# **BCN: Batch Channel Normalization for Image Classification**

Afifa Khaled

#### Chao Li

Jia Ning

afifakhaied@tju.edu.cn

D201880880@hust.edu.cn

ninja@hust.edu.cn

#### Kun He

brooklet60@hust.edu.cn

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

# **Abstract**

Normalization techniques have been widely used in the field of deep learning due to their capability of enabling higher learning rates and are less careful in initialization. However, the effectiveness of popular normalization technologies is typically limited to specific areas. Unlike the standard Batch Normalization (BN) and Layer Normalization (LN), where BN computes the mean and variance along the (N,H,W) dimensions and LN computes the mean and variance along the (C,H,W) dimensions (N,C,H,W)H and W are the batch, channel, spatial height and width dimension, respectively), this paper presents a novel normalization technique called Batch Channel Normalization (BCN). To exploit both the channel and batch dependence and adaptively and combine the advantages of BN and LN based on specific datasets or tasks, BCN separately normalizes inputs along the (N, H, W) and (C, H, W) axes, then combines the normalized outputs based on adaptive parameters. As a basic block, BCN can be easily integrated into existing models for various applications in the field of computer vision. Empirical results show that the proposed technique can be seamlessly applied to various versions of CNN or Vision Transformer architecture. The code is publicly available at https://github.com/AfifaKhaled/Batch-Channel-Normalization.

#### 1. Introduction

In the past decades, machine learning (ML) has become the most widely used technique in the field of artificial intelligence, and more recently, deep learning (DL) has become a prevalent topic, and deep neural networks (DNNs) are widely applied in various domains, including natural language processing, computer vision, and graph mining. Typically, DNNs comprise stacked layers with learnable parameters and non-linear activation functions. While the deep

and complex structure enables them to learn intricate features, it also poses challenges during training due to the randomness in parameter initialization and changes in input data, known as internal covariate shift [11]. This problem becomes more pronounced in deeper networks, where slight modifications in deeper hidden layers are amplified as they propagate through the network, resulting in significant shifts in these layers.

To address the above issue, several normalization methods have been introduced. Specifically, Batch Normalization (BN) [11], Layer Normalization (LN) [3], and Group Normalization (GN) [22] *et al.* have achieved remarkable success with deep learning models. Among them, BN is widely used for deep neural networks.

Despite the great success in many applications, the popular normalization methods still have some weaknesses. For example, BN requires large batch sizes [22], can not be used for online learning tasks, and can not be used for large distributed models because the mini-batches have to be small. To address these issues, LN is proposed to avoid exploiting batch dimension so as to not impose any restriction on the size of each mini-batch [3]. However, LN does not work as well as BN on convolutional layers.

To overcome the limitations of BN and LN as well as to fully embody the advantages of the two techniques, we develop a new normalization technique called Batch Channel Normalization (BCN). In contrast to previous techniques, we aim to normalize along the (C, N, H, W) axes. However, computing the average and variance along (N, C, H, W) directly ignores the different importance between batch dimension and channel dimension. Consequently, as shown in Figure 1, BCN first computes the  $\mu_1$  and  $\sigma_1^2$  of the layer inputs along the (C, H, W) axes. Then, it computes the  $\mu_2$  and  $\sigma_2^2$  along the (L, H, W) axes. Finally, the normalized outputs are combined based on adaptive parameters. To check the effectiveness of the proposed method, we apply BCN to several popular models, ResNet [7], DenseNet [9], Vi-

sion Transformer [5] and BYOL [6], on the image classification task. Our experiments demonstrate that BCN yields promising results, leading to improved training speed and enhanced generalization performance.

Our main contributions are summarized as follows:

- We introduce a new normalization technique termed Batch Channel normalization (BCN) as a simple alternative to BN and LN techniques.
- BCN exploits the channel and batch dependence, and adaptively combines the information of channel dimension and batch dimension, which embodies the advantages of both BN and LN.
- Empirically, we show that our BCN normalization technique can substantially improve the generalization performance of the neural networks compared to existing normalization techniques.

