Differentiable Dynamic Normalization for Learning Deep Representation

Ping Luo [*1,2] **Zhanglin Peng** [*3] **Wenqi Shao** [2,3] **Ruimao Zhang** [2,3] **Jiamin Ren** [3] **Lingyun Wu** [3]

Abstract

This work presents Dynamic Normalization (DN), which is able to learn arbitrary normalization operations for different convolutional layers in a deep ConvNet. Unlike existing normalization approaches that predefined computations of the statistics (mean and variance), DN learns to estimate them. DN has several appealing benefits. First, it adapts to various networks, tasks, and batch sizes. Second, it can be easily implemented and trained in a differentiable end-to-end manner with merely small number of parameters. Third, its matrix formulation represents a wide range of normalization methods, shedding light on analyzing them theoretically. Extensive studies show that DN outperforms its counterparts in CIFAR10 and ImageNet.

1. Introduction

Normalization approaches are indispensable components in recent deep neural networks (DNNs), such as batch normalization (BN) (Ioffe & Szegedy, 2015), layer normalization (LN) (Ba et al., 2016), instance normalization (IN) (Ulyanov et al., 2016), and group normalization (GN) (Wu & He, 2018). They are often stacked after each convolutional or fully-connected layer of a DNN to improve its optimization and generalization ability.

However, existing normalizers have two issues. First, they are designed manually for specific networks and tasks. For example, BN might improve optimization for convolutional neural networks (ConvNets) (Santurkar et al., 2018; Luo et al., 2019b), while LN facilitates gradient propagation in recurrent neural networks (RNNs) (Ba et al., 2016). Second, previous deep networks employed a single normalizer uniformly, leading to sub-optimal performance. For instance, BN is used after every convolutional layer of a ResNet101 (He et al., 2016), which contains a hundred of convolutional layers.

To address the above issues, this work presents Dynamic Normalization (DN), which learns arbitrary normalization operations for different layers of a DNN. DN has several appealing benefits. (i) *Diversity*. It not only represents existing normalizers including BN, IN, LN and GN, but also learns a wide range of variants of them from training data. For example, as shown in Fig.1(b), DN learns to divide a batch of training samples or a set of convolutional channels into any numbers of groups. This is contrary to previous methods that treated the number of groups as a hyper-parameter. (ii) *Easy to implement and use*. Although DN has rich representation capacity, it is simple to implement and train in a differentiable end-to-end manner. (iii) *Versatility*. DN is applicable in various networks, tasks, and batch sizes.

This work has four main **contributions**. (1) We study the problem of learning-to-normalize by Dynamic Normalization (DN), which is the first approach that learns arbitrary normalization formulations from data, without using hand-crafted normalization layers. DN pushes the frontier of normalization in deep learning. (2) Through careful investigation, DN is formulated in matrix notations that unify all previous normalizers, facilitating the usage and understanding of them. For example, DN enables geometric explanations of various approaches. We believe that DN opens up new research directions to analyze normalization methods in a holistic perspective. (3) A Kronecker decomposition of matrix is proposed, enabling DN to have low computation, small number of parameters, and simple implementation as much as possible. (4) Extensive studies in CIFAR10 (Krizhevsky, 2009) and ImageNet (Russakovsky et al., 2015) demonstrate that DN is able to outperform its counterparts.

1.1. Related Work

Normalization. The recent approaches normalize the first two moments of the hidden representations of DNNs. They applied Gaussian standardization to subtract mean of the representation and dividing the centered representation by the standard deviation. Different methods employ different scopes to estimate the statistics, such

[*]Equal contribution [1]Department of Computer Science, The University of Hong Kong [2]Department of Electronic Engineering, The Chinese University of Hong Kong [3]SenseTime Group Ltd. Correspondence to: Ping Luo <pluo.lhi@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

as BN (Ioffe & Szegedy, 2015), LN (Ba et al., 2016), and IN (Ulyanov et al., 2016) as shown in Fig.1(a). These methods are typically used individually and uniformly in a DNN, although they improve optimization and generalization (Santurkar et al., 2018; Teye et al., 2018). In contrast, switchable normalization (SN) (Luo et al., 2019a) selected different normalizers for different convolutional layers from a set of candidate methods (BN, IN, and LN). However, SN requires to enumerate a set of manually defined approaches, less representative than DN that provides a unified representation of normalization methods.

Network Architecture Search. DN is related to but different from neural architecture search (NAS) (Liu et al., 2018; Pham et al., 2018). In NAS, an architecture at time t was sampled according to $p(\mathcal{L})$, where $\mathcal{L}(\Phi_t, \Theta_t)$ is a loss function. Let Φ_t denote a set of 'control parameters' that describe the network architecture (*e.g.* operators), while let Θ_t represent a set of 'network parameters' corresponding to the proposed architecture (*e.g.* convolution kernels). The structural learning in NAS typically iterates two stages in turn, resembling a Markov decision process, where the operators $p(\Phi_{t+1}|\Phi_t)$ are searched on a validation set and then the network parameters Θ_{t+1} are optimized on a training set. However, unlike NAS, the 'control parameters' in DN represent arbitrary normalization methods, which are optimized jointly with the network parameters in a single stage by using only the training set in an end-to-end way.

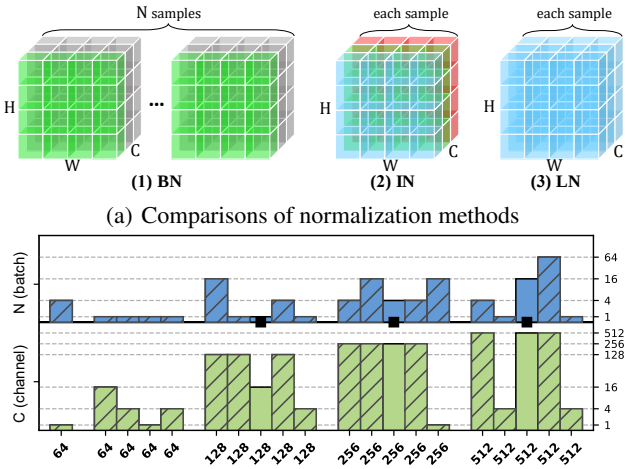
2. Notation and Background

This section introduces notations and background. In general, we denote a scalar by using lowercase letter (*e.g.* 'a') and a matrix by using capital letter (*e.g.* 'A') or bold lowercase letter (*e.g.* '$\mathbf{a}$'). We introduce normalization by using ConvNet as an example, while the discussions can be also applied to multilayer perceptrons (MLPs) and RNNs.

