RESEARCH-ARTICLE

Batch Normalization: Is Learning An Adaptive Gain and Bias Necessary?

YAN WANG, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

XIAOFU WU, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

YUANYUAN CHANG, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

SUOFEI ZHANG, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

QUAN ZHOU, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

JUN YAN, Nanjing University of Post and TeleCommunications, Nanjing, Jiangsu, China

Published: 26 February 2018

ICMLC 2018: 2018 10th International Conference on Machine Learning and Computing
February 26 - 28, 2018
Macau, China

ICMLC '18: Proceedings of the 2018 10th International Conference on Machine Learning and Computing (February 2018)
https://doi.org/10.1145/3195106.3195150
ISBN: 9781450363532

Batch Normalization: Is Learning An Adaptive Gain and Bias Necessary?

Yan Wang, Xiaofu Wu, Yuanyuan Chang, Suofei Zhang, Quan Zhou and Jun Yan
Nanjing University of Posts and Telecommunications
Gulou District, Nanjing 210003, CHINA
{1016010603, xfuwu, 1016010601, zhangsuofei, quan.zhou, yanj}@njupt.edu.cn

ABSTRACT
The state-of-the-art training of deep neural networks requires to normalize the activities of the neurons for accelerating the training process. A standard approach is to employ batch normalization (BN), in which the activations are normalized by the mean and standard deviation of the training mini-batch. To be invertible, BN also introduces an adaptive gain and bias which are applied after the normalization but often before the non-linearity. In this paper, we investigate the effects of learnable parameters, gain and bias, on the training of various typical deep neural nets, including ALL-CNNs, Network In Network (NIN), ResNets. Through extensive experiments, we show that there is no big difference in both training convergence and final test accuracy if we remove the BN layer following the final convolutional layer from a convolutional neural network (CNN) for standard classification tasks. We also observed that without adaptively updating learnable parameters for BN layers, it often requires less time for training of very deep neural nets such as ResNet-101.

CCS Concepts
• **Theory of computation → Machine learning theory**

Keywords
batch normalization; CNN; gain and bias

1. INTRODUCTION
In the last few years, deep neural networks [11][15] have demonstrated impressive breakthrough in natural language processing, image classification and face recognition. Compared to their sh- allow counterparts, deep neural networks are more difficult to train due to gradient vanishing/exploding problems.

For deep neural networks, it was observed that the inputs at each layer may vary drastically, both in their value range and statistic distribution. The drastic change in both the value range and statistic distribution is proved to be essential for gradient vanishing/exploding. To remedy this problem, various normalization methods were proposed, including batch normalization (BN) [13], layer normalization (LN) [6], weight normalization (WN) [14].

ICMLC 2018, February 26–28, 2018, Macau, China

ACM ISBN 978-1-4503-6353-2/18/02…$15.00
DOI: https://doi.org/10.1145/3195106.3195150

For BN, the activations at each layer are normalized by the mean and standard deviation of the training mini-batch. To be invertible, it introduces an adaptive gain and bias for compensating the normalization process. This technique has rapidly became a standard for constructing deep convolutional neural networks. BN allows a large learning rate and partially reduces the requirement of careful initialization or even eliminates the need for dropout [10].

A main shortage of BN lies in that the normalization process introduces some dependencies between the examples in a mini-batch, which limits its usage in recurrent models such as LSTMs [12] and generating models [5]. Instead of using examples in a mini-batch to compute a feature's mean and variance, LN utilizes the combined activities of all units within a layer as the normalizer. Weight normalization is a reparameterization of the weight vectors in a neural network that decouples the length of those weight vectors from their direction. By eliminating the dependencies between examples in a mini-batch, both of these methods can be successfully applied to LSTMs and generating models.

In this paper, we investigate the effects of learnable parameters of BN, namely, adaptive gain and bias, on the performance of deep convolutional neural networks. Through extensive experiments, we show that it is not necessary to adaptively adjust the learnable parameters of gain and bias. In most convolutional layers followed by a BN layer, the gain and bias can be set to be constant values. The only exception often lies in the last convolutional layer if it is followed by a BN layer, which requires to adjust its gain and bias noticeably. However, we found that this BN layer following the last convolutional layer can be totally removed.

