# BCN: BATCH CHANNEL NORMALIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Normalization techniques enable higher learning rates and are less careful in initialization. Unlike the standard Batch Normalization $(BN)$ and Layer Normalization $(LN)$, where $BN$ computes the mean and variance along the $(N, H, W)$ axes ($N$ is the batch axes, $H$ and $W$ are the spatial height and width axes) and $LN$ computes the mean and variance along the $(C, H, W)$ axes ($C$ is the channel axes), this paper presents a simple normalization technique called Batch Channel Normalization $(BCN)$. $BCN$ separately normalizes inputs along the $(N, H, W)$ and $(C, H, W)$ axes, then combine the normalized outputs based on adaptive parameters. $BCN$ exploits both the channel and batch dependence and adaptively combines the advantages of $BN$ and $LN$. 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 $(BCN)$ technique improves the generalization performance of various models.

## 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. Deep Neural Networks ($DNNs$) find extensive applications 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 (Ioffe & Szegedy, 2015). 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. However, these methods have both benefits and drawbacks beyond reducing internal covariate shifts. The disadvantages include increased training time and limited impact on hyper-parameter optimization. Conversely, the advantages encompass the ability to use higher learning rates without encountering vanishing or exploding gradients, regularization that improves generalization properties, accelerated training, and the creation of more reliable models.

Specifically, Batch Normalization ($BN$) (Ioffe & Szegedy, 2015), Layer Normalization ($LN$) (Ba et al., 2016), Group Normalization ($GN$) (Wu & He, 2018) and Instance Normalization ($IN$) (Ulyanov et al., 2016) have achieved remarkable success on deep learning models. Among them, $BN$ is widely used for deep neural networks. Despite their great success in many applications, there are still some problems. For example, $BN$ requires large batch sizes (Wu & He, 2018), 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, $GN$ and $LN$ are proposed to avoid exploiting batch dimension. Unlike $BN$, $GN$ and $LN$ do not impose any restriction on the size of each mini-batch (Ba et al., 2016). For $LN$, it does not work as well as $BN$ with convolutional layers. This motivates us to develop a new normalization technique to overcome the limitations of $BN$ and $LN$ as well as to fully embody the advantages of the two techniques.

In this study, we begin by conducting a comprehensive analysis on commonly employed normalization methods - batch normalization and layer normalization. Our objective is to have a thorough understanding of the strengths and weaknesses associated with each method. Subsequently, we pro-

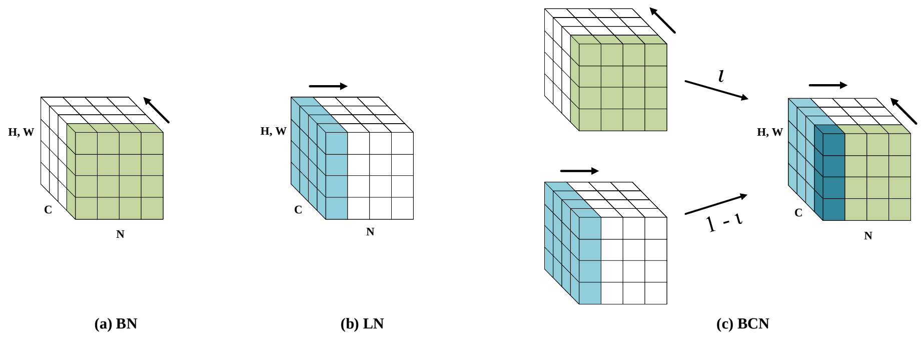


Figure 1: Visualization on several normalization techniques. Each subplot shows a feature map tensor with $N$ the batch axes, $C$ the channel axes, and $(H, W)$ the spatial height and width axes.

pose a novel approach called Batch Channel Normalization ($BCN$), which combines the benefits of both methods while mitigating their respective deficiency.

In contrast to previous techniques, this paper aims 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, we propose a simple technique termed Batch Channel Normalization. First, $BCN$ 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. Then the normalized outputs are combined based on adaptive parameters.

To check the effectiveness of the proposed method, we apply $BCN$ to image classification using Residual network (ResNet) (He et al., 2016), Densely Connected Convolutional Networks ($DenseNet$) (Huang et al., 2018), and Bootstrap Your Own Latent ($BYOL$) (Grill et al., 2020). Our findings 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 (Figure 1).
- $BCN$ exploits the channels dependence and the batch sizes 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

Normalization methods for deep neural networks can be categorized into three groups (Gitman & Ginsburg, 2017).

