# Calibrated Stochastic Gradient Descent for Convolutional Neural Networks

**Li'an Zhuo,[1] Baochang Zhang,[1*] Chen Chen,[2] Qixiang Ye,[*5] Jianzhuang Liu,[4] David Doermann[3]**
[1]School of Automation Science and Electrical Engineering, Beihang University, Beijing
[2]University of North Carolina at Charlotte, Charlotte, NC
[3]Department of Computer Science and Engineering University at Buffalo, Buffalo, NY
[4]Huawei Noah's Ark Lab
[5]University of Chinese Academy of Sciences, China
{lianzhuo, bczhang}@buaa.edu.cn, chenchen870713@gmail.com

## Abstract

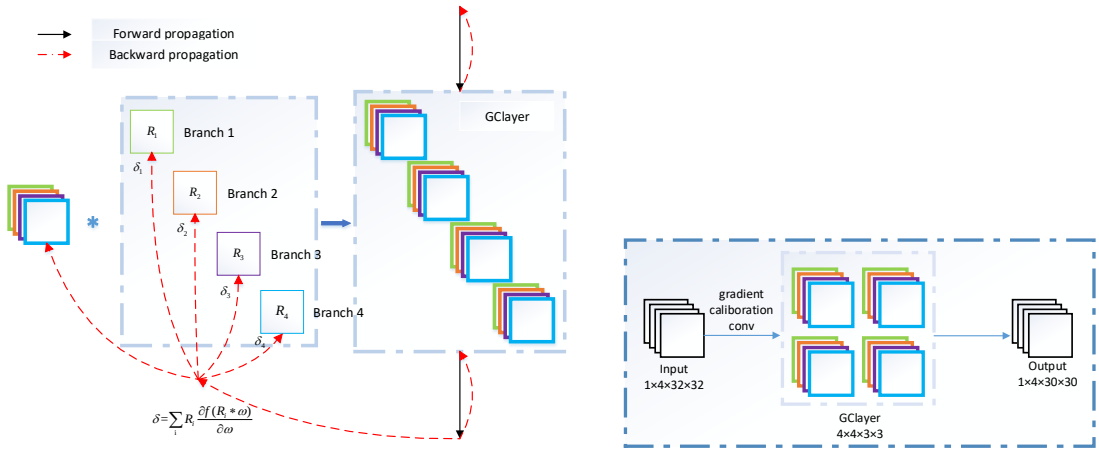
In stochastic gradient descent (SGD) and its variants, the optimized gradient estimators may be as expensive to compute as the true gradient in many scenarios. This paper introduces a calibrated stochastic gradient descent (CSGD) algorithm for deep neural network optimization. A theorem is developed to prove that an unbiased estimator for the network variables can be obtained in a probabilistic way based on the Lipschitz hypothesis. Our work is significantly distinct from existing gradient optimization methods, by providing a theoretical framework for unbiased variable estimation in the deep learning paradigm to optimize the model parameter calculation. In particular, we develop a generic gradient calibration layer which can be easily used to build convolutional neural networks (CNNs). Experimental results demonstrate that CNNs with our CSGD optimization scheme can improve the state-of-the-art performance for natural image classification, digit recognition, ImageNet object classification, and object detection tasks. This work opens new research directions for developing more efficient SGD updates and analyzing the back-propagation algorithm.

## Introduction

Back-propagation (BP) is one of the most popular algorithms for optimization and by far the most important way to train neural networks. The essence of BP is that the gradient descent algorithm optimizes the neural network parameters by calculating the minimum value of a loss function. Previous studies have focused on optimizing the gradient descent algorithm to make the loss decrease faster and more stable (Kingma and Ba 2014) (Dozat 2016) (Zeiler 2012). These algorithms, however, are often used as black-box optimizers, so a theoretical explanation of their strengths and weaknesses is hard to quantify (Bau et al. 2017).

*The work was supported by the Natural Science Foundation of China under Contract 61672079 and 61473086, and Shenzhen Peacock Plan KQTD2016112515134654. This work is supported by the Open Projects Program of National Laboratory of Pattern Recognition. Baochang Zhang and Qixiang Ye are the corresponding authors. Beijing Municipal Science & Technology Commission under Grant Z181100008918014 and NSFC under Grant 61836012. Baochang Zhang is also with Shenzhen Academy of Aerospace Technology, Shenzhen, China.


