Delving into Variance Transmission and Normalization: Shift of Average Gradient Makes the Network Collapse

Yuxiang Liu[1], **Jidong Ge**[1], **Chuanyi Li**[1] **and Jie Gui** [2]

[1] State Key Laboratory for Novel Software Technology, Nanjing University
[2] School of Cyber Science and Engineering, Southeast University
yxliu@smail.nju.edu.cn, {gjd, lcy}@nju.edu.cn, guijie@seu.edu.cn

Abstract

Normalization operations are essential for state-of-the-art neural networks and enable us to train a network from scratch with a large learning rate (LR). We attempt to explain the real effect of Batch Normalization (BN) from the perspective of variance transmission by investigating the relationship between BN and Weights Normalization (WN). In this work, we demonstrate that the problem of the shift of the average gradient will amplify the variance of every convolutional (conv) layer. We propose Parametric Weights Standardization (PWS), a fast and robust to mini-batch size module used for conv filters, to solve the shift of the average gradient. PWS can provide the speed-up of BN. Besides, it has less computation and does not change the output of a conv layer. PWS enables the network to converge fast without normalizing the outputs. This result enhances the persuasiveness of the shift of the average gradient and explains why BN works from the perspective of variance transmission. The code and appendix will be made available on https://github.com/lyxzzz/PWSConv.

Introduction

We have witnessed the growth of deep learning in computer vision. With the maturity of theories for convolutional neural networks (CNNs) (LeCun et al. 1989; Krizhevsky, Sutskever, and Hinton 2012), several research directions, such as image classification (Simonyan and Zisserman 2015; He et al. 2016a; Szegedy et al. 2016; Huang et al. 2017a) and object detection (Redmon et al. 2016; Ren et al. 2015; Liu et al. 2016), can be explored in depth.

Batch Normalization (BN) (Ioffe and Szegedy 2015), benefiting from its ability to accelerate training speed and reduce the impact of complex initialization and variance transmission (Glorot and Bengio 2010; He et al. 2015), has been widely used in most of the state-of-the-art networks. With the help of normalization, the setting of learning rate (LR) is no longer scrupulous. Generally, the network (without BN) will collapse rapidly if a large LR is set for training. Intuitively, this is attributed to a large update caused by inappropriate LR. By contrast, the appearance of BN alleviates this phenomenon. Moreover, BN accelerates the convergence only by modifying the mean and variance of the output layer, which is puzzling.

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Despite normalization's pervasiveness, the real reason for why they are useful still needs to be studied. However, BN is not omnipotent. It tends to use large batch size and will crash by inaccurate batch statistics estimation when we use a miniature batch size. In (Santurkar et al. 2018; Bjorck et al. 2018), BN was thought to make the optimization landscape significantly smoother. Many variants (Ba, Kiros, and Hinton 2016; Ulyanov, Vedaldi, and Lempitsky 2016; Wu and He 2018; Luo et al. 2019) of BN are proposed. Nonetheless, they may not replace BN entirely because of time cost, performance, and other factors. Inspired by those methods of conducting normalization on outputs, weight normalization (WN) (Salimans and Kingma 2016) and other variants (Qiao et al. 2019; Huang et al. 2017b, 2018) that conduct normalization on filters' weights are proposed. Those methods which normalize the filters usually cooperate with BN to speed up convergence. Many of them may not work if the scale of the output is out of control. Although BN and other methods are effective, changing the outputs may block the variance transmission and lose some information. To better understand why normalizing the outputs is beneficial, we intend to analyze BN from the perspective of variance transmission. We attempt to find a method, which is equivalent to BN and simultaneously robust to mini-batch size. In this way, we may find out why BN works to some extent.

In this paper, we study the variance transmission to find out why BN can speed up convergence and keep the network stable when we use a large LR. We point out that the shift of the average gradient, which is reduced by BN and WN inadvertently, will hinder network training. Therefore, we present Parametric Weights Standardization (PWS) as an alternative to BN and WN. PWS solves the shift of the average gradient by normalizing outputs. It needs less computation and is robust to mini-batch size. Moreover, PWS does not directly control the output variance. PWS acts the same as normal conv operation. We have carried out object detection task in VOC and COCO (Lin et al. 2014) datasets, and image classification task in CIFAR10 and ImageNet (Russakovsky et al. 2015). All models are trained from scratch (Shen et al. 2017; Zhu et al. 2019) to exploit the power of normalization. We refocus on variance transmission to find the discrepancy among normalization operations. Our contributions are:

- Theoretically, we reveal how the shift of the average gradient makes a significant impact on variance transmission.

A huge variance will increase the scale of the gradient. Thus the network collapses.

- We propose Parametric Weights Standardization (PWS), a fast and robust to mini-batch size normalization operation, to solve the shift of the average gradient. Even if we use a large LR, PWS can still be trained without modifying the variance of the output layer. The variance transmits naturally. The relationship between BN and PWS may prove why BN is useful.
- The results indicate that normalizing outputs is not unique to get better results and faster convergence speed.

The Shift of the Average Gradient

Variance Transmission

To understand the role of the shift of the average gradient, we first pay attention to the variance transmission and how the scale of variance influences on the gradient. For a conv layer, we can get

$$Y_l = X_l \otimes W_l + b_l, \qquad X_{l+1} = f(Y_l). \tag{1}$$

Here, l represents the index of a layer. $\otimes$ represents the convolutional operation. Y is the output feature map, which is a $\hat{w}$-by-$\hat{h}$-by-d tensor. $\hat{w}$ and $\hat{h}$ are the spatial width and height of the output feature map. d represents the number of output channels or the number of filters in that conv layer. X is the input feature map, which is a $R^{w\times h\times c}$ tensor. w and h are the spatial width and height of the input feature map. c represents the number of input channels of that conv layer. W_l is a $R^{k\times k\times c\times d}$ tensor. k is the spatial filter size of that layer. We use $n_l = k^2c$ to denote the number of units in one filter of that conv layer l. b is a d-by-1 vector of biases. f is the activation function. In our experiments, f function is ReLU (Nair and Hinton 2010).