Overview. Let $F \in \mathbb{R}^{N\times C\times H\times W}$ be a 4D tensor of hidden feature maps, where four dimensions N, C, H and W denote mini-batch size, number of channels, height and width of a channel, respectively. Given a pixel $h \in F$, the recent approaches such as BN, IN, LN, and GN assume that h follows a normal distribution. They normalize h by removing its mean and standard deviation. These methods can be generally formulated as

$$\hat{h} = \frac{h-\mu^k}{\sqrt{(\sigma^k)^2+\epsilon}}, \text{ where } \mu^k = \frac{1}{|\Omega^k|}\sum_{(n,c,i,j)\in\Omega^k} F_{ncij}$$
$$\text{and } (\sigma^k)^2 = \frac{1}{|\Omega^k|}\sum_{(n,c,i,j)\in\Omega^k}(F_{ncij}-\mu^k)^2. \quad (1)$$

In Eqn.(1), $\hat{h}$ is the value of h after normalization, μ^k and σ^k are its mean and standard deviation computed by using a certain normalizer k, $k \in \{\text{BN}, \text{IN}, \text{LN}, \text{GN}\}$. Ω^k is a set of indices that indicates a set of pixels in F used to estimate μ^k and σ^k. For example, $F_{ncij} \in F$ is a pixel at location (i, j) in the c-th channel of the n-th sample. ϵ is a small constant to prevent dividing by zero. Furthermore, a scale parameter γ and a bias parameter β are adopted to rescale and reshift the normalized features, that is, $\gamma\hat{h}+\beta$. For simplicity of notations, these two parameters are not presented in the following narrations.



(a) Comparisons of normalization methods

(b) Learned numbers of groups in 20 DN layers of ResNet18

Figure 1. (a) $N, C, H,$ and W represent batch size, number of channels, height and width of a channel respectively. BN in (a.1) estimates statistics for each channel by averaging over a batch of N samples. IN in (a.2) normalizes each channel of each sample independently. LN in (a.3) estimates statistics for each sample by averaging over C channels. In (b), we show an example of ResNet18 trained with DN in CIFAR10. For a hidden layer, DN partitions $N = 128$ samples in a batch and C channels of a layer into different numbers of groups (see the right of y-axis). The number of channels of each layer is shown in x-axis (maximum 512 channels). For instance, the penultimate layer divides N and C into 64 and 512 groups. The performance of ResNet18 is improved by learning different normalizations for different layers. In (b), '■' indicates the layer with shortcut connection. The layers with kernel size 3 and 1 are indicated by bars with or without slash.

Different normalizers have different definitions of Ω^k in Eqn.(1). For instance, for BN, we have $\Omega^{\text{BN}} = \{(n,i,j)|n = 1...N,\ i = 1...H,\ j = 1...W\}$; for IN, $\Omega^{\text{IN}} = \{(i,j)|i = 1...H,\ j = 1...W\}$; and for LN $\Omega^{\text{LN}} = \{(c,i,j)|c = 1...C,\ i = 1...H,\ j = 1...W\}$.

By taking BN as an example, its mean is computed for each channel independently by $\mu_c^{\text{BN}} = \frac{1}{NHW}\sum_{(n,i,j)\in\Omega^{\text{BN}}} F_{ncij}$, implying that BN averages pixels across height H, width W, as well as batch size N. In contrast, IN treat each channel of each sample independently, and its mean is provided by $\mu_{nc}^{\text{IN}} = \frac{1}{HW}\sum_{(i,j)\in\Omega^{\text{IN}}} F_{ncij}$. The other statistics can be calculated similarly as above.

Switchable Normalization (SN). Unlike previous approaches that estimated statistics in different scopes, SN (Luo et al., 2019a; Shao et al., 2019) linearly combines the statistics of existing methods. The definition of SN is

$$\hat{h} = \frac{h - \sum_{k\in\{\text{BN,IN,LN,GN,...}\}} \lambda^k \mu^k}{\sqrt{\sum_{k\in\{\text{BN,IN,LN,GN,...}\}} \lambda^k (\sigma^k)^2 + \epsilon}}, \quad (2)$$

$$\text{where } \forall \lambda^k \in [0,1] \quad \text{and} \sum_{k\in\{\text{BN,IN,LN,GN,...}\}} \lambda^k = 1.$$

By comparing Eqn.(2) to Eqn.(1), every $\lambda^k \in [0,1]$ is an additional parameter that represents an important ratio of a normalizer. And the sum of all important ratios is 1. They are learned by using the softmax function. In SN, the important ratios for μ^k and σ^k can be different, but they are shared in Eqn.(2) to simplify the notations.

With Eqn.(2), SN inherits benefits of all the previous approaches by learning a weighted combination of their statistics. However, although it demonstrates superiority compared to individual normalizer (Luo et al., 2018), SN has an obvious drawback: it exhaustively enumerates a set of methods for different networks and tasks. Sec.3 resolves this challenge by presenting dynamic normalization.

3. Dynamic Normalization (DN)

DN enables to learn arbitrary forms of normalization.

Definition. DN can be written in matrix notations. Given the feature maps $F \in \mathbb{R}^{N\times C\times H\times W}$, DN is defined by

$$\hat{F} = \frac{F - \frac{1}{\mathrm{Z}_U} U\boldsymbol{\mu} V \frac{1}{\mathrm{Z}_V}}{\frac{1}{\mathrm{Z}_U} U\boldsymbol{\sigma} V \frac{1}{\mathrm{Z}_V}}. \quad (3)$$

Here, F and $\hat{F}$ are the feature maps before and after normalization, $U\boldsymbol{\mu}V \in \mathbb{R}^{N\times C}$ and $U\boldsymbol{\sigma}V \in \mathbb{R}^{N\times C}$ are two N-by-C matrixes representing the learned means and standard deviations respectively. In Eqn.(3), the matrix subtraction and division are applied elementwisely[1] on F. In particular, both $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are N-by-C matrixes, which are the statistics computed the same as IN. In other words, every entry $\mu^{\text{IN}}_{nc} \in \boldsymbol{\mu}$ and $\sigma^{\text{IN}}_{nc} \in \boldsymbol{\sigma}$ represent the statistics estimated independently for each channel c of each sample n. Furthermore, we have $V \in \{0,1\}^{C\times C}$ and $U \in \{0,1\}^{N\times N}$, which are two learnable binary matrixes. They are shared for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ to simplify notations. Moreover, Z_U and Z_V are two scalars, which denote the partition numbers to average the statistics. To ease understanding, we show the computational graph of the forward propagation in Fig.2.