Many improvements have been made on the basic gradient descent algorithm (Ruder 2016), including batch gradient descent (BGD), stochastic gradient descent (SGD) and mini-batch gradient descent (MBGD). MBGD takes the best of both BGD and SGD and performs an update with every mini-batch of training examples. It is typically the algorithm of choice when training a neural network and the term SGD is often employed when mini-batches are used (Zhang, Choromanska, and Lecun 2015). The gradient oscillation of SGD, on the one hand, enables it to jump to a new and potentially better local minimum, but this ultimately complicates the convergence because it can cause overshooting. To circumvent this problem, by slowly decreasing the learning rate, SGD shows similar convergence behavior as BGD, converging close to a local or global minimum for non-convex and convex optimization respectively (Dauphin et al. 2014). An unbiased gradient estimator based on the likelihood-ratio method is introduced in (Gu et al. 2016) to estimate a stable gradient, which however is implemented based on complex mean-field networks that cause inefficiency for model calculation. In (Soudry, Hubara, and Meir 2014), the expectation BP is introduced to optimize the neural network calculation only when a prior distribution is given to approximate posteriors in the Bayesian inference. In (Zhang, Kjellström, and Stephan 2017), a mini-batch diversification scheme for SGD is introduced based on a similarity measure between data points. It gives lower probabilities to mini-batches which contain redundant data, and higher probabilities to mini-batches with more diverse data. Biased gradient schemes (Zhang, Kjellström, and Stephan 2017) (Qian 1999) may reduce the stochastic gradient noise or ease the optimization problem, which could lead to faster convergence. However, the biased estimators are heuristic and difficult to enumerate the situations in which these estimators will work well (Zhang, Kjellström, and Stephan 2017). These algorithms prove to be effective in their engineering applications. However, existing methods have following limitations: (1) they focus on unbiased or biased gradient estimators which rely on a prior knowledge about model optimization or a heuristic method; (2) **unbiased variable estimation** could be used to understand CNNs better, which is however neglected in prior arts. In this paper, we provide a theoretical framework for unbiased variable estimation to optimize the CNNs model parameters in an end-to-




(a) The CSGD procedure ($i$ = 1,...,4 in this example). (b) Gradient calibration convolution

Figure 1: The illustration of the CSGD procedure and gradient calibration convolution in our CCNNs. We can see that the numbers of input and output channels in our convolution are the same, which is used to build gradient calibration layer ($GClayer$) that is generic and easily implemented by simply replicating the same module at each layer. $R$ is shared within each $GClayer$, and thus the new layer can be implemented in low complexity.

end manner. In particular, we do not impose any restriction on the gradient estimator (i.e., biased or unbiased) nor any prior knowledge on the model parameters, which makes our framework more general and thus the performance in practice could be guaranteed.

In this paper, we introduce an calibrated SGD (CSGD) algorithm for stable and efficient training of CNNs. We first develop a theory showing that an unbiased variable estimator can be achieved in a probabilistic way in SGD, which provides a guarantee on the performance in practice and also poses a new direction to analyze the BP algorithm. In particular, we compute the gradient based on a statistic analysis on the importance of each branch of the gradient among training samples. The statistic can be done in the BP framework by designing a generic gradient calibration layer (GClayer), which can be easily incorporated into any CNNs architectures without bells and whistles. We refer to the CNNs based on our CSGD as CCNNs in the following.

**Distinctions between this work and prior art.** In (Soudry, Hubara, and Meir 2014), based on the expectation propagation, the posterior of the weights given the data is approximated using a "mean-field" factorized distribution in an online setting. Differently, ours is automatically calculated in the BP framework. Unlike an analytical approximation to the Bayes update of this posterior, we develop a theory aiming to understand SGD in terms of unbiased estimator. Our work is also different from (Gu et al. 2016) which designs stochastic neural networks based on unbiased BP, only when the distribution is given, i.e., based on Bernoulli and multinomial distributions. Ours is more flexible, without a prior hypothesis on the distribution, which provides a generic convolutional layer to optimise the kernel weights of CNNs. The main contributions of this work are three-fold.

- A theorem is developed to reveal that an unbiased variable estimator can be obtained in a probabilistic way based on a Lipschitz assumption, leading to a calibrated SGD algorithm (CSGD) for optimizing the kernel weights of CNNs.
- We develop a generic gradient calibration layer (GClayer) to achieve the proposed unbiased variable estimation, which can be easily applied to existing CNN architectures, such as AlexNet, ResNets and Wide-ResNet (WRN).
- Experimental results have demonstrated that popular CNN architectures optimised by the proposed CSGD algorithm, dubbed as CCNNs, yield state-of-the-art performance for a variety of tasks such as digit recognition, ImageNet classification and object detection.