The first group involves normalizing different dimensions of the output. Examples include Layer Normalization (Ba et al., 2016), which normalizes inputs across features; Instance Normalization (Ulyanov et al., 2016), which normalizes over spatial locations in the output; and Group Normalization (Wu & He, 2018), which independently normalizes along spatial dimensions and feature groups.

The second group modifies the original batch normalization method (Ioffe & Szegedy, 2015). This group includes methods like Ghost $BN$ (Hoffer et al., 2017), which normalizes independently across different splits of batches, and Batch Re-normalization (Ioffe, 2017) or Streaming Normalization

(Liao et al., 2016), both of which make changes to utilize global averaged statistics instead of batch statistics.

The third group consists of methods that normalize weights rather than activations. This group comprises Weight Normalization (Salimans & Kingma, 2016) and Normalization Propagation (Arpit et al., 2016), both of which divide weights by their $\ell_2$ norm, differing only in minor details. Despite the success of the three groups of normalization techniques, there is a trend in proposing new normalization techniques to combine to gain further improvement.

Our method belongs to the first group. The proposed method aims to normalize along to $(C, N, H, W)$ axes. To do this, $BCN$ first computes the $\mu_1$ and $\sigma_1^2$ of the layer inputs along the $(N, H, W)$ axes. Next, it computes the $\mu_2$ and $\sigma_2^2$ along the $(C, H, W)$ axes. $BCN$ combines the benefits of both methods while mitigating their respective deficiency.

We apply $BCN$ normalization to image classification using Residual network (ResNet)(He et al., 2016), Densely Connected Convolutional Networks ($DenseNet$) (Huang et al., 2018), Bootstrap Your Own Latent ($BYOL$) (Grill et al., 2020) , and show that $BCN$ can outperform the existing techniques and exceed their accuracies.

## 3 Preliminaries

In this section, we provide details of the methods that are most related to our work, the Batch Normalization ($BN$) and Layer Normalization ($LN$).

### 3.1 Batch Normalization

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:

$$\mu_B = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma_B^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2, \qquad \bar{x}_B = \gamma\frac{(x_i - \mu_B)}{\sqrt{(\sigma_B^2 + \epsilon)}} + \beta. \tag{1}$$

We can see that $BN$ computes the $\mu_B$ and $\sigma_B^2$ along the $(N, H, W)$ axes (Ioffe & Szegedy, 2015). During the testing, $BN$ computes the $\mu_B$ and $\sigma_B^2$ by exponential moving average during the training set:

$$\mu = \alpha\mu + (1 - \alpha)\mu_B, \qquad \sigma^2 = \alpha\sigma^2 + (1 - \alpha)\sigma_B^2, \qquad \bar{x} = \gamma\frac{(x_i - \mu)}{\sqrt{(\sigma^2 + \epsilon)}} + \beta, \tag{2}$$

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.

### 3.2 Layer Normalization

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

$$\mu_L = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma_L^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2, \qquad \bar{x}_L = \gamma\frac{(x_i - \mu_L)}{\sqrt{(\sigma_L^2 + \epsilon)}} + \beta. \tag{3}$$

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

## 4 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 goals for all normalization techniques are to improve the model's robustness. $BN$ has several deficiencies, including the inconsistency between the training set and test set, can not be used for online learning tasks, and can not be used for large distributed models because the size of mini-batches has to be small. On the other hand, $LN$ does not work as well as $BN$ with convolutional layers.

In order to address the above limitations of $BN$ and $LN$, we propose to fully embody the advantages of both $BN$ and $LN$. In a similar fashion to the way that $BN$ and $LN$ normalize the input layer, $BCN$ normalizes along $(N, C, H, W)$, which improves the training stability. On the other hand, there is no need to compute the $\mu$ and the $\sigma$ at the inference time comparing to existing normalization techniques. More importantly, since the $\mu$ and the $\sigma$ are pre-computed and fixed at the inference time, the normalization can be fused into the convolution operation. This is very useful to speed up the inference time. Generally, $BCN$ has faster convergence than $BN$ and $LN$ in different deep learning models, such as Residual Network ($ResNet$), Very Deep Convolutional Networks ($VGGNet$), and Bootstrap Your Own Latent ($BYOL$). Alternatively, our research reveals that, fundamentally, $BCN$ (i.e., Batch Channel Normalization) may be more appropriate to be integrated into existing models for various applications in the field of computer vision. This section presents the methodology of our paper, including the method formulation and pseudo code of implementation.