By comparing Eqn.(3) and Eqn.(2), we see that DN learns a normalization method from data by transforming the mean and variance estimated for every channel in every sample. DN has richer representation capacity than SN.

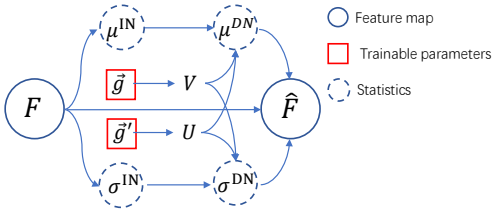


Figure 2. The computational graph of the forward propagation of DN. The learnable parameters are discussed in Sec.3.2.

Representation Capacity. In DN, V aggregates the statistics from the channels, while U aggregates those in a batch of samples. Therefore, different V and U represent different normalization approaches. To see this, we take $\frac{1}{\mathrm{Z}_U} U\boldsymbol{\mu}V \frac{1}{\mathrm{Z}_V}$ as an example to illustrate several special cases. (1) Let $V = I$, $U = I$, $\mathrm{Z}_V = 1$ and $\mathrm{Z}_U = 1$, where I is an identity matrix. DN represents IN by treating each channel as a group as shown in Fig.3(a), because $I\boldsymbol{\mu}I$ is the same as the means of IN. (2) Let $V = I$, $U = \mathbf{1}$, $\mathrm{Z}_V = 1$ and $\mathrm{Z}_U = \frac{1}{N}$, where $\mathbf{1}$ is a matrix of ones. DN turns into BN, since $\frac{1}{N} \cdot \mathbf{1}\boldsymbol{\mu}I$ indicates that the means from a certain channel of N samples are averaged[2]. (3) Let $V = \mathbf{1}$, $U = I$, $\mathrm{Z}_V = \frac{1}{C}$ and $\mathrm{Z}_U = 1$. DN becomes LN, as $I\boldsymbol{\mu}\mathbf{1} \cdot \frac{1}{C}$ averages the means from C channels of each sample. (4) By letting $U = I$ and V be a binary block diagonal matrix that divides the channels into groups, DN can represent GN with any numbers of groups as shown in Fig.3(b,c). Z_V represents the number of channels in each group.

Therefore, by changing V and U, DN covers a wide range of normalization methods, which produce different numbers of statistical values. To the extreme, when $V = \mathbf{1}$, $U = \mathbf{1}$, $\mathrm{Z}_V = \frac{1}{C}$ and $\mathrm{Z}_U = \frac{1}{N}$, DN represents "batch layer normalization" that has a unique value of mean and variance. In summary, DN divides the channels into groups by using V and divides a batch of samples into small batches by using U, where both V and U are binary block-diagonal matrixes. For example, DN is able to perform normalization by adjusting the batch size, leading to better generalization than pervious methods. It is established that computing statistics with a moderately small batch size increases regularization strength (Teye et al., 2018; Luo et al., 2019b).

3.1. DNs in Deep Networks

Let $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^P$ be a set of images and their ground-truth labels, $F^{(1)}, F^{(2)}, ..., F^{(L)}$ be the feature maps of L layers

[1]Note that we have $F \in \mathbb{R}^{N\times C\times H\times W}$, $U\boldsymbol{\mu}V \in \mathbb{R}^{N\times C}$, and $U\boldsymbol{\sigma}V \in \mathbb{R}^{N\times C}$. By default, the latter two matrixes would be repeated along the dimensions of H and W before normalization, to make their sizes the same as F.

[2]Similar to SN, $U\boldsymbol{\sigma}V$ in Eqn.(3) approximates the variance of a batch by using the average of variances of N samples, reducing computation while maintaining performance.

of a deep network. Let the network produce a predicted label $\bar{\mathbf{y}}_i$ given an image $\mathbf{x}_i$,

$$\bar{\mathbf{y}}_i = \text{relu}(\text{DN}(F^{(L)} \cdots \text{relu}(\text{DN}(F^{(1)}(\mathbf{x}_i))))). \quad (4)$$

Here the network has multiple DN layers. $F^{(1)}(\mathbf{x}_i)$ represents a linear transformation of the input layer. $\text{relu}(\cdot)$ is the ReLU activation function. With Eqn.(4), let Θ be a set of network parameters (*e.g.* filters) and $\Phi = \{V^\ell, U^\ell\}_{\ell=1}^L$ be a set of parameters of all the DN layers. These parameters are optimized by minimizing an empirical loss function $\frac{1}{P}\sum_{i=1}^P \mathcal{L}(\bar{\mathbf{y}}_i, \mathbf{y}_i; \Theta, \Phi)$. Eqn.(5) below shows that DN imposes binary block-diagonal constraints to the learnable matrixes V^ℓ and U^ℓ, which should have diagonal blocks of any sizes.

$$\begin{aligned} \arg\min_{\Theta,\Phi} \quad & \frac{1}{P}\sum_{i=1}^P \mathcal{L}(\bar{\mathbf{y}}_i, \mathbf{y}_i; \Theta, \Phi) \quad (5) \\ \text{subject to} \quad & \forall V^\ell \in \{0,1\}^{C^\ell \times C^\ell},\ \ell = 1...L \\ & \forall U^\ell \in \{0,1\}^{N^\ell \times N^\ell},\ \ell = 1...L \\ & \forall V^\ell, U^\ell \text{ are block diagonal matrixes.} \end{aligned}$$

Directly solving this discrete optimization problem is challenging, as the popular solver such as stochastic gradient descent (SGD) is insufficient to find a feasible solution, making DN difficult to use. Furthermore, to the extreme, all the DN layers introduce $(N^2 + C^2)L$ parameters, which are nonnegligible. For example, there are 100 million extra parameters, when a deep model has a hundred of normalization layers and each layer has a thousand of channels.