Generally, the elements in W_l will be initialized to be mutually independent of one another and have a symmetric distribution with a zero mean. As in (Glorot and Bengio 2010), we can assume that the elements in X_l are also mutually independent of each other and share the same distribution. The elements in W_l and the elements in X_l are independent of each other. For any tensor T, we write $\mathrm{Var}[T]$ and $\mathrm{E}[T]$ for the shared scalar variance and mean of all elements in T, respectively. $\|T\|$ denotes the square root of the quadratic sum of all elements in T. If we assume that $b_l = 0$, $\mathrm{E}[Y_l] = n_l\mathrm{E}[X_l]\mathrm{E}[W_l] = 0$ and Y_l has a symmetric distribution around zero impacted by W_l. This leads to:

$$\begin{aligned} &\mathrm{E}[X_l^2] = \mathrm{E}[relu(Y_{l-1})^2] = \frac{1}{2}\mathrm{Var}[Y_{l-1}], \\ &and \quad \mathrm{Var}[Y_l] = \frac{1}{2}n_l\mathrm{Var}[W_l]\mathrm{Var}[Y_{l-1}]. \end{aligned} \tag{2}$$

For every layer, one suggestion is to initialize $\mathrm{Var}[W_l]$ as $\frac{2}{n_l}$ (He et al. 2015) to transmit variance through ReLU.

Then we focus on the backward propagation case. We get a estimation from (Glorot and Bengio 2010; He et al. 2015):

$$\mathrm{Var}[\nabla_{X_l} L] = (\prod_{i=l}^{m-1} \theta n_{i+1}\mathrm{Var}[W_i])\mathrm{Var}[\nabla_{X_m} L], \tag{3}$$

where m represents the number of layers in our networks. θ is 1 (Glorot and Bengio 2010) when $f'(0) = 1$ and θ is $\frac{1}{2}$ (He et al. 2015) when f is ReLU. We use ReLU as function f. The initialization for $\mathrm{Var}[W_l]$ directly impacts on the gradients for filters in all layers. $\mathrm{Var}[Y_l]$ and $\mathrm{Var}[\nabla_{X_l} L]$ may both explode due to an unreasonable initialization for W. Since $\nabla_{W_l} L = X_l \otimes \nabla_{Y_l} L$, $\nabla_{W_l} L$ will be affected and the network may collapse. The influence of variance transmission will become significant if the outputs are not normalized. It deserves to research how to train the network without normalizing the output.

Gradient Explosion Comes From the Shift

Obviously, an inappropriate initialization for weights will cause the gradient explosion. However, the gradient will also explode even though we use a suitable initialization method. We find that a slightly higher LR can also make networks collapse. Figure 1(a) displays $\mathrm{Var}[W_{o,l} - \Delta W_{o,l}]/\mathrm{Var}[W_{o,l}]$ for every layer. Subscript o is used to index a filter in a layer. $\Delta W_{o,l}$ represents $LR\times\nabla_{W_{o,l}} L$. Here we use SSD (Liu et al. 2016) + VGG (Simonyan and Zisserman 2015) as the object detection model because it is sensitive to LR. If we set LR as 1e-2 and do not use BN after every conv layer, the network will collapse. It seems that using a large LR will not change the variance a lot (Figure 1(a)), but prompts the filter to have a large shift (Figure 1(b)). The changes of variance for different filters are insignificant (Figure 1(a)). Nevertheless, the network indeed collapses.

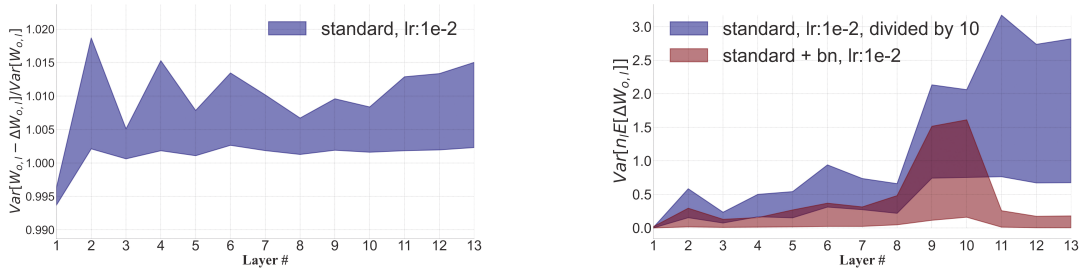
The collapse drives us to define a fine-grained variance propagation formula. Ignoring the bias, we can get $Y_{o,l} = X_l \otimes W_{o,l}$ and $\mathrm{E}[Y_{o,l}] = n_l\mathrm{E}[W_{o,l}]\mathrm{E}[X_l]$ as mentioned before. For a conv layer, the number of output channels is equal to the number of filters. We use $N_l = d$ to denote the number of filters in layer l. According to Figure 1(b), we know $\mathrm{E}[\Delta W_{o,l}]$ for different filters in the same layer are distinct, especially for networks without BN. This shift has a great impact on variance transmission. $Y_{o,l}$ is a part of Y_l, where Y_l is a $\hat{w}$-by-$\hat{h}$-by-d tensor and $Y_{o,l}$ is a $\hat{w}$-by-$\hat{h}$ matrix. We can use $\mathrm{Var}[Y_{o,l}]$ and $\mathrm{E}[Y_{o,l}]$ to express $\mathrm{Var}[Y_l]$:

$$\mathrm{Var}[Y_{o,l}] = \frac{1}{2}n_l\mathrm{Var}[W_{o,l}]\mathrm{Var}[Y_{l-1}], \tag{4}$$

$$\begin{aligned} \mathrm{Var}[Y_l] =& \mathrm{Var}[Y_{l-1}]\frac{\sum_o \frac{1}{2}n_l\mathrm{Var}[W_{o,l}]}{N_l} \\ &+ \mathrm{Var}[n_l\mathrm{E}[W_{o,l}]\mathrm{E}[X_l]]. \end{aligned} \tag{5}$$

