What Deep CNNs Benefit from Global Covariance Pooling: An Optimization Perspective

Qilong Wang[1], Li Zhang[1], Banggu Wu[1], Dongwei Ren[1], Peihua Li[2], Wangmeng Zuo[3], Qinghua Hu[1,*]

[1] Tianjin Key Lab of Machine Learning, College of Intelligence and Computing, Tianjin University, China
[2] Dalian University of Technology, China   [3] Harbin Institute of Technology, China

Abstract

Recent works have demonstrated that global covariance pooling (GCP) has the ability to improve performance of deep convolutional neural networks (CNNs) on visual classification task. Despite considerable advance, the reasons on effectiveness of GCP on deep CNNs have not been well studied. In this paper, we make an attempt to understand what deep CNNs benefit from GCP in a viewpoint of optimization. Specifically, we explore the effect of GCP on deep CNNs in terms of the Lipschitzness of optimization loss and the predictiveness of gradients, and show that GCP can make the optimization landscape more smooth and the gradients more predictive. Furthermore, we discuss the connection between GCP and second-order optimization for deep CNNs. More importantly, above findings can account for several merits of covariance pooling for training deep CNNs that have not been recognized previously or fully explored, including significant acceleration of network convergence (i.e., the networks trained with GCP can support rapid decay of learning rates, achieving favorable performance while significantly reducing number of training epochs), stronger robustness to distorted examples generated by image corruptions and perturbations, and good generalization ability to different vision tasks, e.g., object detection and instance segmentation. We conduct extensive experiments using various deep CNN models on diversified tasks, and the results provide strong support to our findings.

1. Introduction

Global covariance pooling (GCP) that is used to replace global average pooling (GAP) for aggregating the last convolution activations of deep convolutional neural networks
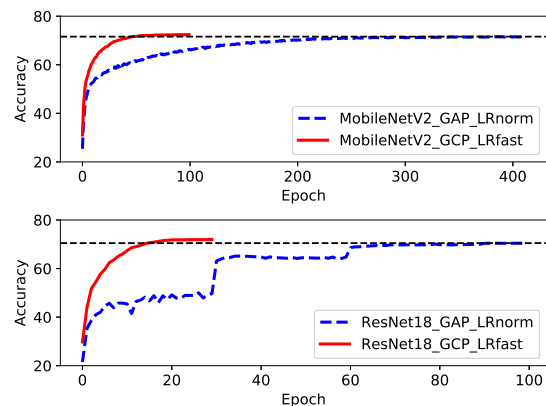
Figure 1. Convergence curves of MobileNetV2 [37] and ResNet-18 [16] with global average pooling (GAP) and global covariance pooling (GCP) on ImageNet. Note that the networks with GCP converge much faster while achieving matching or better results. We account for it from an optimization perspective (see Section 3).

(CNNs) has achieved remarkable performance gains on a variety of vision tasks [20, 29, 12, 40, 11, 26, 28, 25]. Existing GCP-based works mainly focus on obtaining better performance using various normalization methods [20, 29, 26, 28] and richer statistics [40, 9, 8, 3] or achieving comparable results with low-dimensional covariance representations [13, 22, 14, 41]. However, the reasons on effectiveness of GCP on deep CNNs have not been well studied. Although some works explain them from the perspectives of statistical modeling [20, 29, 26] or geometry [26], some behaviors of deep CNNs with GCP still lack of reasonable explanations. For example, as illustrated in Figure 1, why GCP can significantly speed up convergence of deep CNNs. Particularly, the networks with GCP can achieve matching or better performance than GAP-based ones, but only use less than $1/4$ of training epochs of the latter one.

In this paper, we make an attempt to understand the effectiveness of GCP on deep CNNs from an optimization perspective, thereby explaining behaviors of the networks with GCP in an intuitive way. To this end, we explore

the effect of GCP on optimization landscape and gradient computation of deep CNNs, inspired by recent work [38]. Specifically, we first train two widely used CNN models (i.e., MobileNetV2 [37] and ResNet-18 [16]) on large-scale ImageNet [10] using GAP and GCP for aggregating the last convolution activations, respectively. Then, we compare them from optimization perspective, and find that GCP can improve the stability of optimization loss (i.e., Lipschitzness) and the stability of gradients (i.e., predictiveness) over the commonly used GAP. Furthermore, by analyzing back-propagation of GCP and second-order optimization [31, 32, 35] in the context of deep CNNs, we make out that the influence of GCP on optimization shares some similar philosophy with K-FAC [32, 35]. Above findings provide an inspiring view to understand effectiveness of GCP on deep CNNs, which can intuitively and reasonably account for the behaviors of deep CNNs trained with GCP, e.g., GCP makes the optimization landscape more smooth, leading to network convergence to a better local minimum (i.e., better performance as shown in [29, 40, 26]) and much faster convergence as illustrated in Figure 1.