Instead of directly optimizing Eqn.(5), we explicitly construct feasible solutions of V^ℓ and U^ℓ in each forward propagation of a deep network, as discussed in Sec.3.2.

3.2. Binary Kronecker Decomposition

The forward propagation of DN is carefully designed by using matrix decomposition in order to have minimum number of parameters and fast computations.

We devise a matrix decomposition by using Kronecker products. By taking the C-by-C matrix V of a hidden layer as an example[3], V can be decomposed into a set of K small matrixes, that is, $\{V_i | V_i \in \mathbb{R}^{C_i \times C_i}, \forall C_i < C, \prod_{i=1}^K C_i = C\}$. We see that each matrix V_i is a C_i-by-C_i matrix where $C_i < C$. And the multiplications of $C_1, C_2, ..., C_K$ equal C. We define

$$V = f(V_1) \otimes \cdots \otimes f(V_i) \otimes \cdots \otimes f(V_K), \quad (6)$$

where $\otimes$ denotes a Kronecker product and $f(\cdot)$ denotes a function applied elementwisely to every entry of V_i. For example, when $f(a) = a$ is an identity function, Eqn.(6) becomes a sequence of the original Kronecker products.

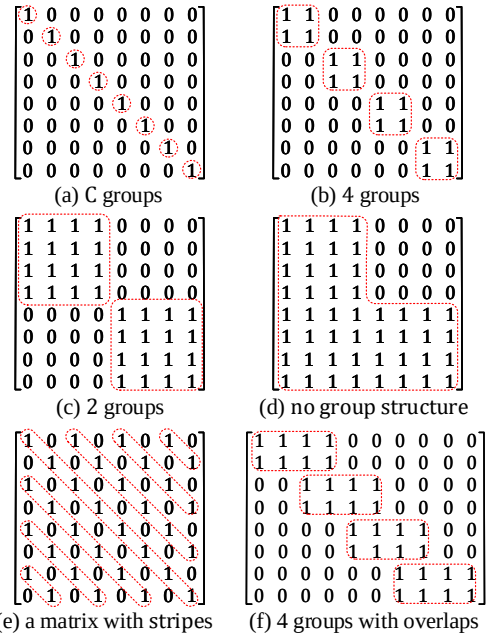


Figure 3. Illustrations of V and U where $V \in \mathbb{R}^{C \times C}$ partitions channels into groups, while $U \in \mathbb{R}^{N \times N}$ divides a batch of samples into groups. An 8-by-8 matrix V is shown as an example. (a,b,c) partition V into C, 4, and 2 groups respectively, while (d,e) are not block diagonal matrixes. (f) has overlapping groups.

A simple solution. Nevertheless, we are interested when $f(a) = \text{sign}(a)$ is a sign function[4]. Eqn.(6) turns into $V = \text{sign}(V_1) \otimes \text{sign}(V_2) \otimes \cdots \otimes \text{sign}(V_K)$, where the binary matrix V can be partitioned into a series of real matrixes $\{V_i\}_{i=1}^K$, which take continuous values. In this case, these matrixes can be learned by using SGD regardless of the binary constraints. In other words, learning a large C-by-C binary matrix V in Eqn.(5) becomes learning a set of small real matrixes as above, reducing the number of parameters from C^2 to $\sum_{i=1}^K C_i^2$. For example, an 8-by-8 matrix with 4 groups as shown in Fig.3(b) can be built by using three small 2-by-2 matrixes, *i.e.* $I \otimes I \otimes \mathbf{1}$, reducing the number of parameters from 64 to $3 \times 2^2 = 12$. Different matrix decompositions lead to different numbers of groups. As shown in Fig.3(c), the one with 2 groups can be built by using $I \otimes \mathbf{1} \otimes \mathbf{1}$.

However, using the sign function directly in Eqn.(6) still possesses two issues. (1) *Parameter Size.* It has number of parameters scaled square of the sizes of the submatrixes, that is, $\sum_{i=1}^K C_i^2$. (2) *Infeasible Solution.* As $\{V_i\}_{i=1}^K$ have continuous values, their compositions do not necessarily produce a block-diagonal matrix after applying the sign function. For example, switching a single entry in I of

[3]The superscript ℓ is omitted to simplify notations.

[4]We define $\text{sign}(a) = 1$ if $a \geq 0$ and $\text{sign}(a) = 0$ if $a < 0$.

$I \otimes \mathbf{1} \otimes \mathbf{1}$ from 0 to 1 already changes the matrix in Fig.3(c) to the one in (d), which does not have diagonal blocks.

3.3. Computations of V and U

Re-parameterization. To resolve the above issues, we reparameterize V_i by choosing $f(\cdot)$ as a binary gated function instead of a simple sign function. Therefore, $f(V_i)$ in Eqn.(6) is written as

$$f(V_i) = \vec{g}_i \mathbf{1} + (1 - \vec{g}_i) I, \ \forall \vec{g}_i \in \vec{\mathbf{g}}, \quad (7)$$

$$\text{where } \vec{\mathbf{g}} = P\mathbf{g} \text{ and } \mathbf{g} = \text{sign}(\tilde{\mathbf{g}}). \quad (8)$$

In these equations, $\tilde{\mathbf{g}}$ is a K-by-1 vector of gates that are continuous learnable parameters. In Eqn.(8), $\mathbf{g} \in \{0, 1\}^{K \times 1}$ represents a vector of binary gates obtained by applying the sign function on $\tilde{\mathbf{g}}$. P represents a constant permutation matrix[5], which permutates the elements of $\mathbf{g}$ to produce the permutated gates $\vec{\mathbf{g}}$. In Eqn.(7), $\mathbf{1}$ is a constant 2-by-2 matrix of ones and I is a constant 2-by-2 identity matrix. Each $\vec{g}_i \in \{0, 1\}, i = 1...K$ is a binary gate that belongs to $\vec{\mathbf{g}}$.