A detailed proof is shown in Appendix. Some researches attribute the variance explosion to the change of input distribution and the amplification of the parameters $W_{o,l}$. While Figure 1(a) shows that $\mathrm{Var}[W_{o,l}]$ are stable and not amplified too much, even in a collapsed network corresponding to Figure 1(a). We assume that parameters are initialized reasonably (such as $n_l\mathrm{Var}[W_{o,l}] = 2$ when ReLU is the activation function) and $n_l\mathrm{Var}[W_{o,l}]$ are relatively stable at the beginning of the training. $\mathrm{E}[X_l]$ is independent of subscript o and can be regarded as a constant in $\mathrm{Var}[n_l\mathrm{E}[W_{o,l}]\mathrm{E}[X_l]]$. Thus we can get:

$$\mathrm{Var}[Y_l] \approx \mathrm{Var}[Y_{l-1}] + \mathrm{E}^2[X_l]\mathrm{Var}[n_l\mathrm{E}[W_{o,l}]]. \tag{6}$$



(a) $\mathrm{Var}[W_{o,l} - \Delta W_{o,l}]/\mathrm{Var}[W_{o,l}]$ (b) Average gradients in a layer ($Var[n_l E[\Delta W_{o,l}]]$)

Figure 1: Measuring the gradients for $W_{o,l}$ (Figure a) and the divergence of different filters at the same layer (Figure b). We use o to index a filter in a conv layer. Therefore, W_l is a k-by-k-by-c-by-d tensor and $W_{o,l}$ is a k-by-k-by-c tensor. ΔW_l can be computed as $LR \times \nabla_{W_l} L$. Figure a shows the influence of a training step on the variance of $W_{o,l}$ using SSD backbone (standard). It is explicit that the variance does not have a huge fluctuation even though the network cannot be trained. Figure b shows the variance of the mean value of different filters at the same layer, represented by $\mathrm{Var}[n_l \mathrm{E}[\Delta W_{o,l}]]$. The value for the standard backbone is too large, thus we divide it by 10. Figure b may mean that gradients for each filter in a layer have distinct mean value. $\mathrm{Var}[n_l \mathrm{E}[\Delta W_{o,l}]]$ represents the extent of the shift of the average gradient.

Generally, variance is transmitted through $\frac{\sum_o \frac{1}{2} n_l \mathrm{Var}[W_{o,l}]}{N_l}$. $\mathrm{Var}[\mathrm{E}[W_{o,l}]] \approx 0$ when we initialize W_l. Thus $\mathrm{Var}[Y_l]$ is in normal range transmitted by $\mathrm{Var}[Y_{l-1}]$. Everything has changed when $\mathrm{Var}[n_l \mathrm{E}[W_{o,l}]]$ increases. It is worth mentioning that the shifts of $\mathrm{E}[W_{o,l}]$ are all coming from gradients because we initialize $\mathrm{E}[W_{o,l}]$ to 0. Thus the curves in Figure 1(b) imply an excessive shift on $\mathrm{E}[W_{o,l}]$. Even though $\mathrm{Var}[n_l \mathrm{E}[W_{o,l}]]$ has been divided by 10 for readability, the maximum of this value is up to 5 at initial layers. $\mathrm{E}^2[X_l]$ is less than $\frac{1}{2}\mathrm{Var}[Y_{l-1}]$, but $\mathrm{E}^2[X_l]$ will not be too small because the elements in X_l are all greater than or equal to 0. The range of $\mathrm{E}^2[X_l]$ and $\mathrm{E}^2[X_l]/\mathrm{E}[X_l^2]$ for every layer is shown in Appendix. The increase in $\mathrm{Var}[Y_l]$ may impact on $\mathrm{E}[X_{l+1}]$ and is transmitted to latter layers. At last few layers, $\mathrm{Var}[n_l \mathrm{E}[W_{o,l}]]$ is theatrically up to 30. Those shifts in different filters are continuously multiplied. Normal transmission may be destroyed by shift of the average gradient.

BN's Effect on the Shift of the Average Gradient

First, we discuss about how BN reduces the shift of the average gradient. Formally, we can define a simplified BN of a certain channel of a layer l by formula $\hat{Y}_o = \frac{Y_o - \mu_o}{\sigma_o}$, where $\mu_o = \mathrm{E}[Y_o]$ and $\sigma_o^2 = \mathrm{Var}[Y_o] + \epsilon$. Subscript o is used to index an output channel. ϵ is a small constant. Y_o and $\hat{Y}_o$ is an N-by-w-by-h tensor. w and h are the spatial width and height of the output feature map. N is the batch size. We use $n = N \times w \times h$ to denote the number of elements in a channel of that output feature map. Here we define $\langle U, V \rangle = vec(U)^T vec(V)$, where vec represents flattening a tensor in row major order. Additionally, $\langle U, V \rangle = U \times sum(V)$ if U is a scalar. $sum(V)$ means the sum of all elements in V. We can get:

$$\|\nabla_{Y_o} L\|^2 = \frac{1}{\sigma_o^2}(\|\nabla_{\hat{Y}_o} L\|^2 - \frac{\langle 1, \nabla_{\hat{Y}_o} L\rangle^2 + \langle \nabla_{\hat{Y}_o} L, \hat{Y}_o\rangle^2}{n}), \quad (7)$$

$$\mathrm{E}[\nabla_{Y_o} L] = 0. \quad (8)$$

Detailed proof is shown in Appendix. $\|\nabla_{Y_o} L\|$ will be restricted by the σ_o compared with $\|\nabla_{\hat{Y}_o} L\|$. Thus σ_o impacts on the gradients for W_o. Moreover, $\mathrm{E}[\nabla_{Y_o} L] = 0$ reduce the shift on the weights when backward propagation. As Figure 1(b) has shown, the shift for filters with BN is reduced.

