Revisiting Batch Norm Initialization

Jim Davis and Logan Frank(✉)

Department of Computer Science and Engineering,
Ohio State University, Columbus, OH, USA
{davis.1719,frank.580}@osu.edu

Abstract. Batch normalization (BN) is comprised of a normalization component followed by an affine transformation and has become essential for training deep neural networks. Standard initialization of each BN in a network sets the affine transformation scale and shift to 1 and 0, respectively. However, after training we have observed that these parameters do not alter much from their initialization. Furthermore, we have noticed that the normalization process can still yield overly large values, which is undesirable for training. We revisit the BN formulation and present a new initialization method and update approach for BN to address the aforementioned issues. Experiments are designed to emphasize and demonstrate the positive influence of proper BN scale initialization on performance, and use rigorous statistical significance tests for evaluation. The approach can be used with existing implementations at no additional computational cost. Source code is available at https://github.com/osu-cvl/revisiting-bn-init.

1 Introduction

Batch normalization (BN) [15] is a standard component used in deep learning, particularly for convolutional neural networks (CNNs) [7,11,28,30,37] where BN layers typically fall into a convolutional-BN-ReLU sequence [11,13]. BN constrains the intermediate per-channel features in a network by utilizing the statistics among examples in the same batch to normalize (or "whiten") the data, followed by a learnable affine transformation to add further flexibility in training. It has been shown that the aggregation of information in a batch is advantageous as it eases the optimization landscape by creating smoother gradients [29] and enables larger learning rates for faster convergence [4,15]. Additionally, the stochasticity from batch statistics can benefit generalization [15,24].

Typically, the default implementation of BN initializes the affine transformation scale (γ) and shift (β) parameters to 1 and 0, respectively, with the expectation that they can adapt to optimal values during training and can undo the preceding normalization if desired [15]. It is common to take this default initialization "as is", with extensions of BN typically adding additional parameters and computations [9,16,22,23] that increase the model complexity.

By inspection of multiple trained BNs, we have observed that the affine parameters often do not change much from their initial values. So either the

S. Avidan et al. (Eds.): ECCV 2022, LNCS 13681, pp. 212–228, 2022.
https://doi.org/10.1007/978-3-031-19803-8_13

affine transformation is hardly needed (staying close to identity) or perhaps there is a limitation in the learning process keeping these parameters from reaching more optimal values. It is unlikely the affine parameters are optimal near identity settings, therefore we question whether the default affine transformation initialization and update approach are appropriate.

In this work, we present a thorough analysis of BN to address the aforementioned hypothesis and provide a new approach for the initialization and updating of BN to enhance overall performance. We will show that 1) reducing the initialization value of the BN scale parameter and 2) decreasing the learning rate on the BN scale parameter can together lead to significant improvements. These alterations to BN are straightforward adaptions to standard implementations with existing libraries (*e.g.*, PyTorch [25], TensorFlow [1]) and have no impact on model complexity or training time. We additionally present a means to further apply BN to the task of input data normalization.

Experiments are provided to compare our proposed BN method to standard BN across multiple popular benchmark datasets and network architectures, and further compare with alternative and existing methods. Notably, we conduct *multiple* training runs, each using a different random seed, for every experiment to properly show statistically significant differences. Results indicate that the proposed technique can yield significant gains with no additional computational costs. Our contributions are summarized as follows:

1. A new BN initialization and update method for improving performance with no increase in parameters or computations.
2. A method that easily integrates into existing BN implementations.
3. An online BN-based input data normalization process.
4. A statistical schema for reporting/evaluating comparative results that eliminates subjectivity and improvements that could be attributed to randomness.

2 Related Work

BN contains operations for both the normalization and transformation of features. Multiple works have proposed related techniques for inter-network feature normalization and/or transformation to achieve improved performance.

Various types of normalization layers exist and differ on how they select which features to group for normalization. Layer normalization (LN) [3] applies to each instance across all input feature channels and is commonly employed in transformer networks [8,33]. Instance normalization (IN) [32] is computed in a similar manner, except it normalizes each channel individually for each instance (example) and is frequently used in image-to-image translation [41]. Group normalization (GN) [36] falls between LN and IN by normalizing groups of channels in individual instances. When the group size equals the total number of channels, GN becomes LN. Similarly, when each channel is a separate group, GN becomes IN. The aforementioned normalization layers were designed to perform better than BN in situations where the training batch size is limited. However, BN remains dominant in the context of CNNs and image classification due to

the ability of having large batch sizes on common datasets [11,28,31,37]. In our work, we focus on improving the affine transformation component in BN, though our work could be applied to any of the normalization layer types.

Related work has appeared to extend the affine transformation component of BN. Rather than using only learnable scale and shift parameters, the approach of [16] predicts scale and shift values using a small autoencoder network and combines them with the standard learnable versions. Similarly, in [22] they combine predicted and learned scale and shift parameters by utilizing a mixture of affine transformations and a feature attention mechanism to obtain the final transformation. A convolutional layer is used to replace the affine transformation in [38]. In [23], an attention-based transformation is introduced to integrate instance-specific information into an extra scale parameter. In [9], two different calibration mechanisms are proposed and integrated into BN to calibrate features and to incorporate instance-specific statistics. These mechanisms consist of an initial centering operation that occurs before normalization and a scaling operation in the affine transformation. Their scaling is similar to [23], differing by using the *normalized* features rather than the *raw* input features for the instance-specific information, as well as employing a different initialization. In [2], it is mentioned that the initial value of the scale parameter could be treated as a hyperparameter, though not experimented or discussed further. Alternative initialization values for the shift parameter and numerical stability constant (ϵ) are explored in [39].