2. EFFECTS OF LEARNABLE PARAMETERS OF BN ON CNNS

2.1 BN and Its Learnable Parameters
For a layer with an m-dimensional input $x = (x^{(1)}, \Lambda\ x^{(m)})$, we will normalize each dimension as

$$x'(k) = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

For a 4-dimensional tensor input of size (N,K,H,W) = (batch, channels, height, weight), the batch normalization is often performed over each channel (feature map). Therefore, with a convolutional layer, the batch normalization is implemented for each channel with the effective batch size of $N \times H \times W$.

The BN Transform ($BN_{\gamma,\beta}$), applied to activation x over a mini-batch, is explicitly shown in Algorithm 1 [13]. In Figure 1, we show an adaptive gain and bias for a typical neuron with BN

along the training process for a ALL-CNN [7]. The detailed configuration of this ALL-CNN is given in Section 3.1. As shown, the learnable gain and bias is of limited value range. This has also been observed for most neurons of BN. Therefore, we presume that learning an adaptive gain and bias may not be necessary for the use of BN in CNNs.

Input:

Values of x over a mini-batch: $B = x_{1\Lambda m}$

Parameters to be learned: γ, β

Output: $y_i = \mathrm{BN}_{\gamma,\beta}(x_i)$

$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ //mini-batch mean

$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ //mini-batchvariance

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$ //normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$ //scalar and shift

Algorithm 1. Batch Normalizing Transform.

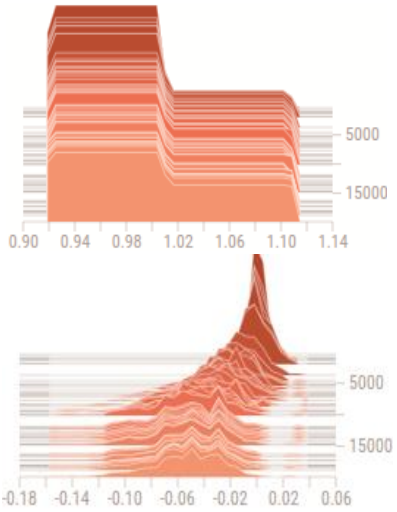

Figure 1. The histograms[1] of an adaptive gain (top) and bias (bottom) on conv6, respectively, for a typical BN layer employed in the ALL-CNN along the training process.

2.2 BN without Adjusting Learnable Gain and Bias

We consider a simplified version of the BN, where the learnable gain and bias are simply set to constant ones, namely,

$$\mathrm{BN}_{\gamma,\beta} \rightarrow \mathrm{BN}_{1,0}$$

Therefore, it simply neglects the learning of parameters γ and β in the standard BN layer. In what follows, we call this simplified version ($\mathrm{BN}_{1,0}$) as CBN for convenience.

[1] Histogram is obtained using TensorBoard from TensorFlow, where the x-, y-, and z-axis denotes the discrete value, the training step, and the number of a given value, respectively.

With the standard BN, backpropagation through a layer is unaffected by the scale of its parameters. This still holds for CBN. Indeed, for a scalar a,

$$BN_{1,0}(Wu) = BN_{1,0}((aW)u)$$

Therefore,

$$\frac{\partial BN_{1,0}(Wu)}{\partial u} = \frac{\partial BN_{1,0}((aW)u)}{\partial u},$$

$$\frac{\partial BN_{1,0}((aW)u)}{\partial (aW)} = \frac{1}{a}\cdot\frac{\partial BN_{1,0}(Wu)}{\partial W}.$$

It means that

- The scale does not affect the layer Jacobian nor the gradient propagation, as did in the standard BN.
- The larger weights lead to smaller gradients, and the CBN still stabilizes the parameter growth.