Eqn.(7) and (8) show that a block-diagonal matrix can be constructed by using a series of Kronecker products involving $\mathbf{1}$ and I with certain permutation that is defined by P. They reparameterize $\{V_i\}_{i=1}^K$ by learning $\tilde{\mathbf{g}}$, instead of learning the matrixes directly. In this case, the number of parameters in DN is reduced from $\sum_{i=1}^K C_i^2$ to K parameters, $K = \log_2 C$. In other words, the parameter sizes to compute V and U of DN are merely scaled logarithmically with respect to the number of channels and the batch size, that is, $\log_2 C$ and $\log_2 N$. For instance, given a hidden layer with $C = 1024$ channels (a common setting in the recent deep models), the C-by-C matrix V can be learned by using just 10 parameters.

Permutated Gates. To produce a binary block-diagonal matrix V in Eqn.(6), it is important to have a specific permutation of the matrixes $\{V_i\}_{i=1}^K$. This is achieved by using P in Eqn.(8), which sorts the elements of $\mathbf{g}$ according to their values in an ascending order, implying that the value of 0 would present before the value of 1 in $\vec{\mathbf{g}}$. That is, for all i, j and $i < j$, we have $\vec{\mathbf{g}}_i \leq \vec{\mathbf{g}}_j$. For example, when $\vec{g}_i = 0$, $f(V_i) = I$; otherwise when $\vec{g}_i = 1$, $f(V_i) = \mathbf{1}$. This means that the matrix I would present before $\mathbf{1}$ in Eqn.(6), always producing a matrix that is binary block diagonal. For instance, when $\mathbf{g} = [1, 1, 0]$, we have $\vec{\mathbf{g}} = P\mathbf{g} = [0, 1, 1]$, producing a matrix $I \otimes \mathbf{1} \otimes \mathbf{1}$ as shown in Fig.3(c). In contrast, if P is not presented, $\vec{\mathbf{g}} = \mathbf{g} = [1, 1, 0]$ and the matrix becomes $\mathbf{1} \otimes \mathbf{1} \otimes I$, which has multiple stripes as shown in Fig.3(e).

[5] P is a square binary matrix that has exactly one entry of 1 in each row and each column, and 0s elsewhere.

Algorithm 1 Computations of DN

1: **Input:** feature map $F \in \mathbb{R}^{N \times C \times H \times W}$; parameters to be learned: $\tilde{\mathbf{g}}^V \in \mathbb{R}^{\log_2 C \times 1}$, $\tilde{\mathbf{g}}^U \in \mathbb{R}^{\log_2 N \times 1}$, $\boldsymbol{\gamma} \in \mathbb{R}^{C \times 1}$, and $\boldsymbol{\beta} \in \mathbb{R}^{C \times 1}$; initialize: $\tilde{\mathbf{g}}^V = 0$, $\tilde{\mathbf{g}}^U = 0$, $\boldsymbol{\gamma} = 1$, $\boldsymbol{\beta} = 0$.
2: **Output:** feature map $\hat{F} = \text{DN}(F)$ after normalization.
3: $\forall \mu_{nc}^{\text{IN}} = \frac{1}{HW} \sum_{i=1,j=1}^{H,W} F_{ncij}$, $\boldsymbol{\mu} \leftarrow \mu_{nc}^{\text{IN}}$
4: $\forall \sigma_{nc}^{\text{IN}} = \sqrt{\frac{1}{HW} \sum_{i=1,j=1}^{H,W} (F_{ncij} - \mu_{nc}^{\text{IN}})^2 + \epsilon}$, $\boldsymbol{\sigma} \leftarrow \sigma_{nc}^{\text{IN}}$
5: compute V and U by applying Eqn.(6-8)
6: $\hat{F} = \boldsymbol{\gamma} \frac{F - \frac{1}{Z_U} U \boldsymbol{\mu} V \frac{1}{Z_V}}{\frac{1}{Z_U} U \boldsymbol{\sigma} V \frac{1}{Z_V}} + \boldsymbol{\beta}$

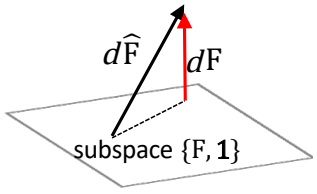


Figure 4. By analyzing DN, we are able to obtain a unified geometric view of the gradients of BN, LN, and GN. In these normalization layers, the gradients propagated to their inputs F, denoted as dF, are the projection of the back-propagated gradients from the above layer, denoted as $d\hat{F}$, to a subspace spanned by the input F and a vector of ones.

3.4. Discussions

Implementation. Algorithm 1 summarizes the implementations of DN by following Eqn.(3) and Eqn.(6-8). Specifically, $\tilde{\mathbf{g}}^V$ and $\tilde{\mathbf{g}}^U$ are the learnable gate parameters of V and U respectively. $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are the scale and shift parameters, which are applied after normalization as shown at the 6^{th} line. DN can be easily implemented by using the recent platforms such as TensorFlow and PyTorch.

In training, the backward propagation can be simply achieved by auto differentiation in the above platforms, and the gradient of the sign function is obtained by following (Rastegari et al., 2016). In the testing stage, the statistics are estimated online for each sample when $U = I$. Otherwise, we process multiple mini-batches and average over them. This is similar to the batch average used in BN and SN.

Geometric Interpretation. In general, as DN provides a holistic representation of normalization approaches, our derivations in the supplementary material show clear geometric meanings of BN, LN, and GN in a unified view, as shown in Fig.4. For example, we demonstrate that the back-propagated gradients of BN, LN, and GN are projections onto the orthogonal complementary subspaces spanned by vectors of ones and their layer inputs. These results are not achievable by previous work such as (Ioffe & Szegedy, 2015; Ba et al., 2016; Tian, 2018), which analyzed normalization methods separately.