The approaches of [9,16,22,23] all initialize the existing BN affine transformation parameters to the standard values of $\gamma = 1$ and $\beta = 0$, but employ different initializations for their additional parameters. In [16,22], their additional parameters are initialized using samples from a normal distribution, thus their initial scale values are not constant across the network and could have a magnitude >1. In [9], they initialize the additional parameters for their scaling operation such that they "play no role at the beginning of training" [9], however, the resulting initial scale value is actually <1. A grid search is performed in [23] to select the best performing initialization values, which also produces a resulting initial scale value <1, with the "theoretical understanding of the best initialization [left as] future work" [23]. For both [9,23], the resulting scale value is <1, constant across all BNs in the network, and fixed for any scenario. The approaches of [9,23] are most similar to our work, but unlike these methods, our approach does not introduce any additional parameters or computations and instead focuses directly on the initialization and updating of the existing scale parameter γ for each BN.

3 Framework

In this section, we initially review the BN formulation then describe the two main tenets of our approach: BN scale initialization and learning rate reduction. We then derive the influence of BN on the backward gradients and discuss various aspects that are influenced by the inclusion of BN. Lastly, we introduce a new BN for input data normalization.

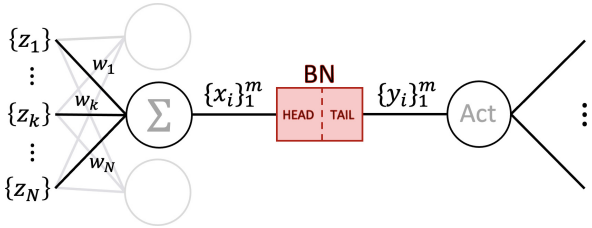

Fig. 1. Flow diagram for a single channel/neuron of BN

3.1 BN Formulation

BN is composed of two sequential channel-wise operations. The first operation, the "head", normalizes the data. The second operation, the "tail", performs an affine transformation on the normalized data. Both operations are applied to each individual channel/neuron across a batch of data.

In the forward pass during training, the head operation first obtains the mean (μ_B) and variance (σ_B^2) of an incoming batch B of data/features containing m examples ($X = \{x_i\}_1^m$) using $\mu_B = \frac{1}{m}\sum_{i=1}^m x_i$ and $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^m (x_i - \mu_B)^2$. With these statistics, the input batch is then normalized to have zero mean and unit standard deviation using

$$\hat{X} = \frac{X - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{1}$$

where ϵ is a small value used for numerical stability. Throughout training, μ_B and σ_B^2 are employed to update momentum-based averages of the overall mean and variance, which serve as global statistics at test time. Last, the tail affine transformation $Y = \gamma \cdot \hat{X} + \beta$ is applied to the normalized data ($\hat{X}$) using learnable scale (γ) and shift (β) parameters (again, one for each channel).

The data flow when using BN is shown in Fig. 1, where BN is typically placed between a linear operator and an activation function. The figure depicts a fully-connected layer for ease in explanation and is easily extended to CNNs.

3.2 Proposed Adjustments to γ

The standard procedure for initializing each BN is setting $\gamma = 1$ and $\beta = 0$, with the expectation that the affine transformation can learn optimal values and could undo the preceding normalization if desired [15]. Instead, as we have observed, the learned BN parameters tend to remain close to their initial values which are not likely to be optimal. Furthermore, we have noticed that the BN normalization head (Eq. 1) can often produce overly large values (*e.g.*, $>6\sigma$) as input to the BN affine tail and subsequent convolutional/fully-connected layer (all before another BN), which can be undesirable for learning [15,20,21]. Therefore, there could exist more optimal values of γ not achieved with current learning strategies.

A possible direct solution based on these observations would be to initialize $\gamma < 1$ such that the data is immediately scaled down at the start of training, addressing the presence of overly large values, and furthermore enabling the shift parameter β to have a broader reach on the scaled data before the activation function. Therefore, we propose to treat γ as a hyperparameter (for all BNs in the network) to be initialized in the interval $(0, 1]$ while leaving the default initialization of $\beta = 0$. We will show later that there exists a range of γ initialization values within the interval that can lead to significant increases in test accuracy over the default $\gamma = 1$ initialization.

Since we observed that the learned BN parameters tend to remain somewhat close to their initialization (regardless of the initial value and other hyperparameter settings), it seems necessary to also consider altering the update strategy of these parameters. A seemingly intuitive solution would be to increase the learning rate (α) for the BN parameters only, thus allowing larger updates and the ability to potentially explore more optimal values in the parameter space. However, we have seen this *degrade* performance. Instead, we propose to *reduce* the learning rate for only the scale γ to enable a more fine-grained search, leaving the shift β with larger updates to have a broader and more stable search of the normalized and γ-scaled data. More specifically, we apply a learning rate reduction on γ using $\alpha_\gamma = (\alpha/c)$, where c is a positive constant. Though the value of c could be considered another hyperparameter, we experimented with a wide range of values for c and found that performance gains can be achieved as long as the learning rate for γ is sufficiently and reasonably reduced. Thus, we *fix* this γ learning rate reduction *constant* to $c = 100$ for *all* experiments.

With a proposed smaller initial value for γ, it is important to theoretically understand any effect it will have on the gradients for learning. We next examine the gradients of BN to show the influence of γ and other important relationships.