## Calibrated Stochastic Gradient Descent

Gradient descent is based on the observation that if a function $f(w)$ is defined and differentiable in a neighborhood of a point, then $f(w)$ decreases fastest if one goes from a given position in the direction of the negative gradient of $f(w)$. It follows that:

$$w^{t+1} = w^t - \theta\delta,$$

where $\theta$ is the learning rate and $\delta$ is the gradient vector. The popular gradient descent method, SGD, performs frequent updates with a high variance that causes the loss to fluctuate widely during training. To aovid this, we project the gradients onto a subspace, which calibrates the objective to obtain a stable solution. The gradient vector $\delta$ is calculated based on the expectation method as:

$$\delta = \sum_{i}^{K} R_i * \delta_i, \tag{1}$$

where $*$ is the Schur product, an element-wise multiplication operator, and $\delta_i$ spans a subspace $\Omega = \{\delta_1, \delta_2, ..., \delta_K\}$ also denoting $K$ branches of gradients. Each element of $R_i$ denotes the probability of its corresponding element in $\delta_i$, which measures $\delta_i$'s importance or contribution to the final $\delta$. The challenge is how to build $\Omega$ and we do so with a new and efficient method. As mentioned, $R_i$ is the measure of the importance of each element in $\Omega$. We further use it to weigh $w$, which means that the more important $R_i$ is, the larger corresponding weight is imposed on $w$. We then define:

$$\delta_i = \frac{\partial f(R_i * w)}{\partial w},$$

where $\delta_i$ is a flow (or branch) of the gradient $\delta$ corresponding to $R_i$ as shown in Fig. 1. The derivative of Eq. 1 is finally obtained by:

$$\delta = \sum_i^K R_i \frac{\partial f(R_i * w)}{\partial w}. \tag{2}$$

That is, by using $f(R_i * w)$, we efficiently solve both $w$ and $\Omega$ in the same framework, which will be elaborated in the following. By designing a new and generic layer, Gradient Calibration layer ($GClayer$), the gradient calculation mentioned above can be easily implemented in the BP process. To better estimate the gradient we can add a Gaussian function $\mathbf{N}(0, \sigma)$ of 0 mean and variance $\sigma$ as the residual, which also makes the theoretical analysis an easier task.

### Implementation of the gradient calibration layer

We set $H_i = R_i * w$ , $i = 1, ..., K$, and $H = (H_1, ..., H_K)$, as the convolutional kernels. We implement $f(R_i * w)$ based on a new layer, which is generic and can be independently used for any network, e.g., CNNs.

$$F^{l+1} = GClayer(F^l, H), \tag{3}$$

where $F^l$ stands for the feature map for the $l$th layer. Note here we omit the layer index (i.e., superscript) for $H$ for simplicity. $GClayer$ denotes the convolutional operation implemented as a new layer or module. A simple example of the forward process is shown in Fig. 1. In the GC convolution, channels of one output feature map are generated as follows:

$$F_k^{l+1} = F_k^l \otimes H_k, \tag{4}$$

where $k \in \{1, ..., K\}$. Let the size of the input feature map be $4 \times 32 \times 32$ with $K = 4$, where a duplication process is only performed by $K$ times on the one channel of input gray-level images. The size of the output feature map is $4 \times 30 \times 30$. We can see that the number of input and output channels in every feature map are the same as shown in Fig. 1 (b), so that $GClayer$ can be easily implemented by simply replicating the same module at each layer. Note that Eq. 1 can then be automatically implemented in BP based on $GClayer$ by estimating $R_i$ elaborated in the following.

### Updating $R_i$

We update $R_i^{t+1}$ based on $R_i^t$ and $\hat{\delta}_i = \frac{\partial f(R_i * w)}{\partial R_i}$. We assume the elements of $R_i$ are probabilities $\in [0, 1]$ and $\sum_i R_i = \mathbf{1}$, where $\mathbf{1}$ denotes a vector with all elements equal to 1. We update $R_i$ during BP as:

$$R_i^{t+1} = |R_i^t + \ell \hat{\delta}_i|.$$

We further normalize $R_i$ such that $\sum_i R_i = \mathbf{1}$. We note that $R_i$ is shared within each layer, i.e., adding only $K \times 3 \times 3$ parameters to each layer, whose index is drop for ease of presentation. This means the number of the additional parameters is much smaller than that in the original filters, and thus $GClayer$ can be implemented in low complexity. Our CSGD algorithm is summarized in Alg. 1. It is based on the BP framework, but unlike conventional methods, ours is initially based on the expectation of gradient and ultimately obtains an unbiased variable estimator as discussed below. It poses a new direction to analyze the BP algorithm. To obtain a better understanding of learning algorithms, the Lipschitz distribution is widely used for a theoretical analysis of neural networks. For instance, in CNNs (Zou, Balan, and Singh 2018), the Lipschitz bound is important in the study of the stability and the computation of the Lipschitz bound is used for generative networks. In (Zou, Balan, and Singh 2018) the authors give a general framework for CNNs and prove that the Lipschitz bound of a CNN can be determined by solving a linear program with a more explicit expression for a suboptimal bound. In light of this, we theoretically show that an unbiased variable estimator can be achieved in CSGD with a Lipschitz assumption in a probabilistic way.

