.57
GN	RN-101	37.59	58.49	40.80	49.02	40.65	21.53
GN+WS	RN-101	41.32	63.07	45.07	53.17	45.39	24.82
GN	RX-50	36.62	57.67	39.57	47.57	39.03	21.73
GN+WS	RX-50	40.25	61.97	43.54	52.18	43.56	24.06
GN	RX-101	33.28	53.92	35.47	43.98	35.67	18.63
GN+WS	RX-101	41.90	63.85	45.38	53.94	45.52	25.51

Table 5. Detection results on COCO using Faster R-CNN [34] and FPN with GN, SN and GN+WS as normalization methods.

each of them which is the default configuration for ResNet that GN was originally proposed for. ResNeXt-50 and 101 are 32x4d. We train the models for 100 epochs with batch size set to 1 and iteration size set to 32. As the table shows, the performance of GN on training ResNeXt is unsatisfactory: they perform even worse than the original ResNets. Especially for ResNeXt-101, the performance difference is large between GN and BN. In the same setting, WS is able to make the training of ResNeXt a lot easier. This enables ResNeXt to train with GN in Mask R-CNN [10] and Faster R-CNN [34], which we will discuss in the next subsection.

Here, we list the hyper-parameters used for getting all those results. For all models, the learning rate is set to 0.1 initially, and is multiplied by 0.1 after every 30 epochs. We use SGD to train the models, where the weight decay is set to 0.0001 and the momentum is set to 0.9. For ResNet-50 with BN or BN+WS, the training batch is set to 256 for 4 GPUs. Without synchronized BN [30], the effective batch size is 64. For other ResNet-50 where batch size is 1 per GPU, we set the iteration size to 64, *i.e.*, the gradients are averaged across every 64 iterations and then one step is taken. This is to ensure fair comparisons because by doing so the total numbers of parameter updates are the same even if their batch sizes are different. We train ResNet-50 with different normalization techniques for 90 epochs. For ResNet-101, we set the batch size to 128 because some of the models will use more than 12GB per GPU when setting their batch size to 256. In total, we train all ResNet-101 models for 100 epochs. Similarly, we set the iteration size for models trained with 1 image per GPU to be 32 in order to compensate for the total numbers of parameter updates.

3.2. Object Detection and Segmentation on COCO

Unlike image classification on ImageNet where we could afford large batch training when the models are not too big, object detection and segmentation on COCO [22] usually use 1 or 2 images per GPU for training due to the high resolution. Given the good performances of our method on ImageNet which are comparable to the large-batch BN training, we expect that our method is able to significantly improve the performances on COCO because of the training setting.

We use a PyTorch-based Mask R-CNN framework[1] for all the experiments except for SN [26] where we use their official PyTorch implementations. We take the models pre-trained on ImageNet, fine-tune them on COCO train2017

[1] https://github.com/facebookresearch/maskrcnn-benchmark

Method	Backbone	#Frame	Top-1	Top-5
GN	ResNet-50	8	42.07	73.20
GN+WS	ResNet-50	8	44.26	75.51
BN	ResNet-50	8	44.30	74.53
BN+WS	ResNet-50	8	46.49	76.46

Table 6. Comparing video recognition accuracy of TSM [21] with different normalization methods on Something-SomethingV1 [8].

Method	Backbone	Mean IOU
GN	ResNet-101	74.90
GN+WS	ResNet-101	77.20
BN	ResNet-101	76.49
BN+WS	ResNet-101	77.15

Table 7. Semantic segmentation performance of DeepLabv3 [5] on PASCAL VOC 2012 validation set. Output stride is 16, without multi-scale or flipping when testing.

set, and test them on COCO val2017 set. To maximize the comparison fairness, we use the models we pre-trained on ImageNet instead of downloading the pre-trained models available online except for SN for which we use the downloaded models SN(8,1) per the instructions on their official GitHub website. We use 4 GPUs to train the models and apply the learning rate schedules for all models following the practice used in the Mask R-CNN framework our work is based on. We use 1X learning rate schedule for Faster R-CNN and 2X learning rate schedule for Mask R-CNN. For ResNet-50 and ResNeXt-50, we use 2 images per GPU to train the models, and for ResNet-101 and ResNeXt-101 we use 1 image per GPU because the models cannot fit in 12GB GPU memory. We then adapt the learning rates and the training steps accordingly. The configurations we run use FPN [23] and a 4conv1fc bounding box head. All the training procedures strictly follow their original settings.