3.3 BN Gradients

To update a network via gradient descent and backpropagation, loss gradients must be able to propagate through the BN module. For example, parameter w_1 in Fig. 1 is updated using gradient descent by

$$w_1^{new} = w_1^{old} - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w_1} \tag{2}$$

Using the chain rule with a batch of data $\{x_i\}_1^m$, the loss with respect to w_1 is computed as

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_1} \tag{3}$$

The BN input x_i is computed from the linear operation $x_i = \sum_{k=1}^{N} w_k \cdot z_{k,i}$, where $z_{k,i}$ is the i^{th} batch value from the k^{th} channel output from the previous layer (see Fig. 1). Therefore $\partial x_i / \partial w_1 = z_{1,i}$ and Eq. 3 becomes

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial x_i} \cdot z_{1,i} \tag{4}$$

Similarly, the gradient $\partial\mathcal{L}/\partial x_i$ through the BN is computed using the chain rule (described in [15]) as

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial \hat{x}_i}\frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial \mathcal{L}}{\partial \sigma_B^2}\frac{\partial \sigma_B^2}{\partial x_i} + \frac{\partial \mathcal{L}}{\partial \mu_B}\frac{\partial \mu_B}{\partial x_i} \tag{5}$$

with the x_i dependent gradients

$$\frac{\partial \hat{x}_i}{\partial x_i} = \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \frac{\partial \sigma_B^2}{\partial x_i} = \frac{2}{m}(x_i - \mu_B) \qquad \frac{\partial \mu_B}{\partial x_i} = \frac{1}{m} \tag{6}$$

and the remaining gradients

$$\frac{\partial \mathcal{L}}{\partial \hat{x}_i} = \frac{\partial \mathcal{L}}{\partial y_i} \cdot \gamma \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}_i}(x_i - \mu_B) \cdot \frac{-1}{2}\left(\sigma_B^2 + \epsilon\right)^{-\frac{3}{2}} \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial \mu_B} = \left(\sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}}\right) + \frac{\partial \mathcal{L}}{\partial \sigma_B^2} \cdot \frac{2}{m}\sum_{i=1}^{m}(x_i - \mu_B) \tag{9}$$

where $\partial\mathcal{L}/\partial y_i$ is the gradient accumulated from all downstream layers. For thoroughness, the gradients for the affine transformation parameters γ and β are

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y_i} \cdot \hat{x}_i \qquad \frac{\partial \mathcal{L}}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y_i} \tag{10}$$

Substituting Eqs. 6–9 into Eq. 5, we show that

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x_i} = &\left(\frac{\partial \mathcal{L}}{\partial y_i} \cdot \gamma\right)\left(\frac{1}{\sqrt{\sigma_B^2 + \epsilon}}\right) + \\ &\left(\sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}_i}(x_i - \mu_B) \cdot \frac{-1}{2}\left(\sigma_B^2 + \epsilon\right)^{-\frac{3}{2}}\right)\left(\frac{2}{m}(x_i - \mu_B)\right) + \\ &\left(\left(\sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}}\right) + \frac{\partial \mathcal{L}}{\partial \sigma_B^2} \cdot \frac{2}{m}\sum_{i=1}^{m}(x_i - \mu_B)\right)\left(\frac{1}{m}\right)\end{aligned} \tag{11}$$

and after multiple reduction steps (novel to our work),

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}} \cdot \left[\frac{\partial \mathcal{L}}{\partial y_i} - \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y_i}(1 - \hat{x}_i)\right] = \frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}} \cdot G_i \tag{12}$$

where G_i is a function of the downstream gradients of y_i (*after* the BN layer) and the normalized values $\hat{X}$ (*within* the BN layer). We refer to the leading ratio $\gamma/\sqrt{\sigma_B^2 + \epsilon}$ as the BN 'gradient factor', which is composed of the affine

scale parameter γ for this current BN and the variance σ_B^2 of the *incoming* data to that BN. Therefore the update for w_1 (Eq. 2) is computed as

$$w_1^{new} = w_1^{old} - \alpha \cdot \frac{\gamma}{\sqrt{\sigma_B^2 + \epsilon}} \left(\frac{1}{m} \sum_{i=1}^{m} G_i \cdot z_i \right) \tag{13}$$

We will next show how this gradient factor relates to various aspects of learning.

Gradient Influence of γ. Following Fig. 1 at the beginning of training *before any updates*, the incoming data variance σ_B^2 to the BN is

$$\sigma_B^2 = \text{Var}(\{x_i\}_1^m) = \sum_{k=1}^{N} w_k^2 \cdot \text{Var}(\{z_{k,i}\}_1^m) = \sum_{k=1}^{N} w_k^2 \cdot \sigma_k^2 \tag{14}$$

As each of the k input batches $\{z_{k,i}\}_1^m$ come from the output of a *previous* BN (each normalized then scaled with γ_{prev}) that is also passed through an activation function,

$$\sum_{k=1}^{N} w_k^2 \cdot \sigma_k^2 = \sigma^2 \sum_{k=1}^{N} w_k^2 = \sigma_{act}^2 \cdot \gamma_{prev}^2 \sum_{k=1}^{N} w_k^2 = \sigma_{act}^2 \cdot \gamma_{prev}^2 \cdot \omega \tag{15}$$

where σ_{act}^2 is the variance of the normalized (but unscaled) data after being passed through an activation function. For example, a unit Gaussian passed through a ReLU yields a *rectified* normal distribution with an empirical $\sigma_{act}^2 \approx (0.58)^2$. When the normalized data is scaled by γ and passed through an activation, the resulting variance is $\sigma^2 = \sigma_{act}^2 \cdot \gamma^2$.