2.3 The Placement of CBN in Deep Convolutional Neural Nets

Deep CNNs are with the layered structure, often composed of a series of convolutional layers, pooling layers and fully-connected layers. In typical CNNs, the fully-connected layer is often placed between the last convolutional layer and the softmax layer. If this last covolutional layer is followed by a standard BN layer, the learnable gain and bias may experience large fluctuations during training.

This is explicitly shown in Figure 2 again for the ALL-CNN [7].

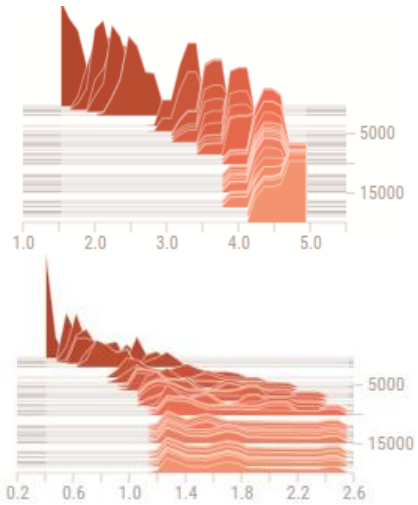

Figure 2. The histograms of an adaptive gain (top) and bias (bottom), respectively, for the BN layer attached to the last convolutional layer along the training process.

Therefore, we are interested in three typical layered structures for the placement of CBNs.

- CBN-ALL: A layered convolutional neural network, where CBNs ($\mathrm{BN}_{1,0}$) are employed for all convolutional layers;
- CBN-LConv-NoBN: A layered convolutional neural network, where CBNs ($\mathrm{BN}_{1,0}$) are employed for all convolutional layers except the last convolution layer;
- CBN-LConv-BN: A layered convolutional neural network, where a standard BN layer ($\mathrm{BN}_{\gamma,\beta}$) is employed for the last

convolution layer and the other convolutional layers are followed by CBNs ($BN_{1,0}$).

3. EXPERIMENTAL VALIDATION

3.1 Deep Neural Networks

Table 1. Architecture of the ALL-CNNs network.

Layer name	Layer description
input	Input 24 ×24 ×3 RGB image
conv1	3 ×3 conv. BN. 96 ReLU, stride 1
conv2	3 ×3 conv. BN. 96 ReLU, stride 1
conv3	3 ×3 conv. BN. 96 ReLU, stride 2
conv4	3 ×3 conv. BN. 192 ReLU, stride 1
conv5	3 ×3 conv. BN. 192 ReLU, stride 1
conv6	3 ×3 conv. BN. 192 ReLU, stride 2
conv7	3 ×3 conv. BN. 192 ReLU, stride 1
conv8	3 ×3 conv. BN. 192 ReLU, stride 1
conv9	3 ×3 conv. BN. 10 ReLU, stride 1
global_pool	global average pooling(6 ×6)
softmax	10/100-way softmax

For experimental validation, we focus on two deep neural networks (DNNs), namely, ALL-CNNs [7], Network In Network (NIN) [9], ResNets [8]. The detailed network structure of the employed ALL-CNNs is given in Table 1, while the NiN and ResNets' structure is just the same as that reported in [9][8] respectively.

3.2 CIFAR-10/100

To evaluate the performance of DNNs with the employment of CBNs for the application of supervised classification, we consider the CIFAR-10/100 data sets of natural images [1]. Both CIFAR-10 and CIFAR-100 contain 50,000 32 × 32 RGB images for training, 10,000 images for validation, and each image is labeled as 10 total number of classes for CIFAR-10 and 100 for CIFAR-100.

The training of ALL-CNNs and NIN on CIFAR-10 adopts a momentum optimization algorithm and data augmentation, L2-weight decay [2] to prevent over-fitting. The initial learning rate is 0.1. The exponential decay is used to reduce the learning rate to 0.01 after 20k steps. The network architecture is shared for CIFAR-10 and CIFAR-100 and the only minor change is the fully-connected layer, which is adapted from CIFAR-10 for matching 100 categories in CIFAR-100.