Based on the foregoing findings, we can explain several merits delivered by GCP for training deep CNNs that have not been recognized previously or fully explored. Firstly, since GCP is able to smoothen optimization landscape, deep CNNs with GCP can support rapid decay of learning rates for fast convergence. Meanwhile, previous work [38] shows that improvement of Lipschitzness can accelerate convergence of deep CNNs. To verify this point, we conduct experiments using a variety of deep CNN architectures (i.e., MobileNetV2 [37], ShuffleNet V2 [30] and ResNets [16]) on ImageNet [10]. The results show that, under the setting of rapid decay of learning rates, deep CNNs with GCP achieve comparable performance to GAP-based ones, while using less than a quarter of training epochs. By adjusting schedule of learning rates, deep CNNs with GCP can converge to better local minima using less training epochs.

Secondly, GCP improves the stability of both optimization loss and gradients, so it makes deep CNNs more robust to inputs perturbed by some distortions. Meanwhile, previous work [7] shows that control of the Lipschitz constant is helpful for improving robustness to examples with crafted distortions for confusing classifiers. Therefore, we experiment on recently introduced IMAGENET-C and IMAGENET-P benchmarks [17], where distortions are achieved by common image corruptions and perturbations. The results show that GCP can significantly improve robustness of deep CNNs to image corruptions and perturbations.

Thirdly, GCP usually allows deep CNNs converge to better local minima, and thus deep CNNs pre-trained with GCP can be exploited to provide an effective initialization model for other visual tasks. Therefore, we verify it by transferring the pre-trained CNN models to MS COCO benchmark [27]

for object detection and instance segmentation tasks, and the results indicate that pre-trained CNNs with GCP is superior to GAP-based ones.

The contributions of this paper are concluded as follows. (1) To our best knowledge, we make the first attempt to understand the effectiveness of GCP in the context of deep CNNs from an optimization perspective. Specifically, we show that GCP can improve the Lipschitzness of optimization loss and the predictiveness of gradients. Furthermore, we discuss the connection between GCP and second-order optimization. These findings provide an inspiring view to better understand the behaviors of deep CNNs trained with GCP. (2) More importantly, our findings above can explain several merits of GCP for training deep CNNs that have not been recognized previously or fully explored, including significant acceleration of convergence with rapid decay of learning rates, stronger robustness to distorted examples and good generalization ability to different vision tasks. (3) We conduct extensive experiments using six representative deep CNN architectures on image classification, object detection and instance segmentation, the results of which provide strong support to our findings.

2. Related Work

DeepO$_2$P [20] and B-CNN [29] are among the first works introducing GCP into deep CNNs. DeepO$_2$P extends the second-order (covariance) pooling method (O$_2$P) [5] to deep architecture, while B-CNN captures interactions of localized convolution features by a trainable bilinear pooling. Wang *et al*. [40] propose a global Gaussian distribution embedding network for utilizing the power of probability distribution modeling and deep learning jointly. Li *et al*. [26] present matrix power normalization for GCP and make clear its statistical and geometrical mechanisms. Beyond GCP, some researches [8, 3] propose to use richer statistics for further possible improvement. The aforementioned methods study GCP from perspectives of statistical modeling or Riemannian geometry. Different from them, we interpret the effectiveness of GCP from an optimization perspective, and further explore merits of GCP for training deep CNNs.

Since deep CNNs are black-boxes themselves, understanding effect of GCP on deep CNNs still is a challenging issue. Many recent works make efforts [42, 43, 38, 2, 44] towards understanding deep CNNs or analyzing the effect of fundamental components, e.g., individual units, batch normalization (BN) [19] and optimization algorithm. Specifically, Zeiler *et al*. [42] and Zhou *et al*. [44] visualize feature maps by deconvolution and image regions of maximal activation units, respectively. Zhang *et al*. [43] and Bjorck *et al*. [2] design a series of controlled experiments to analyze generalization ability of deep CNNs and understand effect of BN on deep CNNs, respectively. In particular, a recent work [38] investigates the effect of BN by exploring opti-

mization landscape of VGG-like networks [39] trained on CIFAR10 [23], while providing theoretical analysis on Lipschitzness improvement using a fully-connected layer. Motivated by [38], we make an attempt to understand *the effect of GCP on deep CNNs* from an optimization perspective.

3. An Optimization Perspective for GCP

In this section, we first revisit global covariance pooling (GCP) for deep CNNs. Then, we analyze smoothing effect of GCP on deep CNNs, and finally discuss its connection to second-order optimization.

3.1. Revisiting GCP

Let $\mathcal{X} \in \mathbb{R}^{W \times H \times D}$ be the output of the last convolution layer of deep CNNs, where W, H and D indicate width, height and dimension of feature map, respectively. To summarize $\mathcal{X}$ as global representations for final prediction, most existing CNNs employ GAP, i.e., $\sum_{i=1}^{N} \mathbf{X}_i$, where the feature tensor $\mathcal{X}$ is reshaped to a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ and $N = W \times H$. Many recent works [29, 26] demonstrate the superiority of GCP over GAP. To perform GCP, the sample covariance matrix of $\mathbf{X}$ can be calculated as

$$\mathbf{\Sigma} = \mathbf{X}^T \mathbf{J} \mathbf{X}, \ \mathbf{J} = \frac{1}{N}(\mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T), \qquad (1)$$

where $\mathbf{I}$ is the $N \times N$ identity matrix, and $\mathbf{1}$ is a N-dimensional vector of all elements being one.