Table 5 reports the Average Precision (AP) of Faster R-CNN trained with different methods and Table 4 reports the Average Precision for bounding box (AP^b) and instance segmentation (AP^m). From the two tables, we can observe results similar to those on ImageNet. GN has limited performance improvements when it is used on more complicated architectures such as ResNet-101, ResNeXt-50 and ResNeXt-101. But when we add our method to GN, we are able to train the models much better. The improvements become more significant when the network complexity increases. Considering nowadays deep networks are becoming deeper and wider, having a normalization technique such as our WS will ease the training a lot without worrying about the memory and batch size issues.

3.3. Video Recognition on Something-Something

In this subsection, we show the results of applying our method on video recognition on Something-SomethingV1 dataset [8]. Something-SomethingV1 is a video dataset which includes a large number of video clips that show humans performing pre-defined basic actions. The dataset has 86,017 clips for training and 11,522 clips for validation.

We use the state-of-the-art method TSM [21] for video recognition, which uses a ResNet-50 with BN as its backbone network. The codes are based on TRN [47] and then adapted to TSM. The reimplementation is different from the original TSM [21]: we use models pre-trained on ImageNet rather than Kinetics dataset [18] as the starting points. Then, we fine-tune the pre-trained models on Something-SomethingV1 for 45 epochs. The batch size is set to 32 for 4 GPUs, and the learning rate is initially set to 0.0125, then divided by 10 at the 26th and the 36th epochs. The batch normalization layers are not fixed during training. With all the changes, the reimplemented TSM-BN achieves top-1/5 accuracy 44.30/74.53, higher than 43.4/73.2 originally reported in the paper.

Then, we compare the performances when different normalization methods are used in training TSM. Table 6 shows the top-1/5 accuracies of TSM when trained with GN, GN+WS, BN and BN+WS. From the table we can see that WS increases the top-1 accuracy about 2% for both GN and BN. The improvements help GN to cache up the performances of BN, and boost BN to even better accuracies, which roughly match the performances of the ensemble TSM with 24 frames reported in the paper.

3.4. Semantic Segmentation on PASCAL VOC

We continue to show the general applicability of Weight Standardization on the task of semantic image segmentation. Specifically, we choose PASCAL VOC 2012 [6], which contains 21 categories including background. Following common practice, the training set is augmented by the annotations provided in [9], resulting in 10,582 images.

We select DeepLabv3 [5] as the base model for its competitive performance and use ResNet-101 as backbone. We finetune from the respective ImageNet checkpoint. Our reimplementation follows every detail, including 16 batch size, 513 image crop size, 0.007 learning rate with polynomial decay, 30K training iterations, except that we use multi-grid (1, 1, 1) instead of (1, 2, 4). During testing, we stick to output stride 16 and do not use multi-scale or left-right flip of input images. As shown in Table 7, BN achieves 76.49% mean IOU, which aligns with the numbers reported in [5]. Adding WS improves upon BN and GN by 0.66% and 2.30%, respectively. This is further evidence that our Weight Standardization also works well for dense image prediction tasks.

Method	Mean Class Accuracy	Overall Accuracy
GN	87.0	89.7
GN+WS	88.8	91.2
BN	89.3	91.7
BN+WS	89.6	92.0

Table 8. Dynamic Graph CNN [41] on ModelNet40 [43].

3.5. Point Cloud Classification on ModelNet40

Here, we show the generalizability of our method to point cloud classification by evaluating our method on ModelNet40 [43], which contains 40 categories of CAD models, including 9,843 shapes for training and 2,468 for testing.

We follow the state-of-the-art method DGCNN [41] and use authors' implementation[2] for all our experiments. All the settings are kept exactly the same as the implementation. As in [42], we replace all BNs with GNs. The best number of groups is 16 based on our grid search and it is fixed for all experiments.