Extensions. DN can be extended in several ways. In the first aspect, the matrixes $\mathbf{1}$ and I in Eqn.(7) can have any scale rather than 2-by-2, producing groups with different sizes. We can also have overlapping groups. For instance, let $A = [1, 1, 0; 0, 1, 1]$ be a 2-by-3 matrix and $\mathbf{1}$ be a 2-by-2 matrix, and then $A \otimes \mathbf{1}$ produces groups with two channels overlapped as shown in Fig.3(f). In the second aspect, we may combine DN with prior knowledge by introducing two smooth terms I and $\mathbf{1}$ into V and U respectively, such as $(U + \mathbf{1})\boldsymbol{\mu}(V + I)$. Intuitively, I treats each channel as a group, while $\mathbf{1}$ treats an entire batch as a group. For example, the learned means can be expressed by four terms, $U\boldsymbol{\mu}V + U\boldsymbol{\mu}I + \mathbf{1}\boldsymbol{\mu}V + \mathbf{1}\boldsymbol{\mu}I$, which represent a DN, a 'grouped BN', a 'batch GN', and a BN respectively. With prior knowledge, we can implicitly combine various normalization methods.

4. Experiments

Sec.4.1 evaluates DN in CIFAR10 (Krizhevsky, 2009) and ImageNet (Russakovsky et al., 2015), where it is demonstrated by comparing with previous normalization techniques. Ablation studies are presented in Sec.4.3. The details of all experimental setups are presented in supplementary material.

4.1. Image Recognition in CIFAR10

Batch sizes. Table 1 compares DN with three representative normalizers, including BN (Ioffe & Szegedy, 2015), GN (Wu & He, 2018), and SN (Luo et al., 2019a). All models are trained on CIFAR10 with different batch sizes, where the gradients are aggregated across GPUs, while the statistics are estimated within each GPU. We see that DN works well in a wide range of batch sizes, outperforming the other methods. We repeat to train all models five times and the standard deviation of the accuracy is smaller than 0.15%, rendering significance of the results.

In particular, by comparing $(8, 8)$, $(4, 8)$ and $(2, 8)$ when the number of GPUs decreases from 8 to 2, we observe that the accuracies slightly increase. We conjecture that reducing the sample size to aggregate the gradients would increase randomness and regularization when training in CIFAR10. By comparing $(8, 8)$, $(8, 4)$ and $(8, 2)$ when the mini-batch size used to estimate the statistics decreases from 8 to 2, it is seen that DN is more robust in small mini-batch. Although GN is independent with mini-batch size, its accuracies are lower than the other methods. Moreover, we see that GN prefers different group numbers for different batch sizes, making hyper-parameter tuning cumbersome.

Architectures. Table 2 reports classification accuracies of different networks trained with DN, BN, SN, and GN on CIFAR10. We observe that DN performs better than the other methods. In addition, we found that BN and SN achieve reasonable results, whereas the performances are impeded by GN, especially when the depth of network is increased.

	(1,128)	(8,8)	(8,4)	(4,8)	(8,2)	(2,8)
BN	**94.80**	93.31	93.01	**94.18**	91.55	**94.84**
GN_{32}	93.67†	90.22†	90.58	92.66†	90.85	93.65†
GN_{16}	93.17	89.49	90.90†	92.32	90.89†	93.21
GN_{8}	93.33	89.52	90.00	91.92	90.06	92.93
SN	94.40	**93.33**	**93.10**	93.87	**92.38**	94.26
DN	**94.98**	**93.81**	**93.45**	**94.67**	**92.45**	**94.95**

Table 1. **Comparisons on CIFAR10.** The classification accuracies (%) are reported on the test set. ResNet18 is trained with BN, GN, SN, and DN by using different batch sizes. The bracket $(\cdot, \cdot)$ denotes (#GPUs, #samples per GPU). GN is trained with different number of groups denoted by the subscripts, where the best result of GN is marked by †. The two best-performing results of each column are shown in bold.

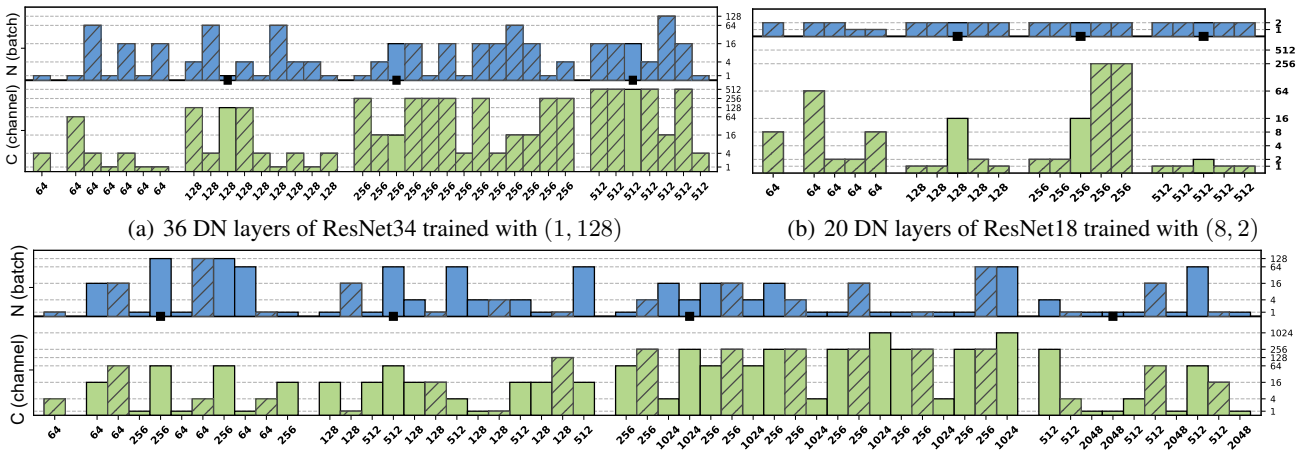
	BN	GN_{32}	GN_{16}	GN_{8}	SN	DN
ResNet18	94.80	93.67	93.17	93.33	94.40	**94.98**
ResNet34	95.16	93.79	93.63	92.43	94.59	**95.35**
ResNet50	95.61	90.18	92.51	91.67	94.72	**95.81**

Table 2. **Comparisons on CIFAR10.** The classification accuracies (%) are reported on the test set. ResNet18, ResNet34, and ResNet50 are trained with BN, GN, SN, and DN by using $(1, 128)$ respectively. The best accuracy of each network architecture (in each row) is shown in bold.