## 4.1 Method Formulation

The idea of normalizing the $(N, H, W)$ axes and $(C, H, W)$ axes has been proposed before (Ioffe & Szegedy, 2015; Ba et al., 2016; Ulyanov et al., 2016; Wu & He, 2018). However, 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 as follows:

$$\mu_1 = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma_1^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_1)^2. \tag{4}$$

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

$$\mu_2 = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma_2^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_2)^2. \tag{5}$$

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

$$\bar{x}_1 = \frac{(x_i - \mu_1)}{\sqrt{(\sigma_1^2 + \epsilon)}}, \qquad \bar{x}_2 = \frac{(x_i - \mu_2)}{\sqrt{(\sigma_2^2 + \epsilon)}}. \tag{6}$$

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

$$\bar{y} = \iota\bar{x}_1 + (1 - \iota)\bar{x}_2, \tag{7}$$

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

$$Y = \gamma\bar{y} + \beta, \tag{8}$$

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

At the inference stage, following previous works (Cai et al., 2021; Luo et al., 2020), $BCN$ normalizes along the $(N, H, W)$ axes by exponential moving average (Mukhoti et al., 2020) during the training as follows:

$$\mu = \alpha\mu + (1 - \alpha)\mu_1, \qquad \sigma^2 = \alpha\sigma^2 + (1 - \alpha)\sigma_1^2, \qquad \bar{x} = \frac{(x_i - \mu)}{\sqrt{(\sigma^2 + \epsilon)}}, \tag{9}$$

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 $\sigma^2$.

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. 4
2: Calculate $\mu_2$ and $\sigma_2^2$ based on Eq. 5
3: Calculate the normalized output $\bar{x}_1$ along $(N, H, W)$ and $\bar{x}_2$ along $(C, H, W)$ axes by Eq. 6
4: Adaptively combine $\bar{x}_1$ and $\bar{x}_2$ based on Eq. 7
5: Calculate the final output $Y$ based on Eq. 8
6: return $Y$
```

```python
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.
    self.x2 = BCN_2(self.num_channels, epsilon=self.epsilon) # 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 the BCN variable to be learnt to adaptively balance the normalized outputs along (N, H, W
         ) axes and (C, H, W) axes.
    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.

### 4.2 IMPLEMENTATION

$BCN$ can be implemented by a few lines of python code in TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2017) 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 and Figure 2.

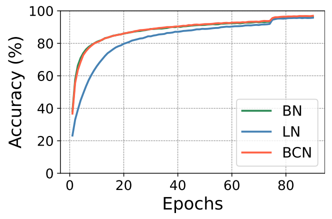
## 5 EXPERIMENTS AND DISCUSSION

### 5.1 DATASETS

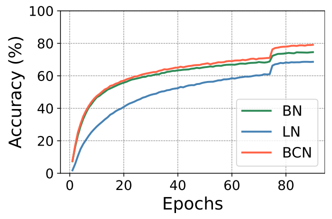
We evaluate the effectiveness of our technique through four representative datasets: CIFAR-10/100 (Krizhevsky et al., 2009), SVHN (Netzer et al., 2011), and ImageNet (Russakovsky et al., 2015). The CIFAR-10/100 datasets, developed by the Canadian Institute for Advanced Research, are widely employed in 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 (Krizhevsky et al., 2009). The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) contains 600,000 32×32 $RGB$ images of printed digits (from 0 to 9) cropped from pictures of house number plates. ImageNet dataset (Russakovsky et al., 2015) has 1.28M training images and 50,000 validation images with 1000 classes.