Normalization plays an important role in GCP, and different normalization methods have been studied, including matrix logarithm normalization [20], element-wise power normalization followed by ℓ_2 normalization [29], matrix square-root normalization [40, 28, 25] and matrix power normalization [26]. Among them, matrix square-root (i.e., the power of $1/2$) normalization is preferred considering its promising performance on both large-scale and small-scale visual classification tasks. Therefore, this paper uses GCP with matrix square-root normalization, i.e.,

$$\mathbf{\Sigma}^{\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}^T, \qquad (2)$$

where $\mathbf{U}$ and $\mathbf{\Lambda}$ are the matrix of eigenvectors and the diagonal matrix of eigenvalues of $\mathbf{\Sigma}$, respectively.

3.2. Smoothing Effect of GCP

To understand the mechanism of GCP, we investigate effect of GCP on optimization landscape of deep CNNs. Inspired by [38], we explore effect of GCP on stability of optimization loss (i.e., Lipschitzness) and stability of gradients (i.e., predictiveness). Specifically, for examining stability of optimization loss, we measure how loss $\mathcal{L}$ changes along direction of current gradient at each training step. Given the input $\mathbf{X}$, variation of optimization loss is calculated as

$$\triangle_l = \mathcal{L}(\mathbf{X} + \eta_l \nabla\mathcal{L}(\mathbf{X})), \eta_l \in [a, b], \qquad (3)$$
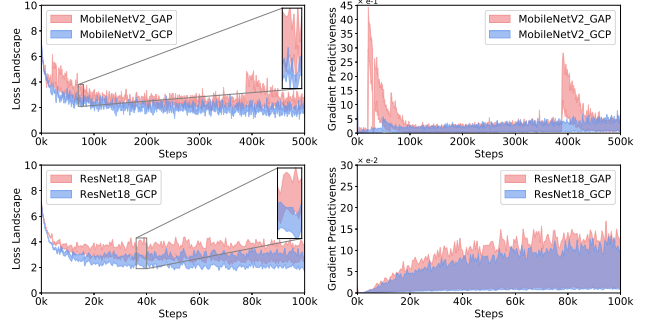

Figure 2. Comparison of (top) MobileNetV2 and (bottom) ResNet-18 trained with GCP or GAP in terms of (left) loss Lipschitzness and (right) gradient predictiveness. Detailed descriptions and discussions can be found in Section 3.2.

where $\nabla\mathcal{L}(\mathbf{X})$ indicates the gradient of loss with respect to the input $\mathbf{X}$, and η_l is step size of gradient descent. Clearly, smaller variation (range) of $\triangle_l$ indicates that optimization landscape is smoother and more easily controlled during training process [38]. As shown in [38], range of $\triangle_l$ in Eqn. (3) reflects the Lipschitzness of optimization loss.

To examine the stability of gradients, we measure how the gradient of loss $\mathcal{L}$ changes by computing the Euclidean distance between the gradient of loss and gradients along the original gradient direction on an interval of step sizes. Thus, the gradient predictiveness can be formulated as

$$\triangle_g = \|\nabla\mathcal{L}(\mathbf{X}) - \nabla\mathcal{L}(\mathbf{X} + \eta_g \nabla\mathcal{L}(\mathbf{X}))\|_2, \eta_g \in [a, b], \quad (4)$$

where η_g is step size. Similar to the stability of optimization loss, smaller range of $\triangle_g$ implies that gradient is more insensitive to step size, having better gradient predictiveness. The loss Lipschitzness and gradient predictiveness greatly affect optimization of deep CNNs, i.e., smoothness of landscape and robustness to hyper-parameters.

According to the discussions above, we train the networks with GCP or GAP while comparing their loss Lipschitzness and gradient predictiveness to analyze smoothing effect of GCP. Without loss of generality, we employ the widely used MobileNetV2 [37] and ResNet-18 [16] as backbone models, and train them on large-scale ImageNet [10]. For training MobileNetV2 and ResNet-18 with GCP, following [26], we reduce dimension of the last convolution activations to 256, and train the networks using stochastic gradient descent (SGD) with the same hyper-parameters in [37] and [16]. Besides, we do not use *dropout* operation to avoid randomness for MobileNetV2, and discard down-sampling operation in *conv5_x* for ResNet-18. The left endpoints in the ranges of $\triangle_l$ and $\triangle_g$ are set to the initial learning rates and right endpoints are as large as possible while ensuring stable training of GAP. As such, $\eta_l (\eta_g) \in [0.045, 1.5]$ and $\eta_l (\eta_g) \in [0.1, 75]$ are for MobileNetV2 and ResNet-18, respectively.