Table 8 shows that applying WS to GN improves 1.8% and 1.5% respectively on mean class accuracy and overall accuracy. BN+WS further improves 0.3% over BN. This demonstrates that WS also works beyond grid-based CNNs.

4. Related Work

Deep neural networks achieve state-of-the-arts in many vision tasks [4, 14, 20, 25, 31, 32, 39, 46]. Data normalization is widely used for speed up training through proper initialization based on the assumption of the data distribution [7, 11]. However, as training evolves, the normalization effects of initialization may fade away. To ensure normalization throughout the entire training process, Batch Normalization [17] was proposed to perform normalization along the batch dimension, and now is a fundamental component in many state-of-the-art deep networks for its fast training speed and superior performances. Despite its great success, BN's performances can dramatically drop when the batch size is reduced or when the data statistics do not support mini-batch training. To explore other dimensions [2, 33], Layer Normalization [3] normalizes data on the channel dimension, Instance Normalization [40] performs BN for each sample individually, and Group Normalization [42] finds a better middle point between Layer Normalization and Instance Normalization. However, all of them have difficulty matching the performances of BN.

To alleviates BN's issue when the batches become small, BR [16] constrains the statistics of BN within a certain range to reduce their drift when the batch size is not sufficiently large, and [30] proposes synchronized BN which synchronizes statistics across multiple GPUs through engineering. Yet, none of them solve the BN's issues because they still rely on batch knowledge.

Weight Normalization [36] is close to our method in that it also considers weights instead of activations for normalization. But as we have shown in the paper, zero centering weights and gradients is the key to the success of our method instead of the division-based normalization. We have also provided the comparison with it in the experiments, and our method outperform it by a very large margin.

To fully solve BN's issues, many researchers study the underlying mechanism of BN. To name a few, [15] shows that BN is able to make optimization trajectories more robust to the parameter initialization. [28] shows that networks with BN have better generalization properties because they tend to rely less on single directions of activations. [19] explores the effect of length-direction decoupling used in BN and [36]. [27] focuses on the generalization properties of BN by studying its regularization effects. [45] proposes a mean field theory for BN to study the gradient explosion issues. [1] provides theoretical analysis on the auto learning rate tuning property of BN. Our work is based on [37] which shows that BN reduces the Lipschitz constants of the loss and the gradients.

Recently, Luo *et al.* [26] use the idea of AutoML [24, 48] to learn how to combine IN, LN, and BN to get a new normalization method. Shao *et al.* [38] extend [26] by learning a sparse switchable normalization, which is more similar to other AutoML techniques. Since these methods study normalization on activations, our method can also be applied to networks that use them. Due to the time limit, we only apply WS to pre-defined normalization and leave the learned normalization to the future work.

5. Conclusion

In this paper, we propose a novel normalization method Weight Standardization, which is motivated by a recent result [37] that shows the smoothing effects of BN on activations. Similar to BN that smooths loss landscape by normalizing activations, our method further smooths the loss landscape by standardizing the weights in convolutional layers. With our proposed WS, the performances of normalization methods that do not require batch knowledge are improved by a very large margin.

We study WS from both theoretical and experimental viewpoints. On the theoretical side, we investigate the smoothing effects of WS on the loss landscape. We have shown that WS reduces the Lipschitz constants of the loss and the gradients. On the experimental side, we have done comprehensive experiments to show the effectiveness of our WS. The results show that WS+GN with batch size 1 is able to match the performances of BN with large batch sizes when large batch is available, and significantly improve the performances when only micro-batch training is available.

[2] https://github.com/WangYueFt/dgcnn

Acknowledgement

We thank Yuxin Wu for sharing the original drawing of Figure 2. We would also like to thank Yingda Xia and Chenxu Luo for sharing the implementation of TSM.