### 5.2 EXPERIMENTAL SETUP

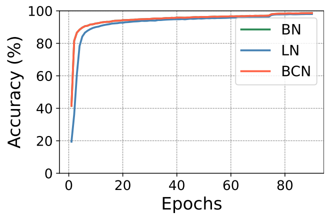
To investigate how $BCN$ and the existing normalization techniques work, we conduct a series of experiments. Five normalization techniques (i.e., $BN$ (Ioffe & Szegedy, 2015), $LN$ (Ba et al., 2016), $IN$ (Ulyanov et al., 2016), $GN$ (Wu & He, 2018), and our $BCN$ ) implemented from scratch in



(a) CIFAR-10

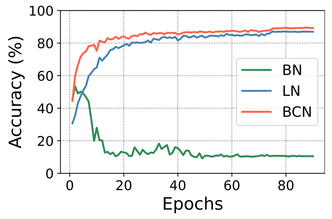


(b) CIFAR-100

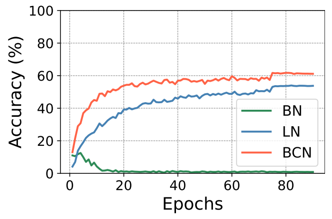


(c) SVHN

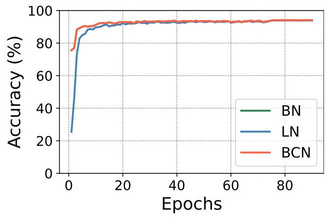
Figure 3: Training accuracy of different normalization techniques on (a) CIFAR-10, (b) CIFAR-100, (c) SVHN.



(a) CIFAR-10



(b) CIFAR-100



(c) SVHN

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

Pytorch (Paszke et al., 2017). The experimental details are the same in the five techniques (i.e., loss function, batch size, etc). On ImageNet, we evaluate ResNet18, VGG16, SqueezeNet and Alexnet with $BN$, $LN$ and $BCN$, respectively. For the batch size of 8, the initial learning rate is set to 0.1. We use accuracy on different datasets to investigate the effectiveness of the $BCN$ normalization technique.

## 5.3 Comparison with Normalization Techniques

In this subsection, we do comparison of our method with typical normalization techniques using typical datasets and neural networks. Specifically, we consider image classification task on $ResNet$ using CIFAR-10/100 and SVHN datasets and $DenseNet$ using ImageNet dataset, and self-supervised learning on $BYOL$ using CIFAR-10 dataset.

### 5.3.1 On Residual Network ($ResNet$)

We perform experiments on ResNet (He et al., 2016) 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 35th and 45th 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 $BCN$ 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. Note that $BCN$ achieves significant improvement over the state-of-the-art techniques on CIFAR-100.

| **Method** | **CIFAR-10** | **CIFAR-100** | **SVHN** |
|---|---|---|---|
| $BN$ | 96.11 | <u>74.50</u> | 98.22 |
| $LN$ | 95.76 | 68.61 | 97.62 |
| $IN$ | <u>96.68</u> | 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.

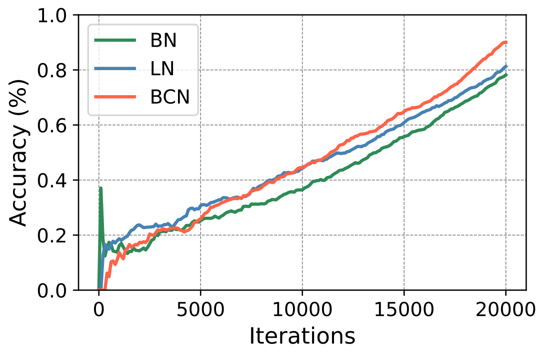


Figure 5: The accuracy curve for ImageNet dataset.