**Algorithm 1:** The CSGD algorithm

1: Set $t = 0$
2: Initialize $w^t$ and $R_i^t$, $i = 1, 2, ..., K$
3: Initialize the learning rates $\theta$ and $\ell$.
4: **repeat**
5: $t = t + 1$;
6: Update $w^{t+1} = w^t - \theta \sum_i R_i^t \frac{\partial f(R_i^t * w^t)}{\partial w^t} + \mathbf{N}(0, \sigma)$;
7: Update $R_i^{t+1} = |R_i^t + \ell \frac{\partial f(R_i^t * w^t)}{\partial R_i^t}|$;
8: Normalize $R$;
9: **until** convergence

## Theoretical analysis

Until now, we have developed a new BP algorithm that introduces the expectation of gradient into the learning process, which ultimately leads to an unbiased variable estimator. In the following, we show that our proposed CSGD can lead the average of the input to the expectation, that is, an unbiased estimator. More specifically, our theorem shows that an unbiased estimator can be obtained in a probabilistic way based on the Lipschitz hypothesis, if a convergence is achieved during training. Such a proof would be very useful to guide a new gradient descent algorithm design in various practical applications, since the exploration of the unbiased variable estimation provides a different investigation into the

BP algorithm from conventional methods. We address how our theorem can be used in the learning stage.

**Definition 1:** Let $x_1, x_2, ..., x_n$ be c-Lipschitz. Then we have:

$$|x_{i-1} - x_i| \leq c_{i-1}, \tag{5}$$

where $x_i$ is a 1D random variable and $c = (c_0, c_1, ..., c_{n-1})$.

**Lemma 1:** For any vector $A$, it follows that:

$$P(\sum_{i=1}^{n} |A_i| \geq \sum_{i=1} \lambda_i) \leq P(\bigcup_{i=1} |A_i| \geq \lambda_i) \leq \sum_{i} P(|A_i| \geq \lambda_i). \tag{6}$$

where $P$ stands for probability. Lemma 1 is obvious. Next, we introduce Lemma 2 and its proof, which will be used in the proof of our theorem.

**Lemma 2:** For a batch set, we first define a loss function based on the $N$ input samples as:

$$f(w) = \frac{1}{N} \sum_{n=1}^{N} f_n(w). \tag{7}$$

Based on gradient descent, we have:

$$E(w^{t+1}) = E(w^t) - \theta \bigtriangledown f(w^t). \tag{8}$$

**Proof:**

We begin to prove Lemma 2 by a distribution ($\hat{U}$, e.g., uniform in SGD) hypothesis on the input data. We have:

$$\begin{aligned} E_{n\sim\hat{U}}[\bigtriangledown f_n(w)] &= \bigtriangledown E_{n\sim\hat{U}}[f_n(w)] \\ &= \bigtriangledown \sum_{i=1}^{N} \hat{U}(n=i) f_i(w) \\ &= \bigtriangledown f(w). \end{aligned} \tag{9}$$

In particular for SGD with a uniform distribution, we have $\bigtriangledown f(w) = \bigtriangledown \frac{1}{N} \sum_{i=1}^{N} f_i(w)$.

Then, a random sample point $n \sim \hat{U}$ is chosen to update the weights based on gradient descent:

$$w^{t+1} = w^t - \theta \bigtriangledown f_n(w^t), \tag{10}$$

and we obtain:

$$E(w^{t+1}) = E(w^t) - \theta E(\bigtriangledown f_n(w^t)). \tag{11}$$

Based on Eq. 9, we have

$$E(w^{t+1}) = E(w^t) - \theta \bigtriangledown f(w^t). \tag{12}$$

Thus, Lemma 2 is proved. □

**Lemma 3:** If $x_1, x_2, ..., x_n$ satisfies Definition 1, then:

$$P(|\bar{x} - E(\bar{x})| \geq \frac{\lambda}{n}) \leq 2e^{\frac{-\lambda^2}{\sum_{i=1}^{n} c_i^2}}, \tag{13}$$

where $x$ is a 1D variable updated via gradient descent (Gaussian noise) to minimize $f(x)$ with $\bigtriangledown f(x) = 0$ when converging, $\bar{x}$ is the average of $x_1, x_2, ..., x_n$, and $c_i$ is predefined based the c-Lipschitz hypothesis. The proof of Lemma 3 can refer to our technical report which will be in https://github.com/bczhangbczhang/.

**Theorem** Let $Y$ be a vector random variable updated based on gradient descent to minimize $f(Y)$. For a set of samples $Y_1, Y_2, ..., Y_n$ satisfying Definition 1, if $\bigtriangledown f(Y) = 0$, then an unbiased estimator is achieved by Eq. 2 in a probabilistic way, that is:

$$P(|\bar{Y} - E(\bar{Y})| \geq \sum_{i} \lambda_i) \leq \sum_{i=1}^{n} a_i, \tag{14}$$

where $\bar{Y}$ is the mean vector, $\lambda = (\lambda_1, \lambda_2, ..., \lambda_n)$ with $\lambda_i \leq 1$, and $a_i = 2e^{\frac{-\lambda_i^2}{2\sum_{j=1}^{n} C^2(i,j-1)}}$ for a matrix $C$.