To assess the performance of CBN in deep convolutional neural networks, we follow the He's approach [8] and train deep ResNets with 29 and 101 layers. For training, the momentum optimizer is employed with momentum setting to 0.9. Both data augmentation and L2 regularization for the weight are employed. A learning rate schedule of (0-31k[0.1], 31k-58k[0.01], 58k-78k[0.001]) is used. Each ResNet is trained with 200 epochs for about 78k steps.

We evaluated 4 typical deep neural network structures with different placements of CBN or BN. 1) The standard use of BN layer following all convolutional layers (BN); 2) CBN-ALL; 3) CBN-LConv-BN; 4) CBN-LConv-NoBN. The differences are detailed in Section 2.3. The classification accuracy is summarized in Table 2.

Table 2. Summary of Test Accuracy Performance.

Network	CIFAR-10(39k)	CIFAR-100(50k)
ALL-CNN-BN	**0.912**	0.669
ALL-CNN-CBN-ALL	0.902	0.649
ALL-CNN-CBN-LConv-BN	0.907	0.667
ALL-CNN-CBN-LConv-NoBN	**0.912**	**0.674**
NIN-BN	**0.899**	0.657
NIN-CBN-ALL	0.871	0.607
NIN-CBN-LConv-BN	0.895	0.660
NIN-CBN-LConv-NoBN	0.894	**0.663**
ResNet-29	0.9145(78k)	0.6948(78k)
ResNet-29-BN	0.9447	0.7785
ResNet-29-CBN-ALL	0.9472	0.7785
ResNet-29-CBN-LConv-BN	**0.9487**	**0.7848**
ResNet-29-CBN-LConv-NoBN	0.9461	0.7663
ResNet-101-BN	0.9506	0.7923
ResNet-101-CBN-ALL	0.9465	0.7930
ResNet-101-CBN-LConv-BN	**0.9507**	**0.7976**

4. ANALYSIS OF EMPIRICAL RESULTS

4.1 Training Convergence and Test Accuracy

First of all, the application of BN with learnable gain and bias to ALL-CNN is investigated. The employed ALL-CNNs has a total of 9 convolutional layers, each of which followed by a standard BN. By setting γ=1, β=0 for the purpose of initialization, the distribution of a given BN layer outputs can be obtained by running the training procedure, which are shown in Figure 3. It can be found that, except for the last convolutional layer, the output of each BN layer is nearly normally-distributed when the training proceeds.

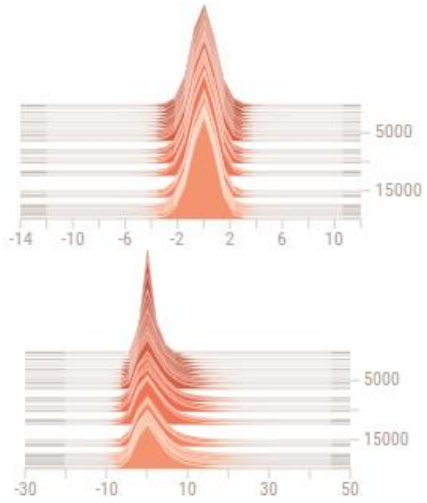

Figure 3. Training of an ALL-CNN over CIFAR-10. Top: the normalized distribution of the layer output. Due to similar distributions of layers 1 to 8, we only plot the distribution of the output for the conv-4 layer. Bottom: the output distribution after BN of the last convolutional layer.

We have present the summary of the test accuracy performance for four typical network architectures. To obtain detailed informa-

tion about the training converge, we plot the training loss, along with the test accuracy, versus training steps as shown in Figure 4, Figure 5 and Figure 6. As shown, there is no clear advantage for the use of BN compared to the use of CBN. We also observe a slight test accuracy advantage of CBN-LConv-BN structure over the standard BN structure for running ResNets over both CIFAR-10 and CIFAR-100 datasets.