From Eq. 15, the scale parameter γ_{prev} of the *previous* layer BNs is therefore embedded in the *incoming* data variance (σ_B^2). At the start of training, where $\gamma_{prev} = \gamma_{curr}$, the scales will essentially cancel in the gradient factor of Eq. 12

$$\frac{\gamma_{curr}}{\sqrt{\sigma_B^2 + \epsilon}} = \frac{\gamma_{curr}}{\gamma_{prev} \cdot \sqrt{\sigma_{act}^2 \cdot \omega + \epsilon}} = \frac{1}{\sqrt{\sigma_{act}^2 \cdot \omega + \epsilon}} \tag{16}$$

again assuming a negligible ϵ. Thus for the initial backward pass, the initial BN scale value for γ has *no effect* on the local BN gradients (the first BN layer will be addressed in Sect. 3.4). While γ will not affect the initial backward pass, it will contribute to the forward pass and loss. Though the value of each γ will naturally migrate toward a more optimal value during training, each BN gradient factor should remain non-degenerate. Therefore initializing $\gamma < 1$ will not cause any cascading issues during training.

Network Weight Initialization. Most network weight initialization schemes take into account the number of either incoming or outgoing links in a layer. For example, Kaiming normal initialization [10] initializes the weights in a network following a normal distribution $\mathcal{N}(0, \sigma^2)$ with $\sigma = \text{gain}/\sqrt{\text{fan\_mode}}$, where 'gain' is a constant determined by the activation function used in the network and 'fan_mode' is a value representing either the number of incoming (fan-in) or

outgoing (fan-out) links. Thus the number of links affects the standard deviation of the normal distribution, with more links yielding a tighter distribution.

As shown in Eqs. 14 and 15, the variance σ_B^2 of the data going into a BN at the start of training is dependent on the sum of the squared *incoming* weights (ω). Therefore, weight initialization approaches based on *fan-in* will initially balance the gradient factor for each BN across the network. In our experiments we therefore employed fan-in weight initialization in our networks.

3.4 BN-Based Input Normalization

Traditional methods for normalizing the input data typically utilize either "fixed" bounds based on the data type min/max or compute "offline" dataset statistics:

- Fixed: Chooses the mean (μ) as the middle value of the data type and the standard deviation (σ) as the $\pm$ maximum possible value of the mean-subtracted data (*e.g.*, $\mu = \sigma = 128$ for an 8-bit [0–255] image).
- Offline: Preprocess the data to estimate the μ and σ based on the average of averages across batches or individual examples in the dataset.

These methods normalize the data using $(X - \mu)\ /\ \sigma$ (as used in Eq. 1 for BN).

The fixed input normalization technique can be problematic for images not spanning the full possible range of values for its data type. For example, with a dataset of low-contrast imagery the true μ and σ would be much smaller. The offline statistics approach obviously requires the extra steps of computing the dataset μ and σ before training.

Consider the BN gradient factor $\gamma/\sqrt{\sigma_B^2 + \epsilon}$ for the weights just before the *first* BN in the network on the *initial* backward pass. Here the variance σ_B^2 is computed from the input data itself (Eq. 14). Examining Eq. 15, we expect this variance to be $\sigma_{act}^2 \cdot \gamma^2 \cdot \omega$ as for all other layers, however neither the scale γ or activation are present. Therefore, a *smaller*/*larger* σ^2 of the input data, where $\sigma^2 \neq \sigma_{act}^2 \cdot \gamma^2 \cdot \omega$, will result in a *larger*/*smaller* gradient factor as compared with the other layers in the network.

This bias with traditional input normalization techniques can be removed by conveniently prepending a new BN layer to the network. Since it would not be ideal to immediately threshold the input data with an activation following this new BN layer, we initialize these BNs to have a modified scale value $\hat{\gamma} = \sigma_{act} \cdot \gamma$, where $\hat{\gamma} = 0.58 \cdot \gamma$ for a ReLU-based network, to account for the missing activation function's influence on the variance. For this BN layer, the built-in shift parameter β is unnecessary as no activation function follows and can therefore be removed or fixed to $\beta = 0$. Thus, employing this BN-based input data normalization method will only add a single scale parameter $\hat{\gamma}$ per input channel (*e.g.*, 3 parameters total for an RGB image in a CNN).

4 Experiments

To evaluate our proposed approach, we conducted a series of classification experiments using several datasets and network architectures, examined different possible versions of our approach, and compared to relevant existing methods.

4.1 Datasets and Network Architectures

We employed four established classification datasets for our evaluation. CIFAR-10 [19] contains 10 classes, each with 5K and 1K images for training and testing, respectively. The larger, yet distinct, CIFAR-100 [19] has 100 classes, each with 500 training and 100 testing examples. The fine-grained visual classification (FGVC) dataset CUB-200 [34] contains 200 classes with 5994 training and 5794 testing examples. Finally, Stanford Cars (ST-Cars) [18] is another FGVC dataset that consists of 196 classes with 8144 and 8041 images for training and testing, respectively. In order to have a validation set for each of the datasets, we randomly sampled 10% of the training examples class-wise for validation.

We primarily employed networks from the ResNet [11] architecture family, using ResNet-18 for CIFAR-10, ResNet-34 for CIFAR-100, and ResNet-50 for CUB-200/ST-Cars. Due to the smaller image sizes of CIFAR-10/CIFAR-100, common modifications to its associated model were made which consisted of altering the first convolutional layer to have a 3×3 kernel with stride of 1 and padding of 1 (originally a 7×7 with stride of 2 and padding of 2) and removing the max pooling operation that followed the initial convolutional layer. For initialization, convolutional layer weights used Kaiming normal initialization [10] with 'fan-in' mode for a ReLU nonlinearity, fully connected layer weights were initialized with a uniform distribution (also using 'fan-in'), and all biases (including the BN shift parameter β) were initialized to 0. The BN scale parameter γ will be initialized with various values to demonstrate our approach. We additionally examined RepVGG [7], MobileNetV2 [28], and ResNeXt [37] architectures (with the same initialization strategy).