The behaviors of MobileNetV2 with 500K steps ($\sim$37 epochs) and ResNet-18 with 100K steps ($\sim$20 epochs) trained with GAP and GCP in terms of loss Lipschitzness and gradient predictiveness are shown in top and bottom of Figure 2, respectively. For both MobileNetV2 and ResNet-18, we observe that the networks with GCP have smaller variations of the optimization loss (i.e., $\triangle_l$) than GAP-based ones, while optimization losses of the networks with GCP are consistently lower than those trained with GAP. These results demonstrate that GCP can improve Lipschitzness of optimization loss while converging faster under the same setting with GAP-based ones. Meanwhile, for changes of the gradient, $\triangle_g$ of the networks with GCP is more stable than GAP-based ones, suggesting the networks with GCP have better gradient predictiveness. In addition, the jumps of both loss landscape and gradient predictiveness for GAP with MobileNetV2 indicate that variations of loss and gradient are considerably large, suggesting varying step sizes are likely to drive the loss uncontrollably higher. In contrast, the variations of one with GCP are fairly small and consistent, suggesting GCP helps for stable training. *In a nutshell, GCP has the ability to smoothen optimization landscape of deep CNNs and improve gradient predictiveness.*

3.3. Connection to Second-order Optimization

Furthermore, we analyze back-propagation (BP) of GCP to explore its effect on optimization of deep CNNs. Let $\mathbf{X}$ be output of the last convolution layer, the gradient of loss $\mathcal{L}$ with respect to $\mathbf{X}$ for GAP layer can be computed as $\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \mathbf{C}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GAP}}}$, where $\mathbf{Z}_{\mathrm{GAP}} = \sum_{i=1}^{N} \mathbf{X}_i$ and $\mathbf{C}$ is a constant matrix. To update weights $\mathbf{W}$ of the last convolution layer, gradient descent is performed as follows:

$$\mathbf{W}_{t+1} \longleftarrow \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}_t, \qquad (5)$$

where $\nabla_{\mathbf{W}} \mathcal{L}_t = \frac{\partial \mathcal{L}}{\partial \mathbf{X}} \frac{\partial \mathbf{X}}{\partial \mathbf{W}_t} = \mathbf{C}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GAP}}} \frac{\partial \mathbf{X}}{\partial \mathbf{W}_t}$, and t indicates t-th iteration.

Let $\mathbf{Z}_{\mathrm{GCP}} = (\mathbf{X}^T \mathbf{J} \mathbf{X})^{\frac{1}{2}}$. The derivative of loss with respect to $\mathbf{X}$ for GCP layer can be written as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = 2\mathbf{J}\mathbf{X} \Big[\mathbf{U} \Big(\Big(\mathbf{K}^T \circ \Big(\mathbf{U}^T 2 \Big(\frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GCP}}} \Big)_{\mathrm{sym}} \mathbf{U} \mathbf{\Lambda}^{\frac{1}{2}} \Big) \Big)$$
$$+ \Big(\frac{1}{2} \mathbf{\Lambda}^{-\frac{1}{2}} \mathbf{U}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GCP}}} \mathbf{U} \Big)_{\mathrm{diag}} \Big) \mathbf{U}^T \Big]_{\mathrm{sym}}, \qquad (6)$$

where $\mathbf{U}$ and $\mathbf{\Lambda}$ are the matrix of eigenvectors and the diagonal matrix of eigenvalues of sample covariance of $\mathbf{X}$, and $\mathbf{K}$ is a mask matrix associated with eigenvalues. Here, $\circ$ denotes matrix Hadamard product; $(\cdot)_{\mathrm{sym}}$ and $(\cdot)_{\mathrm{diag}}$ indicate matrix symmetrization and diagonalization, respectively. More details can refer to [21, 40, 26]. With some assumptions and simplification, Eqn. (6) can be trimmed as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}} \approx 2\mathbf{J}\mathbf{X} \Big(2\mathbf{K}^T \circ \mathbf{\Lambda}^{\frac{1}{2}} + \frac{1}{2} \mathbf{\Lambda}^{-\frac{1}{2}} \Big) \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GCP}}}. \qquad (7)$$

Method	Gradient	Remark
GAP	$\eta \mathbf{C}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GAP}}} \frac{\partial \mathbf{X}}{\partial \mathbf{W}_t}$	$\mathbf{C}$ is a constant matrix
GCP	$\approx \mathbf{F}^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GCP}}} \frac{\partial \mathbf{X}}{\partial \mathbf{W}_t}$	$\mathbf{F}^{-1} = \eta 2\mathbf{J}\mathbf{X}\Big(2\mathbf{K}^T \circ \mathbf{\Lambda}^{\frac{1}{2}} + \frac{1}{2}\mathbf{\Lambda}^{-\frac{1}{2}} \Big)$
K-FAC [35]	$\mathbf{H}^{-1}\mathbf{C}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GAP}}} \frac{\partial \mathbf{X}}{\partial \mathbf{W}_t}$	$\mathbf{H}^{-1} = \eta\big(\frac{\partial \mathcal{L}}{\partial \mathbf{X}}\big)^{-1} \otimes \widehat{\mathbf{X}}^{-1}$

Table 1. Comparison of gradients involved in GAP, GCP and GAP with K-FAC [35]. $\otimes$ indicates Kronecker product.

Details of Eqn. (6) and Eqn. (7) can be found in the supplemental file. By substituting Eqn. (7) into Eqn. (5), we can approximatively update the weights of convolution as

$$\mathbf{W}_{t+1} \longleftarrow \mathbf{W}_t - \mathbf{F}^{-1} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\mathrm{GCP}}} \frac{\partial \mathbf{X}}{\partial \mathbf{W}_t}, \qquad (8)$$

where $\mathbf{F}^{-1} = \eta 2\mathbf{J}\mathbf{X}\Big(2\mathbf{K}^T \circ \mathbf{\Lambda}^{\frac{1}{2}} + \frac{1}{2}\mathbf{\Lambda}^{-\frac{1}{2}} \Big)$.