Furthermore, Fig.1(b) and Fig.5(a,c) illustrate the learned numbers of groups for all the DN layers in three networks that are trained by using $(1, 128)$. We see that the DN layers learn diverse normalizers, rather than BN, GN, and LN. For example, we point out two extreme cases.

First, when N samples and C channels are both treated as a single group, DN learns a batch layer normalization, as shown in the 4^{th} layer of Fig.1(b), the 4^{th} and 6^{th} layer of Fig.5(a). Second, when N groups and C groups are presented in a layer, DN becomes IN. However, this case does not exist in the above models, showing that IN is not desirable when the batch has normal size. This observation is consistent with (Luo et al., 2019a). Apart from the above cases, the 5^{th} and 7^{th} layer in Fig.5(c) are GNs with 64 and 4 groups respectively, while the penultimate layer of Fig.5(a) resembles a grouped BN with its batch partitioned into 16 groups.

Another interesting phenomenon is provided in Fig.5(b), where plots the learned groups of ResNet18 trained with small batch size $(8, 2)$. For example, we see that 18 DN layers treat each sample as a group (*i.e.* 2 groups for the N dimension), 2 layers treat each channel as a group (*i.e.* 256 groups for the C dimension), and 7 layers treat all channels as a single group. They result in 2 INs, 7 LNs, and the remains are GNs. This phenomenon is consistent with (Wu & He, 2018; Luo et al., 2019a) when the batch size is small, where IN, LN, and GN could be used to reduce dependency



(a) 36 DN layers of ResNet34 trained with $(1, 128)$

(b) 20 DN layers of ResNet18 trained with $(8, 2)$

(c) 53 DN layers of ResNet50 trained with $(1, 128)$

Figure 5. Learned numbers of groups for each DN layer in different networks, including (a) ResNet34, (b) ResNet18, and (c) ResNet50, which are trained with different batch sizes in CIFAR10. The y-axis shows that each DN layer partitions the N training samples in a batch as well as the C channels into groups. The x-axis plots the number of channels for each layer. Specifically, (a,c) are trained with $(1, 128)$ *i.e.* $N = 128$ samples in a batch, while (b) is trained with $(8, 2)$ *i.e.* $N = 2$ samples in a batch. In (a,b,c), N can be divided into maximum 128, 2, and 128 groups respectively, while C can be divided into maximum 512, 512, and 2048 groups respectively. In the above figures, '■' indicates the layer with shortcut connection, and the layers with kernel size 3 and 1 are indicated by bars with or without slash.

on batch size.

4.2. ImageNet

Table 3 reports results on ImageNet, where DN performs well in both architectures. This trend is similar to CIFAR10. In ResNet101, DN outperforms SN and BN by 0.8% and 1.4%, showing its benefits in large-scale dataset. We see that IN (Ulyanov et al., 2016) and LN (Ba et al., 2016) are not optimal for image classification. For example, ResNet101 trained with IN and LN achieve 72.2% and 75.3% respectively, which are reduced by 7.0% and 3.9% compared to 79.2% of DN.

In Table 3, BRN (Ioffe, 2017) has two hyper-parameters that renormalize the means and variances. We choose their best settings to obtain 76.3% and 78.1% for BRN, which reduces 1.9% and 1.1% compared to DN. Moreover, BKN (Wang et al., 2018) estimated the statistics in a certain layer by combining those estimated in the preceding layers. BKN obtains 76.8% and 78.3% that are 1.4% and 0.9% drop compared to DN. In addition, the connections between layers in BKN are carefully designed for every specific network, making it difficult to use in practice.

4.3. DN Analysis

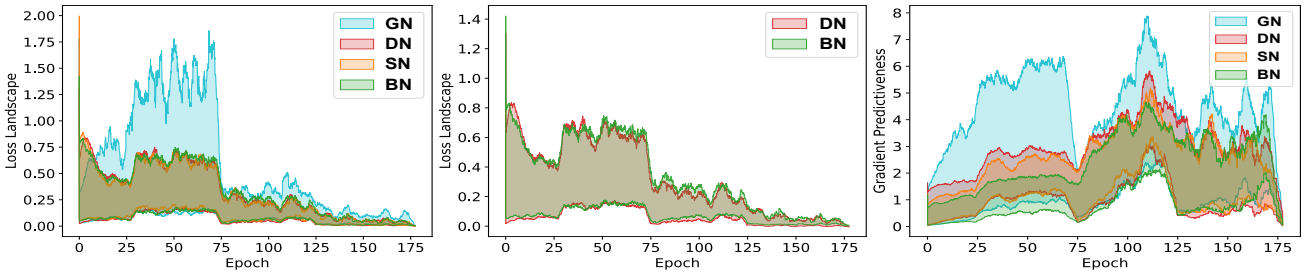
This section investigates characteristics of DN as well as how it helps training deep models.

	BN	GN	LN	IN	SN	BRN	BKN	DN
ResNet50	76.4	75.9	74.7	71.6	76.9	76.3	76.8	**78.2**
ResNet101	77.8	77.6	75.3	72.2	78.4	78.1	78.3	**79.2**

Table 3. **Comparisons on ImageNet.** The top-1 classification accuracies (%) on the validation set are reported. ResNet50 and ResNet101 are trained with BN, GN, LN, IN, SN, BRN (Ioffe, 2017), BKN (Wang et al., 2018), and DN. The batch size is $(8, 32)$. The best accuracy of each network architecture (in each row) is shown in bold.

Learning dynamics. For every DN layer of ResNet18, we plot the learning procedures of the numbers of groups and the values of their corresponding gates as shown in Fig.7. We have two observations. First, different DN layers have different learning dynamics, which are smoothly converged in training. Second, lower layers (*e.g.* $C = 64$) prefer smaller numbers of groups as shown in (a), which is corresponding to the positive gates in (b). Whereas larger number of groups (negative gates) are typically presented in higher layers (*e.g.* $C > 64$). This is consistent with (Luo et al., 2018; 2019b), showing that upper layers would prefer statistics of each instance.