4.2 Training Details

Training is implemented using a standard regime and in a manner where all hyperparameters are set using typical values. Across all experiments, we used SGD with momentum (0.9) and weight decay (1e−4) on the convolutional and fully connected layer weights. BN parameters and all network biases are properly excluded from weight decay as it is not necessary to impose a constraint to minimize these parameter values. Networks were each trained for 180 epochs with a half-period cosine learning rate scheduler and across multiple initial learning rates (to be presented). We used a batch size of 128 for CIFAR-10/CIFAR-100 and 64 for CUB-200/ST-Cars. Data augmentation schemes for CIFAR-10/CIFAR-100 consisted of random horizontal flipping only and for CUB-200/ST-Cars images were resized to 256×256 then randomly cropped to 224×224, followed by random horizontal flipping. After augmentations, the input data is normalized online using our proposed BN-based input normalization technique with the initialization of $\hat{\gamma} = 0.58 \cdot \gamma$ corresponding to ReLU activations (Sect. 3.4). In our experiments, the epoch yielding the best validation accuracy was used to select the final model. Our approach was implemented using PyTorch [25] and all models were trained and evaluated on a single NVIDIA V100 GPU.

We note that we *purposefully* used minimal regularization techniques in our experiments to show noticeable improvements that can be attributed solely to changes within BN using our approach. Our goal is *not* to provide new SOTA scores on the datasets used for evaluation. However, if more regularization is employed to improve baseline performance, our approach may still provide further gains (as shown in an experiment below).

To demonstrate our proposed BN scale initialization approach, we explored a variety of possible γ values in the half-open unit interval (0, 1] and several learning rates α. Smaller values of γ and larger values of α are typically favored and thus we examined a subset of values for γ and α after the initial CIFAR-10 experiments. As described in Sect. 3.2, we also divided α by $c = 100$ for only γ. For our baseline comparison, referred to as 'BASE', we used the default BN scale initialization ($\gamma = 1$) and no learning rate reduction ($c = 1$), but retained the BN-based input normalization for fair comparison.

4.3 Statistical Significance

As variations in the final score (accuracy) can be caused by different random number generator (RNG) seeds resulting in different network weight initializations and different batching of training data [26,35], we conducted several runs for each experiment and performed a statistically-grounded comparative analysis on our results. For *each* experiment (a specific BN scale value γ and learning rate α), we trained 15 networks (of the same form), each with a different RNG seed, and reported the mean and standard deviation of test accuracy. Our seeds were sequential numbers in the range [1, 15], which were made more complex using MD5 as suggested in [17,27].

A result is judged to be significantly better than BASE according to a one-sided paired t-test [6] with a significance level of $p \leq 0.05$ using the scores from the 15 runs. All significant improvements over BASE are emphasized in **bold** in the following tables and the highest mean for each experiment is underlined. We suggest this method as a proper technique to evaluate and compare empirical results to instill confidence for the reader on reported improvements.

4.4 Results

We first conducted experiments on CIFAR-10 using the selected BN scale initialization values, then compared our method with possible alternative BN scale formulations. Next, we evaluated our method on different datasets and network architectures, and then compared with different methods of input normalization. Finally, we compared to relevant established approaches on all of the datasets.

CIFAR-10. For our initial experiments with CIFAR-10, we used $\gamma \in \{1.0, 0.75, 0.5, 0.25, 0.1, 0.05, 0.01\}$ and $\alpha \in \{0.1, 0.01, 0.001\}$. Results on CIFAR-10 are presented in Table 1a. Using the proposed $\gamma < 1$ initialization with the learning rate reduction ($c = 100$), significant improvements are found across all learning rates. These results alone demonstrate that the default settings and learning

Table 1. Accuracy on (a) CIFAR-10 and (b) CIFAR-100, CUB-200, and ST-Cars

(a)

γ	Learning Rate (α) 0.1	0.01	0.001
0.01	85.50±0.39	**87.11**±0.23	**80.37**±0.58
0.05	**90.19**±0.32	**88.84**±0.32	**76.98**±0.71
0.10	**90.80**±0.20	**87.31**±0.37	**74.48**±0.55
0.25	**90.32**±0.24	**85.33**±0.43	**73.83**±0.64
0.50	**90.17**±0.19	84.60±0.35	**72.80**±0.68
0.75	**90.19**±0.18	84.43±0.30	**72.01**±0.58
1.00	**89.81**±0.46	84.48±0.33	71.15±0.56
BASE	89.44±0.45	84.64±0.25	71.32±0.60

(b)

Dataset	γ	Learning Rate (α) 0.1	0.01
CIFAR100	0.05	**68.18**±0.30	**64.01**±0.54
	0.10	**68.80**±0.49	**62.83**±0.48
	0.50	**67.74**±0.44	**59.05**±0.41
	BASE	66.01±0.95	58.48±0.53
CUB-200	0.05	**58.32**±0.54	34.23±0.86
	0.10	**58.52**±0.69	39.92±0.76
	0.50	**50.45**±1.22	**45.31**±0.59
	BASE	46.26±1.59	41.61±1.03
ST-Cars	0.05	**78.29**±0.44	44.18±1.29
	0.10	**78.26**±0.61	48.60±1.02
	0.50	**70.17**±1.45	51.18±2.16
	BASE	64.73±2.87	51.86±1.80

of the BN affine transformation (BASE) are not ideal. For this dataset, there is a noticeable trend that as the learning rate decreases, a smaller initialization value for γ is favored, with smaller values for γ preferred in general. Performance dropped heavily for BASE at lower learning rates, however our approach with smaller γ initializations had much less of a decrease. We also experimented with initially setting the BN scale parameter to $\gamma > 1$, but experiments produced degraded results, as expected.