For every channel Y_o in the output feature map, which is computed by W_o, BN will make it zero-mean. Therefore, the influence of the shift of the average gradient will be eliminated because BN's zero-mean operation will let $\mathrm{Var}[\mathrm{E}[\hat{Y}_{o,l}]]$ be zero, although it changes the output information. We speculate that we can get an analogous result without scaling if we keep a robust variance transmission. For Layer Normalization (LN) (Ba, Kiros, and Hinton 2016), Instance Normalization (IN) (Ulyanov, Vedaldi, and Lempitsky 2016), Group Normalization (GN) (Wu and He 2018), and Switchable Normalization (SN) (Luo et al. 2019), the zero-mean operation of those methods also reduces the shift of the average gradients.

WN's Effect on Variance Transmission

Instead of normalizing the outputs, WN divides the filter weights by its Frobenius norm. Formally, we can define a simplified WN of a filter W in a conv layer by $\hat{W} = \frac{g}{\|W\|} W$, where g is a scalar and W is a k-by-k-by-c tensor. c represents the number of input channels of that conv layer. k is the spatial filter size of that layer. We also use $n_l = k^2 c$ to denote the number of units in a filter of that conv layer l. For a conv operation, WN use $\hat{W}$ instead of W to compute the final output. If we set g to a small value, such as 1, instead of Data-Dependent Initialization (Salimans and Kingma 2016), networks with WN can be trained with a large LR. The reason might be the constrained $\mathrm{Var}[\hat{W}]$. We can get $\mathrm{Var}[\hat{W}]$:

$$\mathrm{Var}[\hat{W}] = \frac{g^2 \mathrm{Var}[W]}{\|W\|^2} = \frac{g^2}{n_l} \frac{\mathrm{Var}[W]}{\mathrm{Var}[W] + \mathrm{E}^2[W]}. \quad (9)$$

The elements in W will be initialized to a symmetric distribution with a zero mean. $\mathrm{E}[W] \approx 0$ is established at the

beginning, leading $\mathrm{Var}[\hat{W}]$ to $\frac{g^2}{n_l}$. Considering (2), we know that the value of $\frac{1}{2}n_l\mathrm{Var}[\hat{W}_l] = \frac{g^2}{2}$ should be constrained. We assume that $\mathrm{E}[W] \approx 0$ at the beginning. $\mathrm{E}[\nabla_W L]$ will be constrained when we set g as 1 due to:

$$\mathrm{E}[\nabla_W L] = \frac{g}{\|W\|}\mathrm{E}[\nabla_{\hat{W}} L], \tag{10}$$

$$\|\nabla_W L\|^2 = \frac{g^2}{\|W\|^2}(\|\nabla_{\hat{W}} L\|^2 - \frac{\langle \nabla_{\hat{W}} L, \hat{W}\rangle^2}{g^2}). \tag{11}$$

The detailed proof is shown in Appendix. Filters will be initialized so that $n_l\mathrm{Var}[W] = \|W\|^2 = 2$ as we mentioned above. $\frac{g}{\|W\|} = \frac{g}{\sqrt{2}}$ directly affect the gradients and the shift of the gradients. A suitable g will prevent the gradient from overflow and shift. However, Data-Dependent Initialization sets g to maintain an equivalent variance between the layer's input and output. It may set g to be relatively large, such as $\sqrt{2}$, which amplifies the shifts and becomes unstable.

Moreover, the gradient for filter W will increase due to the weight decay, and finally numeric overflow if LR is large. (11) seems to be similar to BN's. $\|\nabla_{\hat{W}} L\|^2$ will subtract an non-zero value and be constrained by $\|W\|^2$. The input's variance is stable and will not descend a lot during the training. However, $\|W\|$ will be reduced due to the training steps and the weight decay. The descent of $\|W\|$ makes the gradients greater. When we use a large LR and maintain an equivalent variance, the network becomes sensitive and easy to collapse. The detailed illustration is shown in Appendix.

Parametric Weights Standardization (PWS)

To deal with the shift of the average gradient, we focus on how to solve the gradient shift elegantly without normalizing the outputs. Generally, normalizing the outputs requires more computation than normalizing the filters. However, many methods of normalizing filters do not take the output variance and the computation into account. Most of them should be used with BN, GN or other methods of normalizing the outputs. These methods increase the burden of the network to get a better result. To become robust to mini-batch size, and become faster and stabler than those methods, we defined a PWS layer as follow:

$$\hat{W}_o = \sqrt{\frac{2}{n_l}}\frac{W_o - \mathrm{E}[W_o]}{\sqrt{\mathrm{Var}[W_o] + \gamma}}, \tag{12}$$
$$and \quad Y_o = \alpha_o \cdot X \otimes \hat{W}_o + \beta_o.$$

Here W_o is a k-by-k-by-c tensor. c represents the number of input channels of that conv layer. k is the spatial filter size of that layer. Subscript o denotes the index of a filter at that layer. $n_l = k^2c$ denotes the number of units in a filter of that conv layer l. Y_o is a channel of the batch of output feature map as defined above. X is a batch of input feature map. α_o, β_o and γ are scalars. α_o and β_o are trainable and γ is fixed. $\mathrm{Var}[W_o]$ and $\mathrm{E}[W_o]$ represents the calculated variance and mean of all elements in W_o. It is crucial to multiply $\frac{W_o - \mathrm{E}[W_o]}{\sqrt{\mathrm{Var}[W_o]+\gamma}}$ by $\sqrt{2/n_l}$ to maintain a correct variance of $\hat{W}_o$. Most methods that normalize the filters do not adjust the filters' variance and subsequently amplify the outputs. The variance will explode due to (2). It is straightforward that we can eliminate the shift of the average gradient on different filters by letting $\mathrm{E}[\hat{W}_o] \equiv 0$ due to:

$$\mathrm{E}[\nabla_{W_o} L] = -\sqrt{\frac{2}{n_l}}\frac{\mathrm{E}[\hat{W}_o]\langle \nabla_{\hat{W}_o} L, \hat{W}_o\rangle}{n_l\sqrt{\mathrm{Var}[W_o] + \gamma}} = 0. \tag{13}$$