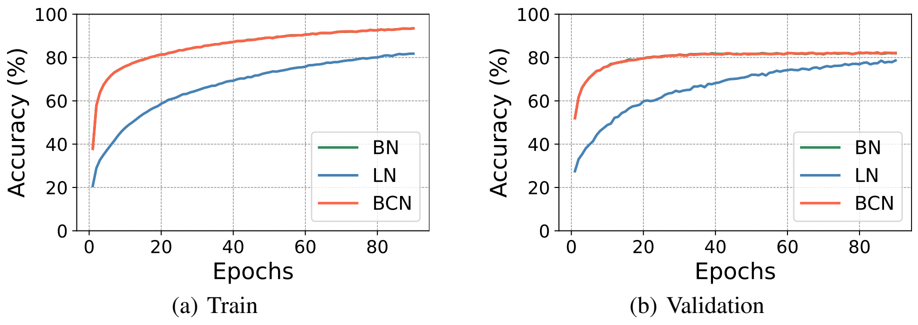


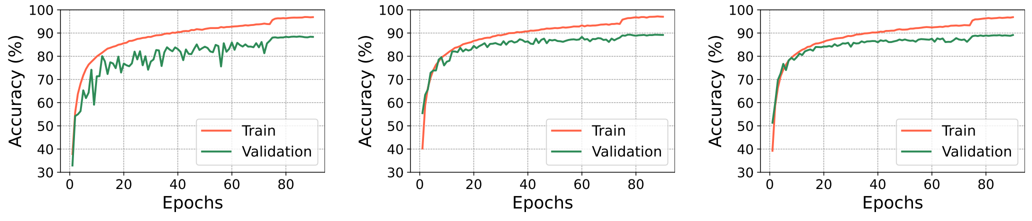
Figure 6: Training and validation accuracy curve of different normalization techniques for BYOL on the CIFAR-10 dataset.

### 5.3.2 On Densely Connected Convolutional Networks ($DenseNet$)

DenseNet is a typical Dense Convolutional Network (Huang et al., 2018). Our baseline model is $DenseNet - 201$ trained with $BN$. To compare with the state-of-the-art techniques, we replace $BN$ with the $BCN$ and $LN$. Figure 5 shows that the state-of-the-art techniques with the proposed $BCN$ can produce better performance than $BN$ and $LN$.

### 5.3.3 On Bootstrap Your Own Latent ($BYOL$)

We have applied $BCN$ to recent state-of-the-art method for self-supervised learning (BYOL (Grill et al., 2020)). We implemented BYOL in PyTorch, using hyperparameter settings as in the original paper (Grill et al., 2020). Applying $BCN$ to this method is simple, requiring a few lines of code change. $BCN$ is applied in both online and target models. The experimental results show that applying $BCN$ in both online and target models have improved the performances, as shown in Figure 6. Our results show that the proposed technique is very useful.



(a) 128 (b) 16 (c) 8

Figure 7: The accuracy curve for different batch size on the CIFAR-10 dataset.

### 5.4 Ablation Study

In this subsection, we experiment with the impact of batch size. We evaluate batch sizes of 128, 16 and 8. Our findings are shown in Figure 7, indicating that the $BCN$ yields favorable results with different batch sizes.

## 6 Limitations

This paper aims to explore the generalization of $BCN$ among the existing normalization techniques. However, there is a need for further investigation into the dynamic learning of cross-channels in the $BCN$. Furthermore, the specific connection between $BCN$, $LN$, $GN$, $BN$ and $IN$ as well as their transformation with each other, requires further investigation (e.g., new models like Vision Transformers).

## 7 Conclusion

In this paper, we proposed a new normalization technique termed Batch Channel normalization ($BCN$). It exploits the channel dependence and batch sizes simultaneously, then 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. Based on our technique, we have a number of possible directions for future work. We will investigate the $BCN$ technique across a wider variety of applications and evaluate the usefulness of $BCN$ across a wider range of $CNN$ architectures (e.g., new models like Vision Transformers).

## References

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)*, pp. 265–283, 2016.

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*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1168–1176, New York, New York, USA, 20–22 Jun 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *Advances in NIPS Deep Learning Symposium*, 2016.

Zhaowei Cai, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Zhuowen Tu, and Stefano Soatto. Exponential moving average normalization for self-supervised and semi-supervised learning, 2021.

Igor Gitman and Boris Ginsburg. Comparison of batch normalization and weight normalization algorithms for the large-scale image classification, 2017.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila 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, 2020.

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)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

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*, pp. 1731–1741, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456. pmlr, 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Qianli Liao, Kenji Kawaguchi, and Tomaso Poggio. Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning, 2016.

Chunjie Luo, Jianfeng Zhan, Lei Wang, and Wanling Gao. Extended batch normalization, 2020.

Jishnu Mukhoti, Puneet K. Dokania, Philip H. S. Torr, and Yarin Gal. On batch normalisation for approximate bayesian inference, 2020.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. The street view house numbers (svhn) dataset. 2011. URL http://ufldl.stanford.edu/housenumbers/.

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.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 901, 2016.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.