With only the learning rate reduction method ($c = 100$, $\gamma = 1$), there remained a significant gain over BASE ($c = 1$, $\gamma = 1$) at the highest learning rate ($\alpha = 0.1$). To confirm the importance of learning rate reduction, we ablated it from our method (*i.e.*, $c = 1$, $\gamma < 1$) and results showed *degraded* performance from the scores presented in Table 1a. Thus, combining our proposed BN scale initialization and learning rate reduction *together* is necessary to obtain the best performance.

In additional experiments with a *stronger* BASE model (employing RandomCrop data augmentation [11,12,40]) having 93.79% accuracy, our proposed BN scale initialization at $\gamma = \{1.0, 0.75, 0.5, 0.25, 0.1\}$ and BN scale learning rate reduction at $c = 100$ still provided statistically significant improvements.

Alternative Scale Formulation. Given the standard affine formulation $y = \gamma \cdot \hat{X} + \beta$ of BN (as we employ), two similar scale-based alternatives could be A1: $Y = \gamma(\hat{X} + \beta)$ and A2: $Y = \gamma(\gamma_0 \cdot \hat{X} + \beta)$. In all three formulations, initialization is $0 < \gamma \leq 1$, $\gamma_0 = 1$, and $\beta = 0$. The two alternatives differ from the standard formulation in that they apply scaling to β as well, which we argue could limit the ability to shift the data sufficiently before the ReLU activation. While A1 does not introduce any extra parameters or additional computations (similar to ours), A2 includes the extra γ_0 parameter (and is similar to [14]).

We compared the three formulations on CIFAR-10 using the same γ values and learning rates presented in Table 1a. All approaches gave significant improvements over BASE, further indicating that the default initialization of

BN is not appropriate. At the highest learning rate, there was only a slight difference between the scores (within 0.15% of each other). However, at the lower learning rates the standard formulation (ours) had a clear and significant advantage (up to 7% gain). These results further support our argument that scaling down only the normalized data with γ enables the shift parameter β to act on a broader range for the following activation function.

CIFAR-100, CUB-200, and ST-Cars. We next evaluated our approach on CIFAR-100, CUB-200, and ST-Cars. As mentioned in Sect. 4.1, we employed ResNet-34 for CIFAR-100 and used ResNet-50 for CUB-200/ST-Cars. Networks for CUB-200/ST-Cars typically employ pretrained ImageNet [5] models and finetune, however since pretrained models utilize the default BN initialization (which affects the BN statistics and network weights learned), we therefore train all of our networks from scratch. The results with $\gamma \in \{0.5, 0.1, 0.05\}$ and $\alpha \in \{0.1, 0.01\}$ are reported in Table 1b.

It is clear that our approach with smaller γ initialization and reduced γ updates can still produce significant improvements in test accuracy for these datasets that contain a larger number of classes than CIFAR-10. For the highest learning rate, all examined scale values produced significantly better results, but at the lower learning rate for ST-Cars it was unable to achieve significant gains with any of the initialization values, though one scale value ($\gamma = 0.5$) was not significantly different from BASE. One could argue that this lower learning rate is not adequate for ST-Cars.

Network Depth and Architectures. We further examined how our approach extends to different ResNet depths and other network architectures on CIFAR-10. In particular, we trained and examined the deeper ResNet-50, ResNet-101, and ResNet-152 models, and additionally employed MobileNetV2 [28], RepVGG-A0 [7], and ResNeXt-50 [37]. Again, since CIFAR-10 consists of small input image sizes, modifications were made to the other architectures. For MobileNetV2, the initial convolutional layer and second and third bottleneck blocks were changed to have a stride of 1. Similarly, for RepVGG-A0, stage 0 and stage 1 were adjusted to have a stride of 1. ResNeXt-50 was modified similar to ResNet (Sect. 4.1).

Results are shown in Table 2a for $\gamma \in \{0.5, 0.1, 0.05\}$ and $\alpha \in \{0.1, 0.01\}$. Our performance increase was even greater with the deeper ResNet models (as compared to ResNet-18), which suggests our method may be particularly beneficial for training networks containing more parameters/layers faster than with the default BN initialization. Furthermore, our approach transferred well to the other network architectures, emphasizing the generality of our $\gamma < 1$ initialization and learning rate reduction.

Input Normalization. We next compared our proposed input data normalization BN layer to the standard fixed and offline methods (Sect. 3.4). All experiments were conducted on CIFAR-10 and employed our proposed $\gamma < 1$ initialization and learning rate reduction ($c = 100$). Therefore, any differences reported are solely due to how the input data was normalized. Results showed that all three methods produced similar results. Therefore, our proposed online BN-

Table 2. (a) Accuracy on CIFAR-10 employing various network depths and network architectures. (b) Comparison with RBN and IEBN variations on CIFAR-10, CIFAR-100, CUB-200, and ST-Cars

(a)