The gradient here is similar to BN's. However, this gradient will directly act on W_o. Thus the gradient for W_o will not shift. Moreover, $\mathrm{E}[\hat{W}_o]$ and $\mathrm{E}[Y_o]$ are correlated due to $\mathrm{E}[Y_o] = n_l\mathrm{E}[X]\mathrm{E}[\hat{W}_o]$ at the beginning of the training, indicating that centralizing the filters is equivalent to centralizing the outputs from the perspective of variance transmission. PWS does not make a strong assumption of letting different inputs to have the same mean value of outputs, which is different from IN. The relationship between BN and PWS may be that they both solve the shift of the average gradient by making $\mathrm{E}[Y_o]$ in different channels equal. This perspective helps us understand why BN works well and why we can use a large LR with BN. Moreover, PWS can be reset to a normal conv layer in the inference stage because PWS does not change the output variance.

Trainable Parameter α

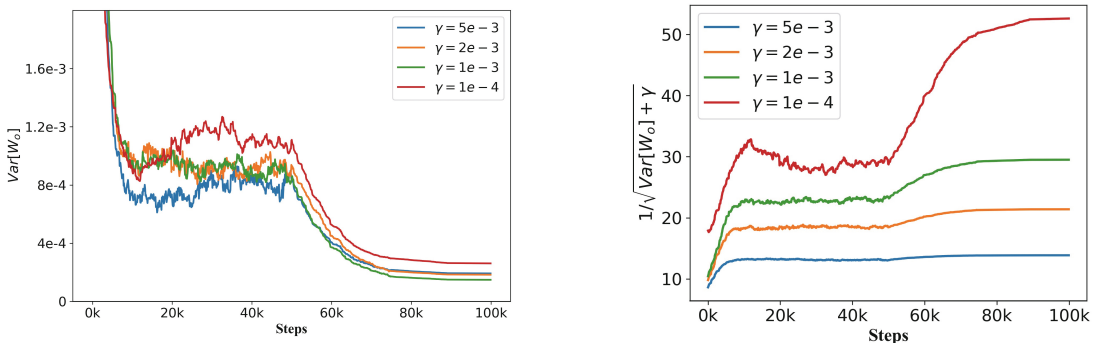
Think about BN's case. In the inference stage, BN will apply constant σ_o and μ_o on the output. Here σ_o and μ_o are calculated by moving mean during the training steps. σ_o and μ_o are fixed values when testing. If we let $P_o = W_o/\sigma_o$ and $Q_o = \beta_o - \mu_o/\sigma_o$, the complete operation will just seem like a regular convolution operation that $Y_o = X \otimes P_o + Q_o$. However, the variance of $\hat{W}_o$ is fixed if we use PWS without α. Thus the output channels will not be attached to a weight and have the same variance. PWS will let the filters in the same layer to have a different weight with α. Simultaneously, α will make PWS layer act the same as conv layer.

Parameter γ

The difference between normalizing outputs and normalizing filters is that the denominator σ in BN is stabler than the denominator in those methods of normalizing filters. $\sqrt{\mathrm{Var}[W_o]}$ will continuously decrease. Figure 2(a) shows the variance of W_o during the training. The variance will be reduced due to weight decay. As the training goes on, the gradient for W_o will increase. If we do not use a large LR, the network may benefit from this increase and converge faster. However, the network may collapse when we use a large LR. γ in $\sqrt{\mathrm{Var}[W_o] + \gamma}$ is not acting as ϵ in BN. As Figure 2(b) has shown, γ can constrain the increase of the reciprocal. Note that with a large LR, even though we set γ to 1e-3 ($\mathrm{Var}[W_o]$ is less than 1.6e-3), the network may also converge. A suitable γ adjusts the training gradient to an acceptable range when we use a large LR. We will show the results in the Experiment part.

Experiments

We conduct experiments in object detection and image classification tasks. To prove PWS's usability, we use VGG (Si-



(a) Variance of W_o

(b) $1/\sqrt{Var[W_o]+\gamma}$

Figure 2: Measuring the minimum of $\mathrm{Var}[W_o]$ in the third conv layer of the backbone for image classification. The training settings are the same as in the Experiments part except for LR and γ. Here LR is 5e-2. The curves of different results adopt different γ. Figure a shows that $\mathrm{Var}[W_o]$ will decrease when the network is training. $\mathrm{Var}[W_o]$ decreases faster, especially at the beginning of the training. γ will not hinder the training of the network and a proper γ will make $\frac{1}{\sqrt{\mathrm{Var}[W_o]+\gamma}}$ stabler.

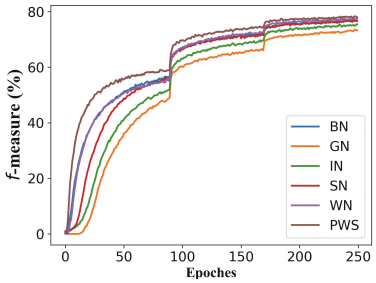


Figure 3: Training conditions of different methods.

monyan and Zisserman 2015) and ResNet (He et al. 2016b) as backbones and train all models from scratch with a large LR. It should be mentioned that we remove all BN layers in the backbone to verify our method's effect. For ResNet, we only remove all BN layers. The network structure is not modified. ReLU is the activation function. We use HE initializer (He et al. 2015) for IN, GN, BN, and SN. We use the same initializer strategy as (Salimans and Kingma 2016) for WN. For PWS, we find that $\sqrt{2/n_l(\mathrm{Var}[W_o]+\gamma)}$ will impact on $\|\nabla_{W_o} L\|^2$. Suppose we initialize the values of all filters in different layers to follow the same distribution. In that case, the gradients for those filters cannot match the gradients for normal conv operation due to different n_l. To match the gradients for normal conv operation, we use HE initializer for PWS rather than set a fixed variance for filters in different layers. α_o is set to 1 and β_o is set to 0. γ is constant for all layers in our experiments. Weight decay for α_o is set as 0. Detailed settings for PWS refer to the configuration in our code repository. Class-wise scores and inference time will also be exhibited in our code repository.