References

[1] S. Arora, Z. Li, and K. Lyu. Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations (ICLR)*, 2019. 8

[2] D. Arpit, Y. Zhou, B. U. Kota, and V. Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. *arXiv preprint arXiv:1603.01431*, 2016. 8

[3] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2, 5, 8

[4] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *International Conference on Learning Representations (ICLR)*, 2015. 1, 8

[5] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 7

[6] M. Everingham, S. M. A. Eslami, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015. 2, 5, 7

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010. 8

[8] R. Goyal, S. E. Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fründ, P. Yianilos, M. Mueller-Freitag, F. Hoppe, C. Thurau, I. Bax, and R. Memisevic. The "something something" video database for learning and evaluating visual common sense. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5843–5851, 2017. 2, 5, 7

[9] B. Hariharan, P. Arbelaez, L. D. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *IEEE International Conference on Computer Vision (ICCV)*, pages 991–998, 2011. 7

[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017. 6

[11] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. 8

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 4, 5, 6

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *European Conference on Computer Vision (ECCV)*, pages 630–645, 2016. 1

[14] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. 1, 8

[15] D. J. Im, M. Tao, and K. Branson. An empirical analysis of deep network loss surfaces. 2016. 8

[16] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1945–1953, 2017. 8

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015. 1, 2, 5, 8

[18] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017. 7

[19] J. Kohler, H. Daneshmand, A. Lucchi, M. Zhou, K. Neymeyr, and T. Hofmann. Towards a theoretical understanding of batch normalization. *arXiv preprint arXiv:1805.10694*, 2018. 8

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012. 8

[21] J. Lin, C. Gan, and S. Han. Temporal shift module for efficient video understanding. *arXiv preprint arXiv:1811.08383*, 2018. 7

[22] T. Lin, M. Maire, S. J. 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 (ECCV)*, pages 740–755, 2014. 2, 5, 6

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017. 6, 7

[24] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *European Conference on Computer Vision (ECCV)*, pages 19–35, 2018. 8

[25] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. 8

[26] P. Luo, J. Ren, and Z. Peng. Differentiable learning-to-normalize via switchable normalization. *arXiv preprint arXiv:1806.10779*, 2018. 5, 6, 8

[27] P. Luo, X. Wang, W. Shao, and Z. Peng. Towards understanding regularization in batch normalization. In *International Conference on Learning Representations (ICLR)*, 2019. 8

[28] A. S. Morcos, D. G. Barrett, N. C. Rabinowitz, and M. Botvinick. On the importance of single directions for generalization. *arXiv preprint arXiv:1803.06959*, 2018. 8

[29] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013. 2

[30] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun. Megdet: A large mini-batch object detector. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6181–6189, 2018. 6, 8

[31] S. Qiao, C. Liu, W. Shen, and A. L. Yuille. Few-shot image recognition by predicting parameters from activations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 8

[32] S. Qiao, W. Shen, Z. Zhang, B. Wang, and A. Yuille. Deep co-training for semi-supervised image recognition. In *European Conference on Computer Vision (ECCV)*, 2018. 8

[33] M. Ren, R. Liao, R. Urtasun, F. H. Sinz, and R. S. Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. *arXiv preprint arXiv:1611.04520*, 2016. 8

[34] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 91–99, 2015. 6

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 2, 5

[36] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 901–909, 2016. 2, 5, 8

[37] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2488–2498, 2018. 1, 2, 8

[38] W. Shao, T. Meng, J. Li, R. Zhang, Y. Li, X. Wang, and P. Luo. Ssn: Learning sparse switchable normalization via sparsestmax. *arXiv preprint arXiv:1903.03793*, 2019. 8

[39] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 8

[40] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 2, 5, 8

[41] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 8

[42] Y. Wu and K. He. Group normalization. In *European Conference on Computer Vision (ECCV)*, pages 3–19, 2018. 1, 2, 5, 6, 8

[43] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1912–1920, 2015. 2, 5, 8

[44] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017. 5, 6

[45] G. Yang, J. Pennington, V. Rao, J. Sohl-Dickstein, and S. S. Schoenholz. A mean field theory of batch normalization. In *International Conference on Learning Representations (ICLR)*, 2019. 8

[46] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille. Single-shot object detection with enriched semantics. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5813–5821, 2018. 8

[47] B. Zhou, A. Andonian, A. Oliva, and A. Torralba. Temporal relational reasoning in videos. In *European Conference on Computer Vision (ECCV)*, pages 803–818, 2018. 7

[48] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 8