Network	γ	Learning Rate (α) 0.1	0.01
ResNet-50	0.05	**91.23**$\pm$0.20	**89.60**$\pm$0.19
	0.10	**91.28**$\pm$0.26	**87.67**$\pm$0.20
	0.50	**89.49**$\pm$0.27	84.74$\pm$0.37
	BASE	86.94$\pm$1.23	85.04$\pm$0.32
ResNet-101	0.05	**91.58**$\pm$0.22	**90.02**$\pm$0.22
	0.10	**91.26**$\pm$0.18	**88.35**$\pm$0.28
	0.50	**89.89**$\pm$0.74	**85.23**$\pm$0.50
	BASE	88.28$\pm$1.39	84.74$\pm$0.56
ResNet-152	0.05	**91.20**$\pm$0.16	**90.00**$\pm$0.17
	0.10	**90.89**$\pm$0.41	**88.31**$\pm$0.33
	0.50	**90.17**$\pm$0.23	**85.23**$\pm$0.62
	BASE	88.73$\pm$0.62	84.15$\pm$0.79
RepVGG-A0	0.05	89.71$\pm$0.30	**88.08**$\pm$0.26
	0.10	**90.63**$\pm$0.26	**86.67**$\pm$0.28
	0.50	**89.96**$\pm$0.20	**84.28**$\pm$0.47
	BASE	89.59$\pm$0.27	83.92$\pm$0.38
MobileNetV2	0.05	88.32$\pm$0.21	**87.02**$\pm$0.38
	0.10	**91.14**$\pm$0.17	**85.10**$\pm$0.30
	0.50	**91.07**$\pm$0.21	**81.25**$\pm$0.41
	BASE	90.60$\pm$0.19	80.03$\pm$0.57
ResNeXt-50	0.05	**92.04**$\pm$0.22	**89.06**$\pm$0.24
	0.10	**92.02**$\pm$0.19	**86.10**$\pm$0.24
	0.50	**91.21**$\pm$0.47	80.31$\pm$0.53
	BASE	88.60$\pm$1.57	82.60$\pm$0.43

(b)

Dataset	Method	Learning Rate (α) 0.1	0.01
CIFAR10	RBN	**90.17**$\pm$0.22	84.72$\pm$0.29
	RBN$^-$	**90.11**$\pm$0.24	84.50$\pm$0.36
	IEBN	**90.18**$\pm$0.26	**85.34**$\pm$0.39
	IEBN$^-$	**90.15**$\pm$0.24	**85.29**$\pm$0.35
	Ours	**90.80**$\pm$0.20	**88.84**$\pm$0.32
	BASE	89.44$\pm$0.45	84.64$\pm$0.25
CIFAR100	RBN	**66.95**$\pm$0.57	**58.95**$\pm$0.42
	RBN$^-$	**66.82**$\pm$0.55	**58.90**$\pm$0.61
	IEBN	**66.94**$\pm$0.39	**60.61**$\pm$0.40
	IEBN$^-$	**66.95**$\pm$0.32	**60.89**$\pm$0.41
	Ours	**68.80**$\pm$0.49	**64.01**$\pm$0.54
	BASE	66.01$\pm$0.95	58.48$\pm$0.53
CUB-200	RBN	48.68$\pm$1.56	**44.68**$\pm$0.59
	RBN$^-$	47.14$\pm$2.72	**43.02**$\pm$1.22
	IEBN	**54.12**$\pm$0.60	**44.92**$\pm$0.74
	IEBN$^-$	**53.81**$\pm$0.76	**44.09**$\pm$0.65
	Ours	**58.52**$\pm$0.69	**45.31**$\pm$0.59
	BASE	46.26$\pm$1.59	41.61$\pm$1.03
ST-Cars	RBN	**68.17**$\pm$1.84	51.87$\pm$1.34
	RBN$^-$	**67.84**$\pm$2.96	52.30$\pm$1.73
	IEBN	**73.60**$\pm$0.92	51.06$\pm$0.87
	IEBN$^-$	**74.04**$\pm$1.55	51.08$\pm$0.78
	Ours	**78.29**$\pm$0.44	51.18$\pm$2.16
	BASE	64.73$\pm$2.87	51.86$\pm$1.80

based input normalization method can serve as a replacement for the traditional techniques, where it *automatically* handles input normalization and avoids the previously mentioned issues that may stem from employing the other techniques (Sect. 3.4).

4.5 Related Approaches

Lastly, we examined the existing related methods of Representative BN (RBN) [9] and Instance Enhancement BN (IEBN) [23] that utilize instance-specific statistics to compute an *extra* scaling parameter (as described in Sect. 2). For this work, we focused on the scaling aspect of RBN/IEBN for a more direct comparison of methods. The data normalization component for both approaches is the same as ours (which produces $\hat{X}$), but the affine transformation of RBN/IEBN is $Y = \gamma \cdot \mathcal{S} \cdot \hat{X} + \beta$ with $\mathcal{S} = sigmoid(w_v \cdot \text{GAP}(\mathcal{X}) + w_b)$, where GAP is the global average pooling operation, w_v and w_b are *additional* learnable weights (one of each per channel), and $\mathcal{S}$ is the result of their additional scaling method.

For RBN, $\mathcal{X} = \hat{X}$ (normalized features) and for IEBN $\mathcal{X} = X$ (raw input features). For RBN, the additional parameters w_v and w_b are initialized to 0 and 1, respectively, while IEBN initializes these parameters to $w_v = 0$ and w_b = -1. The standard BN parameters are initialized to $\gamma = 1$ and $\beta = 0$ for both approaches. Thus the resulting BN has an initial effective scale of $\gamma \cdot \mathcal{S} \approx 0.731$ for RBN and 0.269 for IEBN. Both approaches have smaller initial scale values (*i.e.*, <1) similar to ours, though these initial values are *fixed* for *all* scenarios. Related to our use of a reduced learning rate for γ, their use of a sigmoid will also force smaller gradient updates on w_v and w_b (that affect the resulting scale) since the derivative of a sigmoid function is $sigmoid(x) \cdot (1 - sigmoid(x))$.

We considered two versions of both approaches, one with the instance-specific statistics (RBN, IEBN) and one that removes the instance-specific statistics and w_v (RBN$^-$, IEBN$^-$). This modifies their scaling method to become $\mathcal{S} = sigmoid(w_b)$ and results in the same initial scale value as previously mentioned. Comparisons of RBN, RBN$^-$, IEBN, IEBN$^-$, and BASE with our approach for all of the datasets are presented in Table 2. We report results for our approach using the best performing γ initialization selected from $\gamma \in \{0.5, 0.1, 0.05\}$.