Object Detection in VOC07

All experiments are conducted on SSD (Liu et al. 2016) structure. The LR is set to 1e-2 and is multiplied by 0.1 after 90 and 170 epochs. For normal conv operation, it should be mentioned that the network **cannot be trained** without normalizing outputs when LR is 1e-2. The training finally stops at 240 epochs. We use stochastic gradient descent (SGD) to train the models, where the weight decay is set to 0.0005, and the momentum is set to 0.9. The image size is fixed to 320. We use the same Non-Maximum Suppression (NMS) operation and data augmentation with SSD. All models are trained from scratch in Pascal VOC 07+12 and tested in VOC 07. The train set consists of 16,551 images, and the test set consists of 4,952 images. We do not use any tricks when we train our models and do not use a pre-trained model. γ is set to 1e-3 when LR is 1e-2, and 1e-5 when LR is 1e-3.

Comparison of other normalization methods We experiment with different batch sizes to prove that PWS is robust to mini-batch size. Figure 3 shows the training condition with an LR of 1e-2 and a batch size of 16. The f-measure is calculated by the predicted object box and the ground truth object box. PWS has a faster convergence speed than other methods. Table 1 shows the training speed and the mAP result of different normalization methods in VOC07 test with an LR of 1e-2. These experiments are conducted on a single GPU. Training speed represents number of images that can be trained on an RTX 2080ti per second (batch size$= 8$). IN, BN, GN, and SN are methods of normalizing the outputs. IN does not work well as others, and the possible reason is that IN eliminates the difference in different channels for different images. For every image, IN ensures the $\mathrm{E}[Y_o] \equiv 0$. BN, GN, and SN can work better when training from scratch. However, BN is not robust to mini-batch size (from 76.74 when batch size is 8 to 72.97 when batch size is 2). GN and SN train slower and need more resource. WN and PWS are methods of normalizing the filters. We do not experi-

Methods	Training speed (FPS)	Batch Size 2	4	8
IN	35.34	71.43	72.69	72.97
BN	45.66	72.97	75.18	76.74
GN	29.26	75.08	75.26	75.13
SN	28.56	76.29	76.53	76.86
WN ($g = 1$)	52.11	NaN	76.70	77.06
PWS	52.72	**76.93**	**77.11**	**77.66**

Table 1: Sensitivity to batch size: mAP in VOC07, trained with 8, 4 and 2 images/GPU.

Methods	LR	Backbone	Component $\sqrt{2/n_l}$ in (12)	Trainable α	γ	mAP (%)
Normal conv	1e-3	VGG-16				68.68
BN	1e-3	VGG-16				72.80
SN	1e-3	VGG-16				71.70
PWS	1e-3	VGG-16	✓	✓	1e-5	**72.99**
PWS	1e-3	VGG-16	✓	✓	1e-3	70.32
BN	1e-2	ResNet-50				75.83
PWS	1e-2	ResNet-50	✓	✓	1e-3	**77.19**
BN	1e-2	VGG-16				77.00
PWS	1e-2	VGG-16			1e-3	NaN
PWS	1e-2	VGG-16	✓		1e-3	74.96
PWS	1e-2	VGG-16	✓	✓	1e-3	**77.52**
PWS	1e-2	VGG-16	✓	✓	1e-4	76.50
PWS	1e-2	VGG-16	✓	✓	1e-5	NaN

Table 2: Ablation study in VOC07 test. Batch size is 16.

ment with other methods of normalizing the filters because most of them should be used with BN or GN in computer vision tasks. To make WN feasible to use a large LR, we must manually set g to 1 instead of automatically letting it initialize to ensure the normal spread of variance. Otherwise, the network will collapse at the beginning of the training. However, we found that WN will collapse when using a small batch size. A small batch size leads to more training steps. \${\\{}rm Var}[W_o]\$ becomes smaller due to weight decay, which may make the gradients explode.

Ablation study for PWS Table 2 shows the ablation study for PWS. First, without $\sqrt{2/n_l}$, the network will directly collapse (the row of the first NaN result) and the possible reason is that $\mathrm{Var}[W_l] = 1$ leads to the constraint $\frac{1}{2}n_l\mathrm{Var}[W_l]$ to be $\frac{n_l}{2}$. The row of the first NaN result verifies that networks may directly collapse if $\mathrm{Var}[W_o]$ closes to 1 and the outputs are not normalized. This result indicates that some methods that normalize the filters may directly collapse without BN because they do not concern the variance. The filters' variance must be restricted by this constant parameter to transmit a reasonable variance. The trainable α, which is used to change the scales of filters, works well, improving the result from 74.96 to 77.52. PWS works well under different LR when the backbone is VGG. γ, as we conjecture, constraints the gradient when we use a large LR. The variance of W_o for a conv layer with 64 filters whose spatial size is 3 is 0.00347 ($\frac{2}{3\times3\times64}$). Therefore, setting a large γ will decrease the performance when we use a small LR (1e-3). However, we know that the gradients will increase in WN and PWS. The last row of Table 2 may verify that the decrease of $\|W_o\|$ indeed destroy the training when we use a large LR. Some results have been shown in Figure 2 to illustrate how γ affect the reciprocal. When γ is 1e-5, normal LR can make PWS train well (better than BN in Table 2), but excessive LR will lead to NaN. To avoid the network collapse like WN, we should enlarge γ. For a large LR, the training may not be hindered by a large γ. With a suitable setting, PWS can work well and stably without any other normalization operations compared with WN (collapses when using a small batch size) and is more robust to mini-batch size than BN (from 72.97 to 76.93 when batch size is 2).