### 2. Related Work

# 2.1. Dimension Normalization

The first group involves normalizing different dimensions of the output. Examples include Layer Normalization [3], which normalizes inputs across features; Instance Normalization [20], which normalizes over spatial locations in the output; and Group Normalization [22], which independently normalizes along spatial dimensions and feature groups.

Here we introduce two that are most related to our work, *i.e.*, Batch Normalization [11] and Layer Normalization [3].

**Batch Normalization (BN)** allows faster convergence and stabilizes the learning. During the training set, BN computes the mean  $\mu_B$  and variance  $\sigma_B^2$  of the layer inputs as follows:

$$\mu_B = \frac{1}{n} \sum_{i=1}^{n} x_i,$$
 (1)

$$\sigma_B^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_B)^2,$$
 (2)

$$\bar{x}_B = \gamma \frac{(x_i - \mu_B)}{\sqrt{(\sigma_B^2 + \epsilon)}} + \beta. \tag{3}$$

We can see that BN computes the  $\mu_B$  and  $\sigma_B^2$  along the (N, H, W) axes [11]. During the testing, BN computes the  $\mu_B$  and  $\sigma_B^2$  by exponential moving average during the training set:

$$\mu = \alpha \mu + (1 - \alpha)\mu_B,\tag{4}$$

$$\bar{x} = \gamma \frac{(x_i - \mu)}{\sqrt{(\sigma^2 + \epsilon)}} + \beta,$$
 (5)

where n is batch size,  $\gamma$  and  $\beta$  are learnable parameters. Here  $\alpha$  is usually set to 0.9, and  $\epsilon$  is a small constant.

**Layer Normalization (LN)** computes the  $\mu_L$  and  $\sigma_L^2$  along the (C, H, W) as follows:

$$\mu_L = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{6}$$

$$\sigma_L^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2, \tag{7}$$

$$\bar{x}_L = \gamma \frac{(x_i - \mu_L)}{\sqrt{(\sigma_L^2 + \epsilon)}} + \beta. \tag{8}$$

Unlike BN, LN performs the same computation at training and inference times. Moreover, LN is very useful for stabilizing the dynamics of hidden states in recurrent neural networks.

Our method belongs to this group. The proposed BCN aims to normalize along to (C, N, H, W) axes, which aims to combine the benefits of both BN and LN while mitigating their respective deficiency.

### 2.2. Normalization Improvement

The second group modifies the original batch normalization method [11]. This group includes methods like Ghost BN [8], which normalizes independently across different splits of batches, and Batch Re-normalization [10] or Streaming Normalization [13], both of which make changes to utilize global averaged statistics instead of batch statistics.

While these normalization techniques are practically famous and successful, their improvement has only started to appear recently. At the same time, Batch normalization [11] remains the most famous normalization technique used so far. In addition, these normalization techniques have not been able to approach BN's accuracy in many tasks (e.g., segmentation, detection, and video classification).

#### 2.3. Weight Normalization

The third group consists of methods that normalize weights rather than activations. This group comprises Weight Normalization [19] and Normalization Propagation [2], both of which divide weights by their  $\ell_2$  norm, differing only in minor details.

Several papers have recently proposed techniques to enhance the weight normalization across a wider range of CNN. One approach is to implicit regularization and convergence for weight normalization [21]. They studied the weight normalization technique and reparametrized projected gradient descent for over-parameterized least squares regression. They showed that the non-convex formulation has useful regularization effects compared to gradient descent on the original objective. Due to the limited number of pages, we refer the reader to read more normalization techniques at https://github.com/AfifaKhaled/Normalization-Techniques—survey.