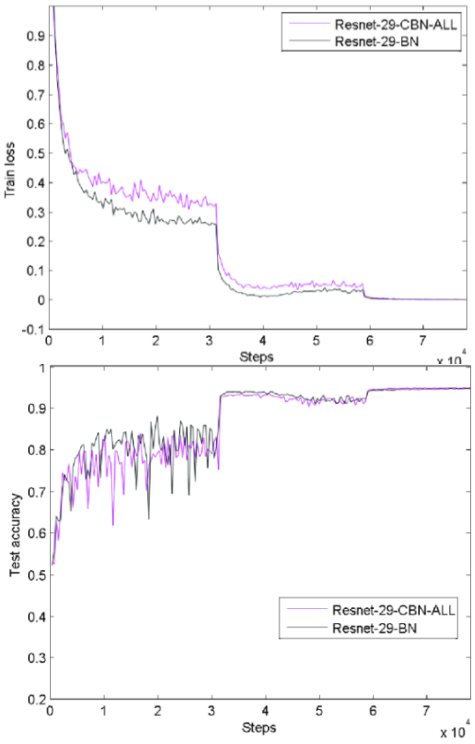

Figure 4. The training loss (top) and test accuracy (bottom) along the training process for ResNets over CIFAR-10.

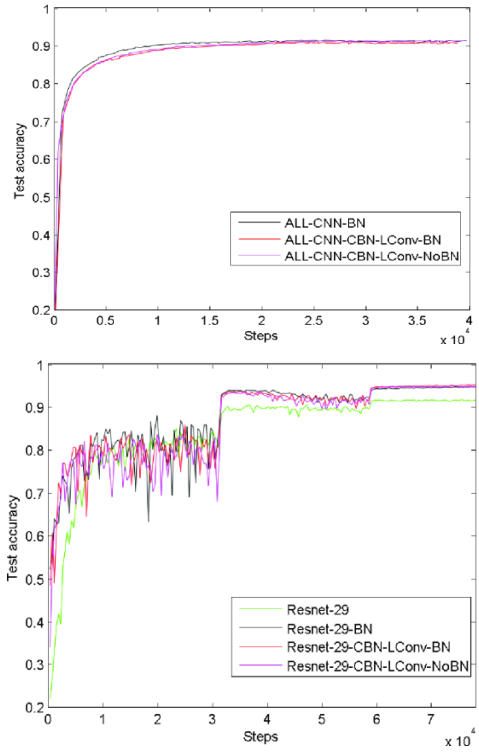

Figure 5. Subgraphs show the accuracy of ALL-CNNs and ResNets over CIFAR-10.

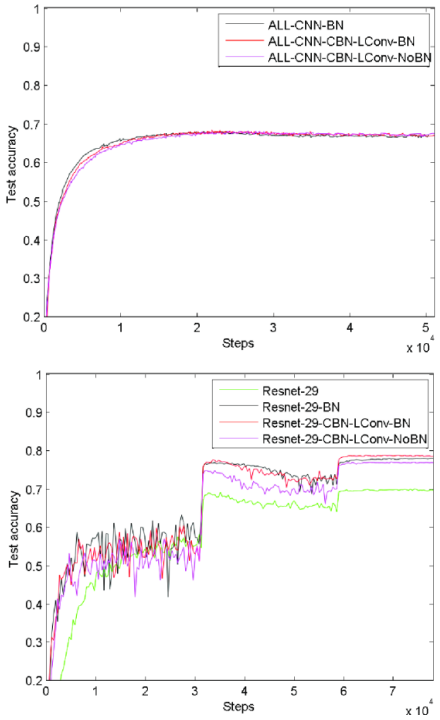

Figure 6. Subgraphs show the accuracy of ALL-CNNs and ResNets over CIFAR-100.

4.2 Do We Need to Put BN After the Last Convolutional Layer?

Whenever CBNs ($BN_{1,0}$) are employed for all convolutional layers, experiments show some performance degradation compared to the use of BNs. If we simply remove the CBN layer following the last convolutional layer, we can achieve almost the same performance as that of the standard BN architecture, as shown in Table 2 for both ALL-CNNs and ResNets.

4.3 Activation Functions

We also investigate the use of different activation functions (Relu, Prelu, Tanh [4]), along with CBN for ALL-CNNs, on the test accuracy. With CIFAR-10 dataset, the results are listed in Table 3. In summary, we found that the proposed CBN-LConv-NoBN structure do work well for almost all activation functions. In addition, we find that CBN does not perform well on a saturated function like tanh.