Optimization Landscape. To compare the impact of different normalizers on the stability of optimization, we follow (Santurkar et al., 2018) to study the "Lipschitzness" of the loss function by computing its gradient at an update step and measuring how the loss changes along that gradient direction (a slice of the loss landscape). Fig.6(a) shows that



(a) variations of loss values (b) variations of losses of DN *v.s.* BN (c) variations of gradient values

Figure 6. (a) Analyses of the loss landscapes of ResNet18 trained with different methods on CIFAR10. The x-axis is number of epochs while the y-axis plots variances of the loss values, which are estimated by using different step sizes (learning rates) along the gradient direction at each update step. (b) compares the variances of losses of DN and BN, where DN has slightly smoother losses than BN after 45 training epochs. (c) calculates the variances of the gradients similar to (a). GN has larger variations of the gradients than the other methods.

GN has wider range of loss values in the gradient direction than the others, while DN, SN, and BN produce comparable smoothing effects on the loss landscape. Similarly, we make analogous measurement for the gradient values along the gradient direction as shown in Fig.6(c). It is also observed that GN has large variation of gradient values, while the others comparably stabilize the gradients.

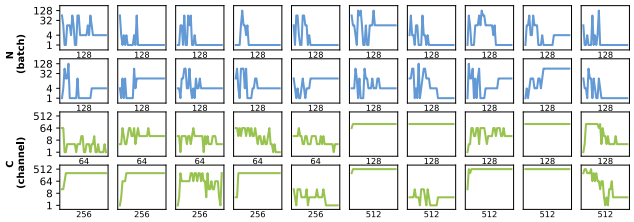
The above smoothing effects impact training in several ways. First, the smoothed loss landscape makes SGD stable, such as avoiding exploding or vanishing gradients. Second, more smooth losses and gradients are less sensitive to the choice of learning rate and initialization.

Internal Covariance Shift (ICS). We also compare the Internal Covariance Shift (ICS) of DN with BN, SN, and GN in the supplementary material. We see that GN has higher ICS than the other methods. Whereas DN has slightly higher ICS than SN and BN, but DN performs better in terms of accuracy, suggesting that normalization method does not necessarily reduce ICS as also discussed in (Santurkar et al., 2018).

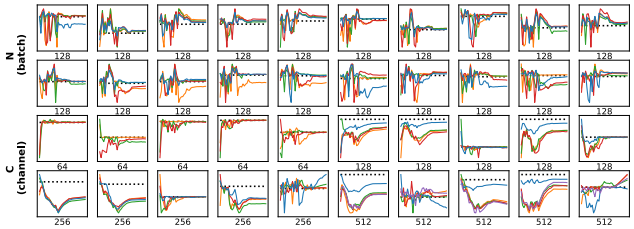
5. Conclusions and Future Work

This work proposed Dynamic Normalization (DN), the first approach that learns arbitrary normalization forms in data- and task-driven way for deep neural networks, without using manually designed normalization layers. DN offers a holistic formulation by representing not only BN, IN, LN and GN, but also a wide range of variants of them, for example, 'layer batch normalization'. Through extensive experiments and studies, DN not only adapts to various networks, batch sizes, and tasks outperforming its counterparts, but also improves training by smoothing the optimization landscape.

DN also opens up a new research direction to theoretically understand normalization methods in a unified view.



(a) Group numbers of 20 DN layers when training ResNet18



(b) Gate values ($\tilde{\mathbf{g}}$) corresponded to 20 DN layers as shown in (a)

Figure 7. The numbers of groups of N and C during training ResNet18 on CIFAR10 with $(1, 128)$ are shown in (a). The corresponding gate values ($\tilde{\mathbf{g}}$) are shown in (b), where the dashed lines represent values of zeros. In (b), when a continuous gate in $\tilde{\mathbf{g}}$ is positive, its corresponding binary gate in $\mathbf{g}$ equals one; otherwise, the binary gate is zero. The number of channels is plotted for each layer (in the bottom).

For example, we show the geometric meanings of many normalizers, where the gradient of a normalization layer, which would be propagated down to its preceding layer, is achieved by projecting its input gradient onto the orthogonal complementary space spanned by its layer inputs and a vector of ones. Understanding the learning dynamics and generalization of different normalization methods by using DN would be an important future direction.

DN might also combine with existing whitened neural networks such as (Desjardins et al., 2015; Luo, 2017b;a), as well as facilitate the investigation of whitening approaches like switchable whitening (Pan et al., 2019).

References

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

Desjardins, G., Simonyan, K., Pascanu, R., and Kavukcuoglu, K. Natural neural networks. *NIPS*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Ioffe, S. Batch renormalization: Towards reducing mini-batch dependence in batch-normalized models. *arXiv:1702.03275*, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Krizhevsky, A. Learning multiple layers of features from tiny images. *Technical report*, 2009.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2018.

Luo, P. Eigennet: Towards fast and structural learning of deep neural networks. *IJCAI*, 2017a.

Luo, P. Learning deep architectures via generalized whitened neural networks. *ICML*, 2017b.

Luo, P., Peng, Z., Ren, J., and Zhang, R. Do normalization layers in a deep convnet really need to be distinct? In *arXiv:1811.07727*, 2018.

Luo, P., Ren, J., Peng, Z., Zhang, R., and Li, J. Differentiable learning-to-normalize via switchable normalization. *ICLR*, 2019a.

Luo, P., Wang, X., Shao, W., and Peng, Z. Towards understanding regularization in batch normalization. *ICLR*, 2019b.

Pan, X., Zhan, X., Shi, J., Tang, X., and Luo, P. Switchable whitening for deep representation learning. In *arXiv:1904.09739*, 2019.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv:1802.03268*, 2018.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization? In *NIPS*, 2018.

Shao, W., Meng, T., Li, J., Zhang, R., Li, Y., Wang, X., and Luo, P. Ssn: Learning sparse switchable normalization via sparsestmax. In *CVPR*, 2019.

Teye, M., Azizpour, H., and Smith, K. Bayesian uncertainty estimation for batch normalized deep networks. In *ICML*, 2018.

Tian, Y. A theoretical framework for deep locally connected relu network. In *arXiv:1809.10829*, 2018.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016.

Wang, G., Peng, J., Luo, P., Wang, X., and Lin, L. Batch kalman normalization: Towards training deep neural networks with micro-batches. *NIPS*, 2018.

Wu, Y. and He, K. Group normalization. *ECCV*, 2018.