Previous works [31, 32, 35] show second-order optimization (i.e., $\mathbf{W}_{t+1} \longleftarrow \mathbf{W}_t - \mathbf{H}^{-1}\nabla \mathcal{L}_t$) can speed up training of deep neural networks. However, computation of inverse of Hessian matrix ($\mathbf{H}^{-1}$) is usually very expensive and is sensitive to noise. Therefore, many methods [31, 32, 1] are proposed to approximate $\mathbf{H}^{-1}$. Recently, K-FAC [32, 35] based on an accurate approximation of the Fisher information matrix has proven to be effective in optimizing deep CNNs, which approximates $\mathbf{H}^{-1}$ using Kronecker product between inverse of input of convolution layer (i.e., $\widehat{\mathbf{X}}^{-1}$) and inverse of gradient of the loss with respect to the output (i.e., $\big(\frac{\partial \mathcal{L}}{\partial \mathbf{X}}\big)^{-1}$). The gradients involved in GAP, GCP and GAP with K-FAC are compared in Table 1, where the trimmed BP of GCP (7) shares some similar philosophy with K-FAC. The key difference is that $\mathbf{F}^{-1}$ is computed by the output $\mathbf{X}$ and its eigenvalues, while $\mathbf{H}^{-1}$ is approximated by the input $\widehat{\mathbf{X}}$ and the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$. The experiments in Section 4.1 show that ResNet-50 with GCP uses less training epochs to achieve result matching that of K-FAC [35], which may indicate that BP of GCP is a potential alternative of pre-conditioner for Hessian matrix.

4. Merits Benefited from GCP

In previous section, we explore the effect of GCP on deep CNNs from an optimization perspective. Specifically, we show smoothing effect of GCP on optimization landscape, and discuss the connection between GCP and second-order optimization. Furthermore, these findings can also account for several merits delivered by GCP for training deep CNNs that have not been recognized previously or fully explored, including significant acceleration of network convergence, stronger robustness to distorted examples generated by image corruptions and perturbations, and good generalization ability to different vision tasks. In this
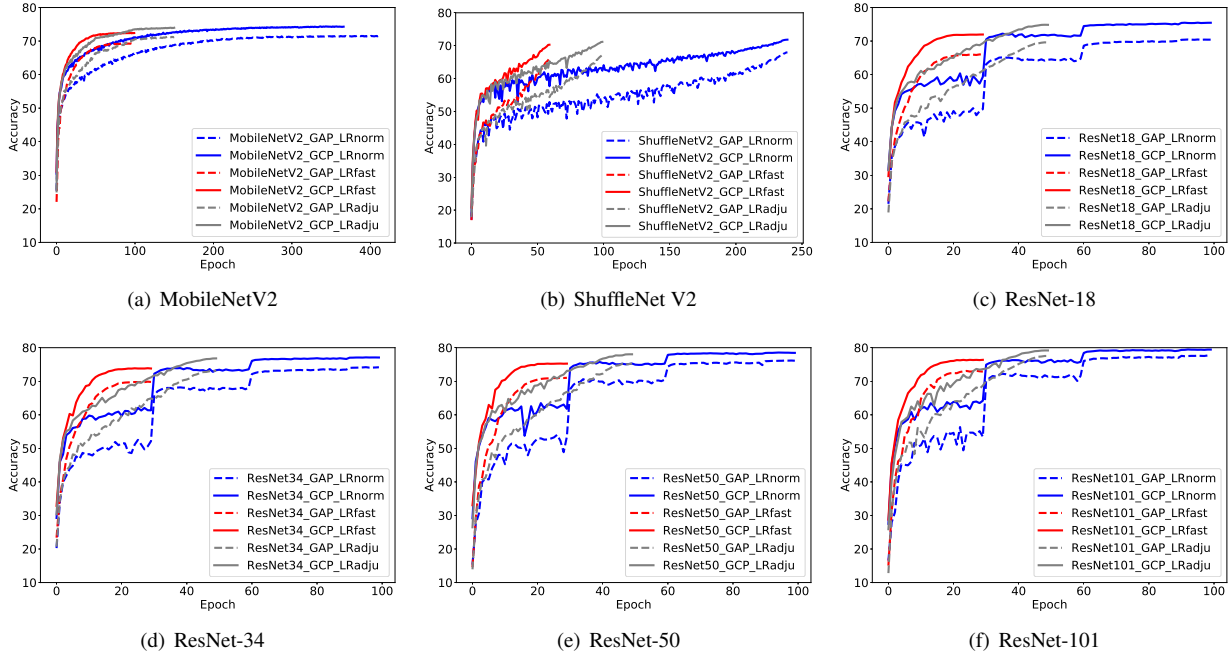
Figure 3. Convergence curves of different deep CNN models trained with GAP and GCP under various settings of learning rates (i.e., LR_{norm}, LR_{fast} and LR_{adju}) on ImageNet.
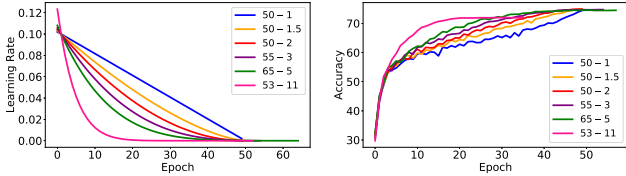