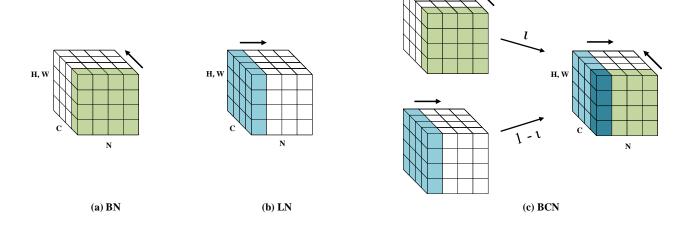


Figure 1. Visualization on several normalization techniques. Each subplot shows a feature map tensor with N the batch axes, C the channel axes, and (H, W) the spatial height and width axes.

# 3. Methodology

The motivation behind the success of the normalization techniques has been an important research topic. In this section, we investigate the motivation for developing the new normalization technique. Our research has revealed that the normalization goal for all normalization techniques is to improve the model's robustness [23].

# 3.1. Method Formulation

The idea of normalizing the (N, H, W) axes and (C, H, W) axes has been proposed before [3, 11, 20, 22]. Earlier works typically perform normalization along (N, H, W) axes or (C, H, W) axes independently. We aim to perform normalization along (N, C, H, W). However, computing the average and variance along (N, C, H, W) directly ignores the different importance between batch dimension and channel dimension. Consequently, we propose to separately normalize along the (N, H, W) and (C, H, W) axes, then combine the normalized outputs based on adaptive parameters  $\iota$ . Doing so could improve the training, validation, and test accuracy, as we show experimentally in the next section.

In a similar fashion to how BN normalizes the layer inputs, during the training, BCN first computes the average  $\mu_1$  and the variance  $\sigma_1^2$  of the layer inputs along (N, H, W) axes:

$$\mu_1 = \frac{1}{n} \sum_{i=1}^{n} x_i,\tag{9}$$

$$\sigma_1^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_1)^2.$$
 (10)

Second, BCN computes the average  $\mu_2$  and variance  $\sigma_2^2$  along (C, H, W):

$$\mu_2 = \frac{1}{n} \sum_{i=1}^n x_i,\tag{11}$$

$$\sigma_2^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_2)^2.$$
 (12)

Next,  $\bar{x}_1$  and  $\bar{x}_2$  are normalized using  $\mu_1$ ,  $\sigma_1^2$  and  $\mu_2$ ,  $\sigma_2^2$ , respectively:

$$\bar{x}_1 = \frac{(x_i - \mu_1)}{\sqrt{(\sigma_1^2 + \epsilon)}},$$
 (13)

$$\bar{x}_2 = \frac{(x_i - \mu_2)}{\sqrt{(\sigma_2^2 + \epsilon)}}.$$
(14)

BCN introduces additional learnable parameter  $\iota$  to adaptively balance the normalized outputs along the axes of (N, H, W) and (C, H, W).

$$\bar{y} = \iota \bar{x}_1 + (1 - \iota)\bar{x}_2,\tag{15}$$

Then, the output of BCN normalization can be formulated as follows:

$$Y = \gamma \bar{y} + \beta, \tag{16}$$

where  $\gamma$  and  $\beta$  are learnable parameters and  $\epsilon$  is a small constant for numerically stability.

At the inference stage, since the  $\mu$  and the  $\sigma$  are precomputed and fixed, the normalization can be fused into the convolution operation.

### Algorithm 1 Batch Channel Normalization (BCN)

#### Require:

Input  $x = \{x_1, x_2, ..., x_n\}$ , Parameters to be learned:  $\iota$ ,  $\beta$  and  $\gamma$ 

#### **Ensure:**

 $Y = BCN_{\gamma,\beta,\iota}(x_i)$ 

1: Calculate  $\mu_1$  and  $\sigma_1^2$  based on Eq. 9 and 10

2: Calculate  $\mu_2$  and  $\sigma_2^2$  based on Eq. 11 and 12

3: Calculate the normalized output  $\bar{x}_1$  along (N, H, W) and  $\bar{x}_2$  along (C, H, W) axes by Eq. 13 and 14