**Proof:** our theorem means that the expectation of $\bar{Y}$ is achieved in a probabilistic way, given that $\bar{Y}$ is a variable updated based on the CSGD algorithm. Before proving the theorem, we introduce the Lipschitz assumption on the $i$th dimension of $Y_j$ that:

$$||Y_j^i - Y_{j-1}^i|| \leq C(i, j-1), \tag{15}$$

which could be easily satisfied in the learning process.

According to Lemma 3, Eq. 2 and Eq. 15, we have:

$$P(|Z^i| \geq \lambda_i) \leq 2e^{\frac{-\lambda_i^2}{2\sum_{j=1}^{n} C^2(i,j-1)}}, \tag{16}$$

where $Z = \bar{Y} - E(\bar{Y})$. Let $a_i = 2e^{\frac{-\lambda_i^2}{2\sum_{j=1}^{n} C^2(i,j-1)}}$. Based on Lemma 1 we have:

$$P(|\bar{Y} - E(\bar{Y})| \geq \sum_{i=1}^{n} \lambda_i) \leq \sum_{i=1}^{n} a_i. \tag{17}$$

Thus, our theorem is proved. □

We note that $\bigtriangledown f(Y) = 0$ could be satisfied when the algorithm converges. Thus, our theorem does not require that the gradient estimator is unbiased or not, which make it more general.

## Implementation and Experiments

In this section, we evaluate our CSGD algorithm for CNNs on several benchmark datasets including CIFAR-10, CIFAR-100 (Krizhevsky 2009). In addition, PASCAL VOC 2007 is also used to validate the effectiveness of our network optimization approach on the object detection task. The network architectures we used for evaluation include Wide ResNets (WRN), ResNets and AlexNets.

**Backbone network architectures.** ResNets are introduced to ease the training of networks that are substantially deeper than those used previously. The ResNets architecture utilizes skip connections or short-cuts to jump over some layers, which can partially avoid the gradient vanishing problem. WRN is a network structure similar to ResNets, and it introduces a novel architecture by decreasing the depth and increasing the width of residual networks. We follow the same layer settings detailed in (Zagoruyko and Komodakis 2016). AlexNet is one of the most famous CNNs, which is used as the backbone of our method for the object detection task. It contains eight layers: the first

five are convolutional layers, and the last three are fully connected layers. Based on these network architectures, we build our CNNs using the proposed CSGD algorithm (denoted as CCNNs). Their performances are extensively validated in the following experiments.

## Experiments on natural image classification

For the natural image classification task, we use the CIFAR-10 and CIFAR-100 datasets (Krizhevsky 2009) which consist of 60,000 color images of size 32x32 in 10 or 100 classes, with 600 or 6,000 images per class. There are 50,000 training images and 10,000 test images.

We first evaluate $K$ on the performance of our CCNNs on CIFAR-10 by replacing the convolution layers of ResNets-18 with our GClayers. The results show that the performance becomes better when increasing $K$ as shown in Table 1. $\sigma$ in Eq. 2 is also tested and the results in the same table show that the performance becomes better when decreasing $\sigma$, e.g., 7.61% vs. 7.68% when $\sigma = 0.0001$ vs. $\sigma = 0.001$. We choose $K = 4$ and $\sigma = 0.0001$ in all the following experiments, considering that it already achieves much better performance than ResNet-18 with Gaussian noise ((Neelakantan et al. 2015)). Moreover, compared with $\sigma = 0$ denoting the original ResNets, Gaussian noise benefits the final performance in terms of the error rate. We also plot the training and testing error curves of CCNNs-18 in Fig. 2, which show that our CCNNs achieve more stable and better training and test results with a slight faster convergence speed than the original ResNet-18.

We further use WRNs to test CCNNs on the datasets. We replace the convolution layers with our GClayers and set up 40-layer and 28-layer networks with the same basic blocks and hyper-parameters as WRNs. The network stages are 16-32-64-128 and 64-64-128-256. The details of the CCNNs architecture are presented in Table 2. We use a weight decay of 0.0001 and momentum of 0.9. These models are trained on 4 GPUs (Titan XP) with a mini-batch size of 128. The training procedure is terminated at 64k iterations, which is determined based on a 45k/5k train/validation split. We follow the same data augmentation strategy in (Zou, Balan, and Singh 2018) for training. Horizontal flipping is adopted, and a $32 \times 32$ crop is sampled randomly from the image padded with 4 pixels on each side. For testing, we only evaluate the single view of the original $32 \times 32$ image.