Structure	Method	LR schedule	AP^{bbox}	AP^{bbox}_{50}	AP^{bbox}_{75}
Mask R-CNN†	BN	1x	38.0	58.6	41.4
	GN	1x	38.2	59.2	41.1
	PWS	1x	**38.9**	**59.6**	**42.5**
SSD	BN	1x	25.2	**41.3**	26.4
	PWS	1x	**25.3**	41.2	**26.6**
	BN	2x	26.6	42.7	28.1
	PWS	2x	**27.2**	**44.0**	**28.7**

Table 3: Detection result in COCO. †indicates the implementation by (Chen et al. 2019).

Object Detection in COCO

We experiment with Mask R-CNN (He et al. 2017) + FPN (Lin et al. 2017) and SSD structure for COCO dataset. The settings for SSD are the same as the experiments in VOC07 except for LR schedule. Batch size is 16 for SSD. LR is set to 1e-2. The standard LR schedule (1x) contains 80 epochs. LR is multiplied by 0.1 after 30, 50, and 70 epochs, respectively. 2x LR schedule contains 150 epochs. LR is multiplied by 0.1 after 60, 100, and 140 epochs, respectively. SSD uses VGG-16 as the backbone. The settings for Mask R-CNN are the same as in (Chen et al. 2019). We use ResNet-50 as backbone. Mini-batch size is 4. The standard LR schedule contains 12 epochs. LR is multiplied by 0.1 after 8 and 11 epochs, respectively. For Mask R-CNN we use pretrained model to alleviate the impact of batch size on BN. The train set consists of 118k images, and the test set consists of 5k images. For PWS conv, we remove all BN layers and replace all conv layers in the backbone with PWS layer. γ is set to 1e-4. Details are available on our github.

Table 3 shows the results in COCO. For Mask R-CNN, BN gets an inferior result to GN and PWS due to the small batch size. PWS can provide a competitive result with BN and GN and be robust to mini-batch size. For SSD, the results for LR schedule 1x and LR schedule 2x imply that PWS can get similar results to BN under normal batch size. Moreover, PWS does not normalize the outputs and needs less computation, which is distinctive and promising.

Image Classification in CIFAR10

We experiment with PWS for image classification task in CIFAR10. This dataset consists of 60k images, with 50k training examples and 10k test examples. The LR for networks is multiplied by 0.1 after 100, 150, and 180 epochs, respectively. The training finally stops at 200 epochs. We use stochastic gradient descent (SGD), where the weight decay

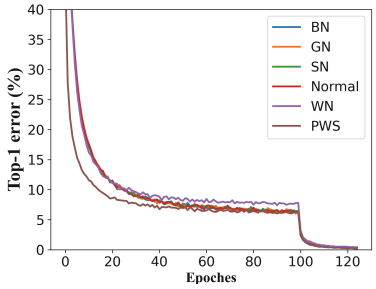


Figure 4: Top-1 error for CIFAR-10 (LR:5e-2).

	Residual		Plain	
Methods	LR: 1e-1	LR: 5e-2	LR: 1e-1	LR: 5e-2
Normal	8.82	9.11	7.28	7.40
BN	**6.12**	9.14	**5.90**	7.67
GN	6.82	9.09	6.77	7.54
SN	6.19	8.90	5.95	7.34
WN	7.18	7.41	6.98	7.38
PWS	6.50	**6.75**	6.16	**6.30**

Table 4: The classification error for CIFAR-10

is set to 0.0005, and the momentum is set to 0.9. The batch size is 128. The image size is fixed to 32. γ is set to 1e-3 for PWS. We follow the simple data augmentation in (Lee et al. 2015) for training. All models are trained from scratch.

We use both plain and residual architectures to verify the generality of PWS. The plain architecture is similar to the ConvPool-CNN-C architecture of (Springenberg et al. 2015), with a modification that abandons all dropout layers. The residual architecture is the same as (He et al. 2016a). Details about the network architecture will be presented in Appendix. GN does not work well. A possible reason may be that the number of channels in a layer is not large enough. In this experiment, γ will not affect a lot because the network is not deep enough. From Table 4, we find BN, GN, and SN will not work better than PWS when LR is small, which may due to the change of output variance. When the LR is set to 1e-1, PWS works not the best. This may be because the gradients for α are too large, leading to some unreasonable updates. Therefore, PWS and WN will both be affected.

Image Classification in ImageNet

We experiment with PWS for image classification task in ImageNet. We train on the 1.28M training images and evaluate on the 50k validation images. We use ResNet-50 as our backbone. The setting for PWS is the same as we discussed in COCO detection task. To get equivalent outputs and gradients to BN, we use LN to normalize the input and the output of ResNet-50. It should be emphasized that we remove all BN layers in ResNet-50. Batch size is 64. LR is 0.025 at the beginning and multiplied by 0.1 after 30, 60, and 90 epochs. The training finally stops at 100 epochs. We use stochastic gradient descent (SGD), where the weight decay is set to 0.0001, and the momentum is set to 0.9. The

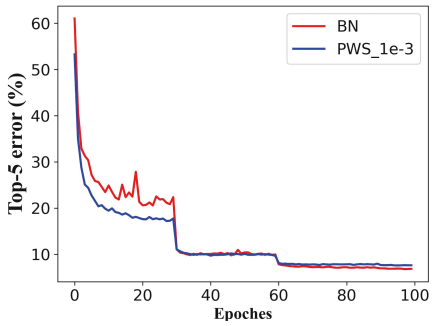


Figure 5: Top-5 error for ImageNet val set.

Measurement	BN	GN	LN	IN	WN	PWS
val error (%)	**23.6**	24.3	25.3	28.4	28.2	24.0

Table 5: Comparison of top-1 error rates (%) of ResNet-50 in the ImageNet validation set.

image size is fixed to 224. We follow the same data augmentation strategy in ResNet-50. From Table 5, it seems that PWS may not work the best for classification task. However, PWS gets a similar result to GN. On the contrary, WN has degraded performance on ImageNet. PWS is the only method which normalize the filters and get a competitive result. Moreover, PWS requires less computation (Table 1) and converges faster (Figure 4 and Figure 5).