4: Adaptively combine  $\bar{x}_1$  and  $\bar{x}_2$  based on Eq. 15

5: Calculate the final output Y based on Eq. 16

6: return Y

Following previous works [4, 14], BCN normalizes along the (N, H, W) axes by exponential moving average [15] during the training as follows:

$$\mu = \alpha \mu + (1 - \alpha)\mu_1,\tag{17}$$

$$\sigma^2 = \alpha \sigma^2 + (1 - \alpha)\sigma_1^2,\tag{18}$$

$$\bar{x} = \frac{(x_i - \mu)}{\sqrt{(\sigma^2 + \epsilon)}},\tag{19}$$

where  $\alpha$  is set to 0.9 in our experiments.

The key difference between BCN normalization and existing normalization techniques is that under BCN, all the channels in a layer share the same normalization terms  $\mu$  and  $\sigma^2$ .

## 3.2. Implementation

BCN can be implemented by a few lines of Python code in PyTorch [17] or TensorFlow [1] where computing  $\bar{x}_1$  along (N, H, W) and  $\bar{x}_2$  along (C, H, W) is implemented. The overall BCN process is presented in Algorithm 1, with the corresponding Python code in Figure 2.

# 4. Experiments and Discussion

#### 4.1. Datasets

We evaluate the effectiveness of our technique through four representative datasets: CIFAR-10/100 [12], SVHN [16], and ImageNet [18]. The CIFAR-10/100 datasets, developed by the Canadian Institute for Advanced Research, are widely employed in various experiments. CIFAR-10 consists of 60,000 32x32 color images divided into 10 object classes, with 50,000 training images and 10,000 test images. On the other hand, CIFAR-100 comprises 100 classes with 600 images per class [12]. The Street View House Numbers (SVHN) dataset [16] contains 600,000 32x32 *RGB* images of printed digits (from 0 to 9) cropped from pictures of house number plates. ImageNet dataset [18] has

1.28M training images and 50,000 validation images with 1000 classes.

# 4.2. Experimental Setup

To investigate how BCN and the existing normalization techniques work, we conduct a series of experiments. Five normalization techniques, *i.e.*, BN [11], LN [3], IN [20], GN [22], and our BCN, are implemented from scratch in Pytorch [17].

The experimental details are the same in the five techniques (*i.e.*, loss function, batch size, *etc.*). On ImageNet, we evaluate the performance of ResNet18, VGG16, SqueezeNet and AlexNet with BN, LN and BCN, respectively. We use accuracy on different datasets to investigate the effectiveness of the BCN normalization technique.

# 4.3. Comparison with Normalization Techniques

In this subsection, we compare our method with typical normalization techniques using popular datasets and neural networks. Specifically, we compare the image classification performance of ResNet with different normalization techniques on CIFAR-10/100 and SVHN datasets. We also compare different normalization techniques on DenseNet using the ImageNet dataset and self-supervised learning on BYOL using the CIFAR-10 dataset. In addition, exploring BCN on new models like Vision Transformers using the CIFAR-10/100 and SVHN datasets.

#### 4.3.1 Results on ResNet

We perform experiments on ResNet [7] for the image classification task. The model is trained by stochastic gradient descent (SGD) starting with a learning rate of 0.1 and then reduced by a factor of 10 at the 75th and 85th epochs, respectively. A batch size of 8 and a momentum of 0.9 are used.

We show results of BCN, BN, and LN during training and validation on the three datasets in Figure 3 and Figure 4. As we can see, BCN learns most rapidly. On CIFAR-10, in about 20 epochs, it has achieved about 86.12% training accuracy and 84.16% validation accuracy, whereas in the same number of epochs, the BN and LN show 86.04% and 79.74% training accuracy and 12.87% and 78.58% validation accuracy, respectively.