We conduct the experiments to compare CCNNs with the state-of-the-art networks (*i.e.* NIN (Boureau, Ponce, and LeCun 2010), VGG (Simonyan and Zisserman 2014), and ResNet (He et al. 2015) in terms of error rate and the amount of parameters. On CIFAR-10, Table 2 shows that CCNNs consistently improve the performance regardless of the number of parameters or kernels as compared with the baseline ResNet. We further compare CCNNs with the Wide Residue network (WRN) (Zagoruyko and Komodakis 2016), and again CCNNs achieve a better result ($3.81\%$ vs. $4\%$ error rate). Our model is also half the size of WRN, providing a significant advantage in terms of model efficiency.

Similar to CIFAR-10, one can also observe the performance improvement on CIFAR-100, with similar parameter sizes.

Table 1: Results (error rate (%) on CIFAR-10). CCNNs are based on ResNets-18, which has a smaller network stage (16-16-32-64). * denotes ResNet with Gaussian Noise (Neelakantan et al. 2015)).

| Model | $K$ | 1 | 2 | 4 | 8 |
|---|---|---|---|---|---|
| CCNNs | $\sigma = 0$ | 9.68 | 8.52 | 7.82 | 7.63 |
| | $\sigma = 0.001$ | 9.60 | 8.49 | 7.68 | 7.54 |
| | $\sigma = 0.0001$ | 9.48 | 8.32 | 7.61 | 7.42 |
| ResNet-18* | $\sigma = 0.001$ | - | - | 9.72 | - |

## Large-scale image classification: ImageNet

To show the effectiveness of CCNNs on larger images, we evaluate the network on the ImageNet (Deng et al. 2009) dataset. ImageNet consists of images with a much higher resolution. In addition, the images usually contain more than one attribute per image, which may have a large impact on the classification accuracy.

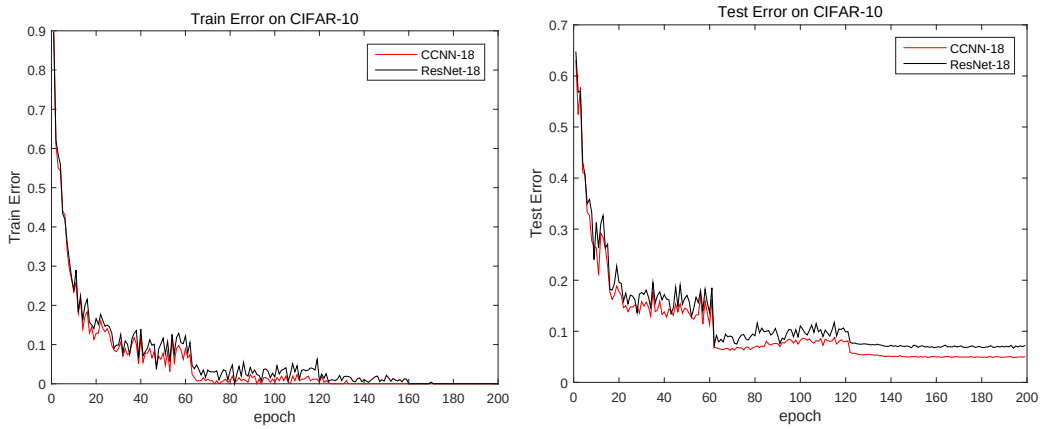
For the ImageNet experiment, we train 18-layer CCNNs based on ResNets-18. CCNNs and ResNet are trained with 120 epochs. The learning rate is initialized as 0.1 and decreased to 1/10 of the previous size every 15 epochs. Top-1 and Top-5 errors are used as evaluation metrics. The results are shown in Table 3. Compared to the baseline ResNet-18, our CCNNs achieve better classification performances (*i.e.,* Top-1 error: 29.2% vs. 30.7%, and Top-5 error: 10.3% vs. 10.8%) , which further validates the effectiveness of our method.

## Experiments on object detection

Object detection is one of the fundamental problems in computer vision, which aims to detect all instances of objects from known classes, such as people and cars in images. It has various real-world applications, ranging from robotics, autonomous car, to video surveillance and image retrieval. It is very challenging due to the severe scale variation, viewpoint change, intra-class variation, shape variation, and occlusion of objects, as well as background clutters.

**PASCAL VOC 2007 dataset.** It consists of $2,501$ training, $2,510$ validation, and $4,092$ test images with bounding box annotations for 20 categories (Everingham 2007). We use both training and validation sets for training and evaluate the detection performance on the test set in terms of the mean average precision (mAP) following the standard PASCAL VOC protocol, which reports average precision (AP) at $50\%$ intersection-over-union (IoU) of the detected boxes with the ground-truth.