Table 3. The test accuracy of different activation functions.

	Relu	Prelu	tanh
BN	0.912	0.904	0.848
CBN-LConv-NoBN	0.912	0.906	0.773l

4.4 Trainable Parameters

It was pointed out in [3] that the use of BN often results into the increase of about 30% in computation overhead at each iteration. With CBN, there is no need to update two learnable parameters, namely, gain and bias. Therefore, the number of trainable

parameters is correspondingly reduced, which is shown in Table 4 for ResNet-29 and ResNet-101. The train accuracy versus wall-clock-time graph is shown in Figure 7 for ResNet-101. For this very deep neural net, it clearly requires less time for training with the use of CBNs.

Table 4. Parameters of ALL-CNN and BN on CIFAR-100.

	ResNet-29	ResNet-101
No-BN	param=4222666 (16.11 MB)	param=39638730 (151.21 MB)
BN	param=4238922 (16.17 MB)	param=39736394 (151.58 MB)
CBN-LConv-BN	param=4224714 (16.12 MB)	param=39642826 (151.23 MB)

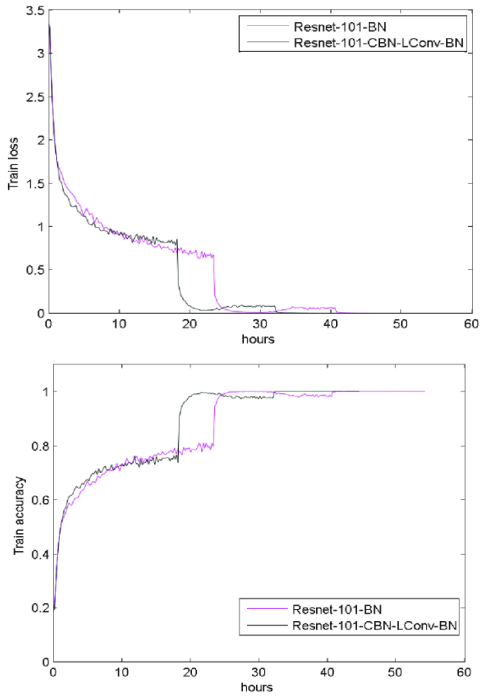

Figure 7. The training loss (top), accuracy (bottom) versus wall-clock-time for ResNet-101 on CIFAR-10.

5. CONCLUDING REMARKS

As an efficient approach for overcoming gradient vanishing /exploding, batch normalization has become a standard ingredient for constructing DNNs. In this paper, we argue that it is not necessary to adjust the learnable gain and bias along the training process. Through extensive experiments, we show that the use of constant gain and bias (CBN) is enough. In our view, the use of CBN can force the layer's outputs (or features) locating in a standard value range with zero-mean and unit-variance. As the learnable parameters do have little effect on the performance, it is interesting to ask if we can find a equivalent normalization approach without resorting to batch statistics. This is very important for RNNs and generating models.

6. ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grants 61372123, 61701252, 61771256, 61401228, China Postdoctoral Science Foundation under Grant 2015M581841, Postdoctoral Science Foundation of Jiangsu Province under Grant 1501019A, Open Research Fund of National Engineering Research Center of Communications and Networking of Nanjing University of Posts and Telcommunications under Grant TXKY17009, and the Scientific Research Foundation of Nanjing University of Posts and Telecommunications under Grant NY213002.

7. REFERENCES

[1] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[2] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[3] D. Mishkin and J. Matas. All you need is a good init. *arXiv preprint arXiv:*1511.06422, 2015.

[4] E. Fan. Extended tanh-function method and its applications to nonlinear equations. *Physics Letters A*, 277(4):212–218, 2000.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *stat*, 1050:21, 2016.

[7] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* , pages 770–778, 2016.

[9] Lin, Min, Q. Chen, and S. Yan. Network In Network. *Compu*ter *Science*(2013).

[10] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[12] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[14] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.