In addition, Table 1 shows the results of test accuracy for BCN normalization and the representative normalization techniques (BN, LN, IN, and GN) on the CIFAR-10, CIFAR-100, and SVHN datasets. All experiments are conducted under the same learning rate, loss function, batch size, *etc*. The results show that BCN is generally applicable, gaining the best or the second best results. For example, under the CIFAR-100 dataset, BCN achieves significant improvement over the state-of-the-art techniques.

```
def BatchChannelNorm(x, gamma, beta, momentum=0.9, num_channels, eps=1e-5):
    self.num_channels = num_channels
   self.epsilon = epsilon
   self.x1 = BCN_1(self.num_channels, epsilon=self.epsilon) # normalized along (N, H, W) axes.
   {\tt self.x2 = BCN\_2(self.num\_channels,\ epsilon=self.epsilon)}\ \#\ {\tt normalized\ along\ (C,\ H,\ W)\ axes.}
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale and offset
   self.gamma = nn.Parameter(torch.ones(num_channels))
   self.beta = nn.Parameter(torch.zeros(num_channels))
    # iota is the BCN variable to be learned to adaptively balance the normalized outputs along (N, H, W) axes and (C, H, W)
    self.iota = nn.Parameter(torch.ones(self.num_channels))
   X = self.x1(x)
   Y = self.x2(x)
   Result = self.iota.view([1, self.num_channels, 1, 1]) * X + (1 - self.iota.view([1, self.num_channels, 1, 1])) * Y
   Result = self.gamma.view([1, self.num_channels, 1, 1]) * Result + self.beta.view([1, self.num_channels, 1, 1])
   return Result
```

Figure 2. Python code of Batch Channel normalization (BCN) based on PyTorch.

Method	CIFAR-10	CIFAR-100	SVHN
BN	96.11	74.50	98.22
LN	95.76	68.61	97.62
IN	96.68	73.42	98.93
GN	95.91	70.15	98.49
BCN	96.97	79.09	<u>98.63</u>

Table 1. Comparison of the test accuracy on three datasets. The best results appear in bold, and the second best are underlined.

#### 4.3.2 Results on BYOL

We apply BCN to a recent state-of-the-art method BYOL [6] for self-supervised learning. We have implemented BYOL in PyTorch, using hyperparameter settings as in the original paper [6]. BCN is applied in both online and target models. As shown in Figure 5, applying BCN can improve the performance of BYOL.

#### 4.3.3 Results on ViT

Nowadays, there is growing interest in developing Vision Transformer (ViT) methods [5] across a wider range of applications. We implemented ViT from scratch. We have tested ViT having different batch sizes and embedding dimensions on CIFAR-10/100 and SVHN datasets. Figure 6 shows that the performance of ViT improved when the normalization technique was replaced by BCN. Replacing BCN lead to nearly 0.73 % training accuracy and 0.72 % testing accuracy. In addition, Figure 7 shows the performance of BCN in the testing set. Overall, these results support our hypothesis that compared to the existing normalization techniques BN and LN, BCN can have a better performance in new models like ViT.

#### 4.4. Results on DenseNet

DenseNet is a typical Dense Convolutional Network. We have implemented DenseNet-201 in PyTorch. A careful se-

lection of learning rate value can lead to better performance results. To this end, we experiment with different learning rates to investigate what suits our datasets and topology. In this paper, we initially set the learning rate to 3e-2. Similarly, we perform experiments to identify a batch size. The batch size 512 is used. Max batch size is used to fill in GPU memory when training. In our experiments, we trained for 90 epochs, and every epoch had 2503 iterations. In addition, we record training accuracy by steps and testing accuracy by epochs. We compare the training, testing and validation performance of different normalization techniques by combining DenseNet-201 with BN, and BCN in the ImageNet dataset. As illustrated in Figure 10 and Figure 11, the proposed BCN can produce a good performance.