When evaluated against each other using a one-sided paired t-test (with a significance level of $p \leq 0.05$), our approach significantly outperformed RBN/RBN$^-$ and IEBN/IEBN$^-$ across all datasets and learning rates except for ST-Cars at the lower 0.01 learning rate, where no approach was significantly different from another. As mentioned, this learning rate is arguably insufficient for ST-Cars.

Our work has shown that *flexibility* in the initialization of γ coupled with a reduced learning rate is advantageous to achieve larger performance gains as compared to a *fixed* initial scale value for all situations. Furthermore, the minor differences in results between RBN/IEBN and RBN$^-$/IEBN$^-$ suggest that the contribution of instance-specific information is not as important as the scaling itself. However, it may be possible to incorporate instance-specific statistics with our approach for further gains.

5 Conclusion

We revisited BN to address the observed issues of learned BN parameters remaining close to initialization and passing forward overly large values from the normalization. We derived and empirically demonstrated across multiple datasets and network architectures that initializing the BN scale parameter $\gamma < 1$ and reducing the learning rate on γ can yield statistically improved performance over the default initialization and update strategy, suggesting that current training strategies are preventing BN from achieving the most optimal γ value. The proposed alterations do not structurally change BN (*i.e.*, no additional parameters) and can easily be applied to existing implementations. Additionally, we presented a special prepended BN to automatically handle input data normalization during training.

Acknowledgements. This research was supported by the U.S. Air Force Research Laboratory under Contract #GRT00054740 (Release #AFRL-2021-3711). We also thank the DoD HPCMP for the use of their computational resources.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., et al.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (2016)
2. Arpit, D., Zhou, Y., Kota, B., Govindaraju, V.: Normalization propagation: a parametric technique for removing internal covariate shift in deep networks. In: International Conference on Machine Learning (2016)
3. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
4. Bjorck, N., Gomes, C.P., Selman, B., Weinberger, K.Q.: Understanding batch normalization. In: Advances in Neural Information Processing Systems (2018)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)
6. Devore, J.L.: Probability & Statistics for Engineering and the Sciences, 8th edn. Brooks/Cole, Cengage Learning (2011)
7. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: RepVGG: making VGG-style ConvNets great again. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (2021)
8. Dosovitskiy, A., et al.: An image is worth 16×16 words: transformers for image recognition at scale. In: International Conference on Learning Representations (2021)
9. Gao, S.H., Han, Q., Li, D., Cheng, M.M., Peng, P.: Representative batch normalization with feature calibration. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (2021)
10. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: IEEE International Conference on Computer Vision (2015)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (2016)
12. Hoffer, E., Ben-Nun, T., Hubara, I., Giladi, N., Hoefler, T., Soudry, D.: Augment your batch: improving generalization through instance repetition. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (2020)
13. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: IEEE Conference on Computer Vision and Pattern Recognition (2017)
14. Huang, L., Qin, J., Liu, L., Zhu, F., Shao, L.: Layer-wise conditioning analysis in exploring the learning dynamics of DNNs. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12347, pp. 384–401. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58536-5_23
15. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning (2015)
16. Jia, S., Chen, D.J., Chen, H.T.: Instance-level meta normalization. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (2019)
17. Jones, D.: Good practice in (pseudo) random number generation for bioinformatics applications. Technical report, University College London (2010)
18. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3D object representations for fine-grained categorization. In: IEEE Workshop on 3D Representation and Recognition (2013)

19. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
20. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE (1998)
21. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.-R.: Efficient BackProp. In: Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 1524, pp. 9–50. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49430-8_2
22. Li, X., Sun, W., Wu, T.: Attentive normalization. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12362, pp. 70–87. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58520-4_5
23. Liang, S., Huang, Z., Liang, M., Yang, H.: Instance enhancement batch normalization: an adaptive regulator of batch noise. In: AAAI Conference on Artificial Intelligence (2020)
24. Luo, P., Wang, X., Shao, W., Peng, Z.: Towards understanding regularization in batch normalization. In: International Conference on Learning Representations (2019)
25. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems (2019)
26. Picard, D.: `torch.manual_seed(3407)` is all you need: on the influence of random seeds in deep learning architectures for computer vision. arXiv preprint arXiv:2109.08203 (2021)
27. PyTorch: Torch Generator. https://pytorch.org/docs/stable/generated/torch.Generator.html. Accessed Aug 2021
28. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: inverted residuals and linear bottlenecks. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018)
29. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? In: International Conference on Neural Information Processing Systems (2018)
30. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
31. Tan, M., Le, Q.: EfficientNet: rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning (2019)
32. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: the missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022 (2016)
33. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems (2017)
34. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The caltech-UCSD birds-200-2011 dataset. Technical report, California Institute of Technology (2011)
35. Wightman, R., Touvron, H., Jégou, H.: ResNet strikes back: an improved training procedure in timm. arXiv preprint arXiv:2110.00476 (2021)
36. Wu, Y., He, K.: Group normalization. Int. J. Comput. Vis. **128**(3), 742–755 (2019). https://doi.org/10.1007/s11263-019-01198-w
37. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (2017)
38. Xu, Y., et al.: Batch normalization with enhanced linear normalization. arXiv preprint arXiv:2011.14150 (2020)

39. Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., Schoenholz, S.S.: A mean field theory of batch normalization. In: International Conference on Learning Representations (2019)
40. Zhang, S., Nezhadarya, E., Fashandi, H., Liu, J., Graham, D., Shah, M.: Stochastic whitening batch normalization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2021)
41. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: IEEE International Conference on Computer Vision (2017)