Figure 4. (Left) Curves of various settings ($\rho - e_f$) of lr and (Right) their corresponding convergence curves using ResNet-18 as backbone model on ImageNet. Here e_f and ρ denote the final epoch and power of the polynomial decay in Eqn. (9).

work, all programs are implemented by Pytorch package[1], and run on a workstation with four NVIDIA RTX 2080 Ti GPUs. The detailed descriptions are given as follows.

4.1. Acceleration of Network Convergence

It is well known that training of deep CNNs is a time-consuming process, requiring a mass of computing resources. Therefore, acceleration of network convergence plays a key role in fast training of deep CNNs, especially for large-scale training datasets. Previous study [25] empirically shows the networks trained with GCP converge faster than GAP-based ones, which can be explained by our findings. Specifically, Lipschitzness improvement [38, 34] and connection to second-order optimization [35] brought by GCP can accelerate convergence of networks. Here,

we further show deep CNNs with GCP can support rapid decay of learning rates for significantly improving convergence speed, due to its ability of smoothing optimization landscape. To verify it, we conduct extensive experiments using six representative deep CNN architectures on ImageNet, including MobileNetV2 [37], ShuffleNet V2 [30] and ResNets [16] of 18, 34, 50 and 101 layers.

Specifically, we train all the networks with GAP and GCP by varying the settings of learning rates (lr). Firstly, we use the setting of lr in each original paper, which is indicted by LR_{norm}. As shown in [35], ResNet-50 converges much faster using second-order optimization (K-FAC), in which lr is scheduled by polynomial decay, i.e.,

$$\ell_e = \ell_0 \times \left(1 - \frac{e - e_s}{e_f - e_s}\right)^{\rho}, \qquad (9)$$

where ℓ_0 is the initial lr; e, e_s and e_f indicate e-th, the start and the final epochs, respectively. The parameter ρ controls decay rate. Inspired by [35], we employ the setting of lr in Eqn. (9) for fast training of ResNets. To determine parameter ρ, we set $\ell_0 = 0.1$ and train ResNet-18 with GCP on ImageNet using various ρ while controlling e_f be less than 65, which is consistent with [35]. Figure 4 illustrates the curve of each lr and its corresponding convergence. Clearly, larger ρ leads to faster convergence but lower accuracy. The setting with ($\rho = 11, e_f = 53$) has the fastest convergence, denoted by LR_{fast} hereafter, which converges within 30 epochs. Among them, the setting with ($\rho = 2, e_f = 50$) makes the best trade-off be-

Backbone Model	Pooling Method	lr	BS	Training Epochs	Matching Epoch	Top-1 Accuracy	Top-5 Accuracy
MobileNetV2	GAP	LR_{norm}	96	400	N/A	71.58	90.30
	GCP	LR_{adju}	192	150	$68_{(\downarrow 332)}$	$73.97_{(\uparrow 2.39)}$	$91.54_{(\uparrow 1.24)}$
ShuffleNet V2	GAP	LR_{norm}	1,024	240	N/A	67.96	87.84
	GCP	LR_{adju}	1,024	100	$78_{(\downarrow 162)}$	$71.17_{(\uparrow 3.21)}$	$89.74_{(\uparrow 1.90)}$
ResNet-18	GAP	LR_{norm}	256	100	N/A	70.47	89.62
	GCP	LR_{adju}	256	50	$32_{(\downarrow 68)}$	$74.86_{(\uparrow 4.39)}$	$91.81_{(\uparrow 2.19)}$
ResNet-34	GAP	LR_{norm}	256	100	N/A	74.19	91.61
	GCP	LR_{adju}	256	50	$38_{(\downarrow 62)}$	$76.81_{(\uparrow 2.62)}$	$93.09_{(\uparrow 1.48)}$
ResNet-50	GAP	LR_{norm}	256	100	N/A	76.17	92.93
	GCP	LR_{adju}	256	50	$40_{(\downarrow 60)}$	$78.03_{(\uparrow 1.86)}$	$93.95_{(\uparrow 1.02)}$
	GAP	K-FAC [35]*	4,096	35	N/A	$75.10_{(\downarrow 1.07)}$	N/A
	GCP	LR_{fast}	256	30	$20_{(\downarrow 15)}$	$75.31_{(\downarrow 0.86)}$	$92.15_{(\downarrow 0.78)}$
ResNet-101	GAP	LR_{norm}	256	100	N/A	77.67	93.89
	GCP	LR_{adju}	256	50	$41_{(\downarrow 59)}$	$79.18_{(\uparrow 1.51)}$	$94.51_{(\uparrow 0.62)}$

Table 2. Comparison of different CNNs trained with GAP using LR_{norm} and those trained with GCP using LR_{adju} on ImageNet. 'Matching Epoch' indicates that at which the networks with GCP achieve comparable performance with the corresponding networks with GAP undergoing full training epochs. *: The result of K-FAC is duplicated from the original paper [35].