# 4.5. Ablation Study

In this subsection, we explore the impact of the batch size. We evaluate various batch sizes: 128, 16, and 8. Our findings are shown in Figure 9, indicating that the BCN yields favorable results with different batch sizes. To explore whether BCN mitigates the weakness of BN and LN and, as pointed by [11], that BN has a bad performance in the case of small batch size. In this subsection, we focus on addressing the minibatch problem of BN. The experiments show that BCN alleviates the problem of BN with small batch size. BCN achieves good performance with small batch sizes 4 and 2, as shown in Figure 8. In a batch

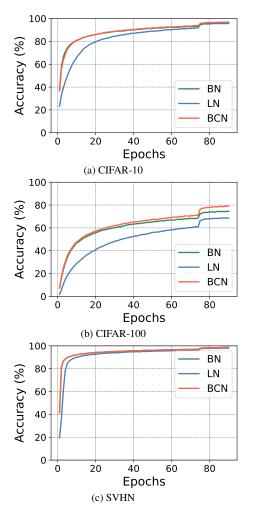


Figure 3. Training accuracy of ResNet with different normalization techniques on (a) CIFAR-10, (b) CIFAR-100, (c) SVHN.

size of 8 and 20 epochs, BCN has achieved about 86.12% training accuracy and 84.16% validation accuracy, whereas in the same number of epochs and 2 batch size, BCN has achieved about 84.58% training accuracy and 81.42% validation accuracy. This proves that the proposed technique works well in a small batch size.

## 5. Conclusion

In this paper, we proposed a new normalization technique termed Batch Channel normalization (BCN). It simultaneously exploits the channel and batch dimensions and adaptively combines the normalized outputs. Our experiments on typical models and datasets show that BCN can consistently outperform the state-of-the-art normalization techniques, demonstrating that BCN is a general normalization technique. An ablation study of directly computing the average and variance along (N, C, H, W) can be done as a future work. Moreover, we will investigate the BCN

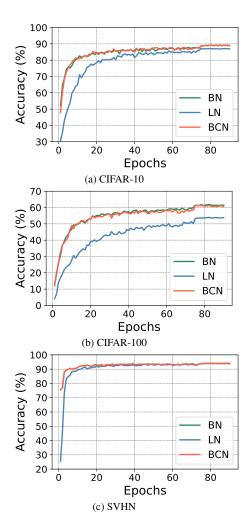


Figure 4. Validation accuracy of ResNet with different normalization techniques on (a) CIFAR-10, (b) CIFAR-100, (c) SVHN

technique across more applications and evaluate the usefulness of BCN across a wider range of CNN architectures.

### References

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, 2016. 4
- [2] Devansh Arpit, Yingbo Zhou, Bhargava Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1168–1176, New York, New York, USA, 2016. 2
- [3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton.

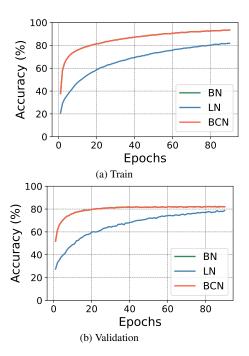


Figure 5. Training and validation accuracy curve of different normalization techniques for BYOL on the CIFAR-10 dataset. Note that the results of BN are exactly the same as that of BCN.

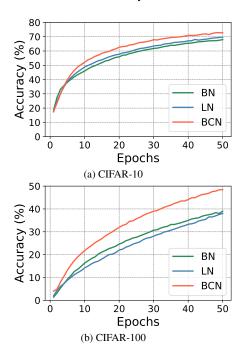


Figure 6. Training accuracy of ViT with different normalization techniques on (a) CIFAR-10, (b) CIFAR-100.