We incorporate our CCNNs into two-stage Faster R-CNN (Ren et al. 2015) which is implemented based on the Caffe2 platform. In CCNNs, AlexNet (Krizhevsky, Sutskever, and Hinton 2012) or ResNet-18 is used as the backbone network that is first pre-trained on the ILSVRC CLS-LOC dataset (Russakovsky et al. 2015). We train Faster R-CNN using the approximate joint training method with the effective mini-batch size of $4$. For anchors, we use $4$ scales with box areas of $64 \times 64$, $128 \times 128$, $256 \times 256$, and $512 \times 512$ pixels, and 3 aspect ratios of $1:1$, $1:2$, and $2:1$. $256$ anchors




(a) Training error on CIFAR-10 (b) Test error on CIFAR-10

Figure 2: Training error and test error curves on CIFAR-10 dataset. Compared with the baseline, CCNNs achieve a slight faster convergence speed and lower training and testing errors.

Table 2: Comparison results on CIFAR-10 and CIFAR-100 datasets. $R$ is neglected when counting the parameter size of CCNNs. CCNNs are based on WRNs.

| Method | | | error rate(%) | |
|---|---|---|---|---|
| | | | CIFAR-10 | CIFAR-100 |
| NIN | | | 8.81 | 35.67 |
| VGG | | | 6.32 | 28.49 |
| | # network stage kernels | # params | | |
| ResNet-110 | 16-16-32-64 | 1.7M | 6.43 | 25.16 |
| ResNet-1202 | 16-16-32-64 | 10.2M | 7.83 | 27.82 |
| WRN-40 | 64-64-128-256 | 8.9M | 4.53 | 21.18 |
| WRN-28 | 160-160-320-640 | 36.5M | 4.00 | 19.25 |
| CCNNs2-40 | 16-64-128-256 | 17.9M | 4.62 | 21.67 |
| CCNNs3-28 | 64-64-128-256 | 17.6M | 3.81 | 19.11 |

Table 3: Results on ImageNet dataset. CCNNs are based on ResNets $K = 2$.

| Models | accuracy (%) | |
|---|---|---|
| | Top-1 | Top-5 |
| ResNet-18 | 69.3 | 89.2 |
| CCNNs-18 | 70.8 | 89.7 |

are then randomly sampled in an image to compute the loss function of the region proposal network (RPN). We re-scale the images such that their shorter side is 600 pixels. A learning rate of $0.004$ for $12.5k$ mini-batches, and $0.0004$ for the next $5k$ mini-batches, a momentum of $0.9$ and a weight decay of $0.0005$ are used.

With the mAP measure in Table 4, the results clearly show that AlexNet-CCNNs achieve a better performance (1.1%) than the original Faster RCNN. On ResNet-18, we can observe a similar phenomenon where ResNet-CCNNs outperform the baseline ResNet. Considering that ResNets are widely used in various real-world applications, our CCNNs are able to further boost thier performance, demonstrating the superiority of the proposed CSGD algorithm. Both CCNNs achieve much better performance than the backbone networks on *aero*, *bus*, *tv*, etc. In Fig. 3, we provide some detection examples based on ResNet-CCNNs, e.g., *tvmonitor*, *bus*, *bird*, and *person*. They demonstrate that CCNNs are effective for detection even though they could be misled by noisy background for localization.

## Conclusion

In this paper, an calibrated SGD (CSGD) algorithm for deep neural network optimization has been proposed by providing a theoretical investigation to an unbiased variable estimator based on a Lipschitz assumption. CSGD is of high efficiency based on a mini-batch of samples as SGD. We further develop a generic gradient calibration layer, which can be easily incorporated into any deep neural networks, e.g., CNNs, with only a small fraction of the network parameters added. Extensive experiments and comparisons on the commonly used benchmarks show that the proposed CSGD algorithm can effectively improve the performance of the

Table 4: Comparison to the state-of-the-art methods on PASCAL VOC 2007 in terms of mAP (%) on the test set.

| method | bbone | mAP | aero | bic | bir | boa | bot | bus | car | cat | cha | cow | dta | dog | hor | mbi | per | pln | she | sof | tra | tv |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F. R-CNN | Alex | 51.5 | 56.4 | 63.6 | 47.2 | 34.3 | 31.3 | 54.0 | 69.4 | 60.6 | 31.0 | 56.3 | 47.4 | 56.2 | 68.6 | 62.2 | 60.0 | 26.5 | 50.6 | 40.0 | 59.6 | 54.4 |
| | CCNNs | **52.6** | 58.4 | 61.0 | 48.7 | 34.3 | 28.2 | 57.6 | 69.4 | 61.3 | 31.5 | 59.9 | 51.0 | 54.7 | 69.8 | 65.9 | 61.3 | 28.2 | 48.9 | 40.8 | 66.0 | 55.7 |
| F. R-CNN | Res18 | 68.3 | 69.2 | 79.3 | 65.8 | 54.0 | 47.4 | 77.7 | 79.0 | 78.5 | 52.3 | 73.3 | 63.4 | 73.7 | 80.0 | 74.8 | 77.4 | 45.1 | 64.6 | 69.6 | 75.1 | 65.8 |
| | CCNNs | 68.71 | 71.5 | 78.5 | 63.3 | 54.7 | 47.9 | 78.1 | 80.3 | 75.8 | 52.8 | 73.1 | 63.5 | 73.7 | 81.2 | 74.4 | 77.1 | 43.6 | 63.9 | 68.9 | 76.4 | 67.5 |