Method		Setting of lr
Backbone	lr	
MobileNet V2	LR_{norm}	0.045×0.98^e
	LR_{fast}	0.06×0.92^e
	LR_{adju}	$\begin{cases} linear(6e^{-2}, 1e^{-3}, 0), \\ linear(1e^{-2}, 1e^{-4}, 50), \\ linear(1e^{-3}, 1e^{-5}, 100) \end{cases}$
ShuffleNetV2	LR_{norm}	
	LR_{fast}	$0.5 \times (1 - \frac{step}{t\_step})$
	LR_{adju}	
ResNet	LR_{norm}	$0.1^{(e//30)+1}$
	LR_{fast}	$0.1 \times (1 - \frac{e-1}{52})^{11}$
	LR_{adju}	$0.1 \times (1 - \frac{e-1}{49})^{2}$

Table 3. Detailed settings of lr for various CNNs.

Method	IMAGENET-C		IMAGENET-P	
	mCE	Relative mCE	mFP	mT5D
MobileNetV2 + GAP	87.1	114.9	79.8	96.5
MobileNetV2 + GCP	$81.7_{(\downarrow 5.4)}$	$110.6_{(\downarrow 4.3)}$	$64.3_{(\downarrow 15.5)}$	$87.6_{(\downarrow 8.9)}$
ShuffleNet V2 + GAP	92.7	126.7	94.7	108.2
ShuffleNet V2 + GCP	$85.2_{(\downarrow 7.5)}$	$112.6_{(\downarrow 14.1)}$	$75.2_{(\downarrow 19.5)}$	$95.5_{(\downarrow 12.7)}$
ResNet-18 + GAP	84.7	103.9	72.8	87.0
ResNet-18 + GCP	$76.3_{(\downarrow 8.4)}$	$101.3_{(\downarrow 2.6)}$	$53.2_{(\downarrow 19.6)}$	$77.1_{(\downarrow 9.9)}$
ResNet-34 + GAP	77.9	98.7	61.7	79.5
ResNet-34 + GCP	$72.4_{(\downarrow 5.5)}$	$96.9_{(\downarrow 1.8)}$	$47.7_{(\downarrow 14.0)}$	$72.4_{(\downarrow 7.1)}$
ResNet-50 + GAP	76.7	105.0	58.0	78.3
ResNet-50 + GCP	$70.7_{(\downarrow 6.0)}$	$97.9_{(\downarrow 7.1)}$	$47.5_{(\downarrow 10.5)}$	$74.6_{(\downarrow 3.7)}$
ResNet-101 + GAP	70.3	93.7	52.6	73.9
ResNet-101 + GCP	$65.5_{(\downarrow 4.8)}$	$89.1_{(\downarrow 4.6)}$	$42.1_{(\downarrow 10.5)}$	$68.3_{(\downarrow 5.6)}$

Table 4. Comparison of GAP and GCP on IMAGENET-C and IMAGENET-P.

Backbone	lr	D	Top-1 Acc.	Top-5 Acc.
MobiNetV2	LR_{norm}	256	74.36	91.90
		128	73.28 ($\downarrow$ 1.08)	91.30 ($\downarrow$ 0.60)
	LR_{adju}	256	73.97	91.54
		128	72.58 ($\downarrow$ 1.39)	90.90 ($\downarrow$ 0.64)
ResNet-50	LR_{norm}	256	78.56	94.16
		128	78.21 ($\downarrow$ 0.35)	93.93 ($\downarrow$ 0.23)
	LR_{adju}	256	78.03	93.95
		128	77.64 ($\downarrow$ 0.39)	93.67 ($\downarrow$ 0.28)

Table 5. Results of MobileNetV2 and ResNet-50 trained with GCP using different lr and dimensions (D) of input on ImageNet.

tween convergence speed and classification accuracy, denoted by LR_{adju}.

For MobileNetV2, the original lr is scheduled by exponential decay (i.e., 0.045×0.98^e), and we set LR_{fast} by decreasing base number while increasing the initial lr, i.e., 0.06×0.92^e. LR_{adju} is scheduled by a stage-wise linear decay, i.e., $linear(l_s, l_e, n) = l_s - \frac{l_s - l_e}{50}(e - n)$ for e-th epoch where l_s, l_e and n are initial lr, final lr and start epoch in each stage. The lr of ShuffleNet V2 in the original paper is scheduled by a step-wise linear decay, and number of total steps (i.e., t_step) is 3e5 ($\sim$ 240 epochs). For ShuffleNet V2, LR_{fast} and LR_{adju} are set by reducing training epochs to 60 and 100, respectively. The detailed settings of lr are summarized in Table 3.

The convergence curves of different networks trained with GAP and GCP under various settings of lr are illustrated in Figure 3, from which we have the following observations. (1) Comparing with the networks trained under LR_{norm}, those trained under the settings of LR_{fast} and LR_{adju} suffer from performance degradation. However, performance degradations of the networks with GCP are

less than those based on GAP, especially under LR_{fast}. (2) The networks trained with GCP using LR_{fast} achieve better or matching results using only about 1/4 of the number of epochs for training networks with GAP under LR_{norm}.