Layer normalization. *Advances in NIPS Deep Learning Symposium*, 2016. 1, 2, 3, 4

[4] Zhaowei Cai, Avinash Ravichandran, Subhransu Maji, Charless C. Fowlkes, Zhuowen Tu, and Stefano Soatto. Exponen-

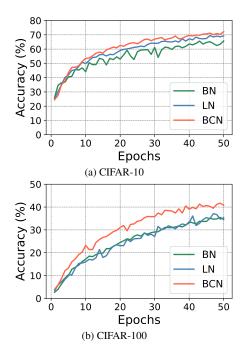


Figure 7. Testing accuracy of ViT with different normalization techniques on (a) CIFAR-10, (b) CIFAR-100.

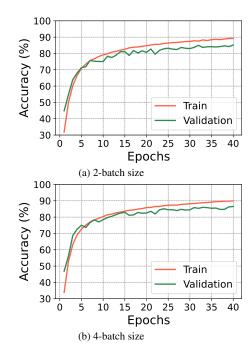


Figure 8. The accuracy curve for mini-batch size on the CIFAR-10 dataset.

tial moving average normalization for self-supervised and semi-supervised learning. *CoRR*, abs/2101.08482, 2021. 4

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Syl-

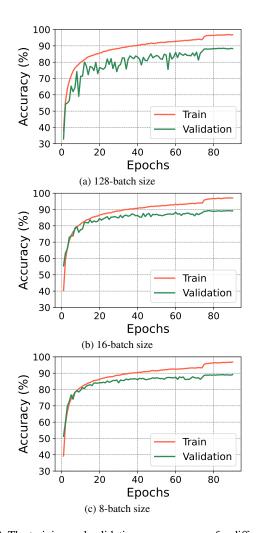


Figure 9. The training and validation accuracy curve for different batch sizes on the CIFAR-10 dataset.

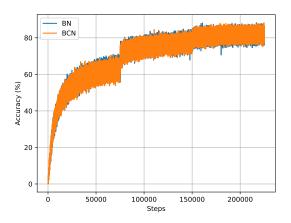


Figure 10. Training accuracy of DenseNet with ImageNet dataset.

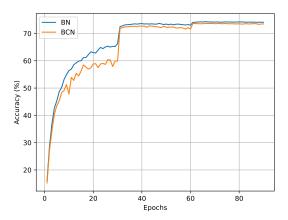


Figure 11. Testing accuracy of DenseNet with ImageNet dataset.

- vain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. 2, 5
- [6] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *CoRR*, abs/2006.07733, 2020. 2, 5
- [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778, 2016. 1, 4
- [8] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 1731–1741, 2017.
- [9] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [10] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances* in *Neural Information Processing Systems*, 2017. 2
- [11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learn*ing, pages 448–456. pmlr, 2015. 1, 2, 3, 4, 5
- [12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 4
- [13] Qianli Liao, Kenji Kawaguchi, and Tomaso A. Poggio. Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning. *CoRR*, abs/1610.06160, 2016. 2
- [14] Chunjie Luo, Jianfeng Zhan, Lei Wang, and Wanling Gao.

- Extended batch normalization. *CoRR*, abs/2003.05569, 2020. 4
- [15] Jishnu Mukhoti, Puneet K. Dokania, Philip H. S. Torr, and Yarin Gal. On batch normalisation for approximate bayesian inference. *CoRR*, abs/2012.13220, 2020. 4
- [16] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. The street view house numbers SVHN dataset. 2011. 4
- [17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 4
- [18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. 4
- [19] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, page 901, 2016.
- [20] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 2, 3, 4
- [21] Xiaoxia Wu, Edgar Dobriban, Tongzheng Ren, Shanshan Wu, Zhiyuan Li, Suriya Gunasekar, Rachel Ward, and Qiang Liu. Implicit regularization of normalization methods. *CoRR*, abs/1911.07956, 2019. 2
- [22] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision* (ECCV), pages 3–19, 2018. 1, 2, 3, 4
- [23] Amir Ziaee and Erion Çano. Batch layer normalization: A new normalization layer for CNNs and RNNs. In *Proceedings of the 6th International Conference on Advances in Artificial Intelligence*, pages 40–49, 2022. 3