Conclusion

PWS is similar to BN. However, it is robust to mini-batch size. The variance transmits naturally with PWS. PWS also has a faster training speed than other normalization operations. The experiment results show that the average gradient migration can affect the network training. Moreover, the results of PWS indicate that normalizing outputs is not the only way to get better results and faster convergence speed.

Nevertheless, it still has problems. We must pay more attention to variance transmission due to the decrease of $\sqrt{\mathrm{Var}[W_o] + \gamma}$. The network can benefit from a proper γ, which is different from WN. Therefore, how to find a suitable γ automatically remains a problem. Our suggestion is to use a large γ when LR is large and use a small γ when LR is suitable for normal conv. If your network is not deep, the choice of γ will not have a huge impact. One suggestion for the choice of γ is to set it as $0.1\sqrt{n_l/2}$.

In this paper, we theoretically analyze the effect of BN from the perspective of variance propagation. We guess that the shift of the average gradient is a problem, which causes the network to collapse. We propose a fast and robust to mini-batch size method called PWS. It is proved that BN and PWS play the same role in the shift of the average gradient, indicating why BN is beneficial. PWS provides another way to speed up the network fitting. We refocus on variance transmission, which is critical to help us understand how networks work and how to make the network more robust.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No.61802167, No.61802095), Natural Science Foundation of Jiangsu Province (Grant No.BK20201250), and Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2019-2-15). Jidong Ge and Jie Gui are the corresponding authors of this paper.

References

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* .

Bjorck, N.; Gomes, C. P.; Selman, B.; and Weinberger, K. Q. 2018. Understanding batch normalization. In *NIPS 2018*, 7694–7705.

Chen, K.; Wang, J.; Pang, J.; Cao, Y.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Xu, J.; Zhang, Z.; Cheng, D.; Zhu, C.; Cheng, T.; Zhao, Q.; Li, B.; Lu, X.; Zhu, R.; Wu, Y.; Dai, J.; Wang, J.; Shi, J.; Ouyang, W.; Loy, C. C.; and Lin, D. 2019. MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155* .

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS 2010*, 249–256.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask r-cnn. In *ICCV 2017*, 2961–2969.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV 2015*, 1026–1034.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep Residual Learning for Image Recognition. In *CVPR 2016*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity Mappings in Deep Residual Networks. In *ECCV 2016*, 630–645.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017a. Densely Connected Convolutional Networks. In *CVPR 2017*, 2261–2269.

Huang, L.; Liu, X.; Lang, B.; Yu, A. W.; Wang, Y.; and Li, B. 2018. Orthogonal Weight Normalization: Solution to Optimization Over Multiple Dependent Stiefel Manifolds in Deep Neural Networks. In *AAAI 2018*.

Huang, L.; Liu, X.; Liu, Y.; Lang, B.; and Tao, D. 2017b. Centered Weight Normalization in Accelerating Training of Deep Neural Networks. In *ICCV 2017*, 2822–2830. IEEE Computer Society.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML 2015*, 448–456.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS 2012*, 1106–1114.

LeCun, Y.; Boser, B. E.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. E.; and Jackel, L. D. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1(4): 541–551.

Lee, C.; Xie, S.; Gallagher, P. W.; Zhang, Z.; and Tu, Z. 2015. Deeply-Supervised Nets. In *AISTATS 2015*, volume 38 of *JMLR Workshop and Conference Proceedings*. JMLR.org.

Lin, T.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollar, P.; and Zitnick, C. L. 2014. Microsoft COCO: Common Objects in Context 740–755.

Lin, T.-Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; and Belongie, S. 2017. Feature pyramid networks for object detection. In *CVPR 2017*, 2117–2125.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S. E.; Fu, C.; and Berg, A. C. 2016. SSD: Single Shot MultiBox Detector. In *ECCV 2016*, 21–37.

Luo, P.; Ren, J.; Peng, Z.; Zhang, R.; and Li, J. 2019. Differentiable Learning-to-Normalize via Switchable Normalization. In *ICLR 2019*.

Nair, V.; and Hinton, G. E. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML 2010*, 807–814.

Qiao, S.; Wang, H.; Liu, C.; Shen, W.; and Yuille, A. 2019. Weight standardization. *arXiv preprint arXiv:1903.10520* .

Redmon, J.; Divvala, S. K.; Girshick, R. B.; and Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *CVPR 2016*, 779–788.

Ren, S.; He, K.; Girshick, R. B.; and Sun, J. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS 2015*, 91–99.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M. S.; et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115(3): 211–252.

Salimans, T.; and Kingma, D. P. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS 2016*, 901–909.

Santurkar, S.; Tsipras, D.; Ilyas, A.; and Madry, A. 2018. How Does Batch Normalization Help Optimization? In *NIPS 2018*, 2488–2498.

Shen, Z.; Liu, Z.; Li, J.; Jiang, Y.; Chen, Y.; and Xue, X. 2017. DSOD: Learning Deeply Supervised Object Detectors from Scratch. In *ICCV 2017*, 1937–1945.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR 2015*.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. A. 2015. Striving for Simplicity: The All Convolutional Net. In *ICLR 2015*.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the Inception Architecture for Computer Vision. In *CVPR 2016*, 2818–2826.

Ulyanov, D.; Vedaldi, A.; and Lempitsky, V. 2016. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv: Computer Vision and Pattern Recognition* .

Wu, Y.; and He, K. 2018. Group Normalization. In *ECCV 2018*, 3–19.

Zhu, R.; Zhang, S.; Wang, X.; Wen, L.; Shi, H.; Bo, L.; and Mei, T. 2019. ScratchDet: Training Single-Shot Object Detectors From Scratch. In *CVPR 2019*, 2268–2277.