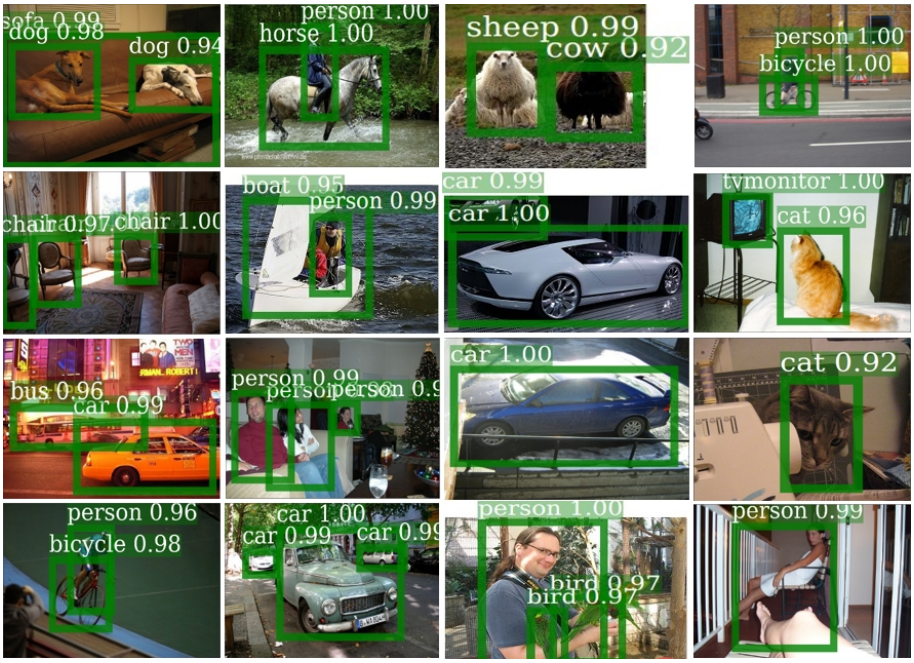



Figure 3: Examples of Faster RCNN using CCNNs (ResNets as the backbone) on the VOC2007 dataset.

popular CNNs, e.g. ResNets, leading to new state-of-the-art results on the benchmarks.

## References


Bau, D.; Zhou, B.; Khosla, A.; Oliva, A.; and Torralba, A. 2017. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 3319–3327.

Boureau, Y.-L.; Ponce, J.; and LeCun, Y. 2010. A theoretical analysis of feature pooling in visual recognition. *International Conference on Machine Learning* 111–118.

Dauphin, Y. N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; and Bengio, Y. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *International Conference on Neural Information Processing Systems*, 2933–2941.

Deng, J.; Dong, W.; Socher, R.; Li, L. J.; Li, K.; and Li, F. F. 2009. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

Dozat, T. 2016. Incorporating nesterov momentum into adam. In *International Conference on Learning Representations*, 1–8.

Everingham, M. 2007. The pascal visual object classes challenge, (voc2007) results. *http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/index.html.* 111(1):98–136.

Gu, S.; Levine, S.; Sutskever, I.; and Mnih, A. 2016. Muprop: Unbiased backpropagation for stochastic neural networks. In *ICLR*, 1861–1869.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385* 770–778.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *Computer Science*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems*, 1097–1105.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. *Tech Report*.

Neelakantan, A.; Vilnis, L.; Le, Q. V.; Sutskever, I.; Kaiser, L.; Kurach, K.; and Martens, J. 2015. Adding gradient noise improves learning for very deep networks. *Computer Science*.

Qian, N. 1999. Qian, n.: On the momentum term in gradient descent learning algorithms. *Neural Networks* 12(1):145–151.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans Pattern Anal Mach Intell* 39(6):1137–1149.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

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

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *Computer Science*.

Soudry, D.; Hubara, I.; and Meir, R. 2014. Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In *International Conference on Neural Information Processing Systems*, 963–971.

Zagoruyko, S., and Komodakis, N. 2016. Wide residual networks. *British Machine Vision Conference*.

Zeiler, M. D. 2012. Adadelta: An adaptive learning rate method. *arXiv preprint*.

Zhang, S.; Choromanska, A.; and Lecun, Y. 2015. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, 685–693.

Zhang, C.; Kjellström, H.; and Stephan, M. 2017. Determinantal point processes for mini-batch diversification. In *In the proceedings of Uncertainty in Artificial Intelligence (UAI)*, 1–8.

Zou, D.; Balan, R.; and Singh, M. 2018. On lipschitz bounds of general convolutional neural networks. *arXiv preprint arXiv:1808.01415*.