Moreover, we compare the networks trained with GAP under LR_{norm} (i.e., the original settings) and those trained with GCP using LR_{adju}. According to the results in Table 2, we make a summary as follows. (1) Comparing to the networks trained with GAP under LR_{norm}, those trained with GCP under LR_{adju} achieve higher accuracies while

using less training epochs. (2) The networks with GCP obtain matching or comparable accuracies to GAP-based ones using much less training epochs, especially for lightweight CNN models, i.e., MobileNetV2 and ShuffleNet V2. For example, MobileNetV2 with GCP achieves matching accuracies to GAP-based one using only 68 epochs, while the latter one needs about 400 epochs. (3) Comparing to the second-order optimization method K-FAC [35], GCP obtains moderate accuracy gain using less training epochs, while achieving comparable accuracies with K-FAC using only 20 epochs. Furthermore, GCP is easier to implement. *The extensive experiments above strongly support our finding: GCP can significantly speed up convergence of deep CNNs with rapid decay of learning rates.*

Additionally, we assess the effect of dimension of covariance representations (COV-Reps) on behavior of convergence using MobileNetV2 (MobiNetV2) and ResNet-50 on ImageNet. If the dimension of input features is D, GCP will output a $D(D + 1)/2$-dimensional COV-Reps. Here, we set D to 256 (the default setting) and 128, and train the networks under the settings of LR_{norm} and LR_{adju}, respectively. The results are given in Table 5, from which one can see that lower-dimensional COV-Reps still allow faster convergence of deep CNNs, but suffer from larger performance degradation in the case of faster convergence (i.e., LR_{adju}). This indicates that dimension of COV-Reps has a nontrivial effect on the behavior of convergence. Therefore, how to compress COV-Reps while preserving merits of high-dimensional ones is an important issue. Albeit many works are proposed to compress COV-Reps [13, 22, 41], they still have not been verified in large-scale scenarios. A potential solution is to learn compact COV-Reps from high-dimensional ones based on knowledge distillation [18], which will be studied in future.

4.2. Robustness to Distorted Examples

Improvement of loss Lipschitzness and gradient predictiveness brought by GCP make the networks more robust to inputs perturbed by distortions. We also note similar conclusion is stated in [7]. To verify this point, we conduct experiments on recently introduced IMAGENET-C and IMAGENET-P benchmarks [17]. Different from the works [4, 33] that study effect of adversarial distortions as a type of worst-case analysis for network robustness, these two benchmarks are designed to evaluate robustness of deep CNNs to common image corruptions and perturbations, which have a connection with adversarial distortions and play a key role in safety-critical applications.

The IMAGENET-C benchmark performs fifty types of corruptions (e.g., noise, blur, weather and digital) on validation set of ImageNet, and each type of corruption has five levels of severity. The IMAGENET-P benchmark generates a series of perturbation sequences on validation set

of ImageNet by performing more than ten types of perturbations, such as motion and zoom blur, brightness, translation, rotation, scale and tilt perturbations. Following the standard protocol in [17], we train all CNN models on training set (clean images) of ImageNet, and report the results on IMAGENET-C and IMAGENET-P benchmarks. The evaluation metrics include mean Corruption Error (mCE) and Relative mean Corruption Errors (Relative mCE) for IMAGENET-C, mean Flip Rate (mFR) and mean Top-5 Distance (mT5D) for IMAGENET-P. For details of the metrics one can refer to [17]. *Note that lower values indicate better performance for all evaluation metrics.*

For a fair comparison, we employ evaluation code released by the authors. Note that AlexNet [24] is a baseline model, which obtains value of 100 for all evaluation metrics. The results of different deep CNNs with GAP and GCP are given in Table 4, from which we can see that the networks with GCP significantly outperform GAP-based ones, suggesting that GCP can greatly improve the robustness of deep CNNs to common image corruptions and perturbations. Note that VGG-VD19 [39] and VGG-VD19 with BN achieve 88.9, 122.9, 66.9, 78.6 and 81.6, 111.1, 65.1, 80.5 in terms of mCE, Relative mCE, mFR, mT5D, respectively. Albeit BN improves the Lipschitzness, it is not robust to perturbations. In contrast, GCP is robust to both corruptions and perturbations. In [17], many schemes are suggested to improve the robustness to corruptions and perturbations, and our work shows that GCP is a novel and promising solution. Moreover, it is potential to combine GCP with other schemes for further improvement.

4.3. Generalization Ability to Other Tasks

Since GCP usually allows deep CNNs converge to better local minima, the networks with GCP pretrained on large-scale dataset can provide a better initialization model to other vision tasks. That is, they may have good generalization ability. To verify this, we first train the networks with GCP on ImageNet, and then directly apply them to object detection and instance segmentation tasks. Specifically, using ResNet-50 and ResNet-101 as backbone models, we compare performance of the networks trained with GAP and GCP on MS COCO [27] using Faster R-CNN [36] and Mask R-CNN [15] as basic detectors. For training networks with GCP, Li et al. [26] suggest no down-sampling in *conv5_1*. This increases resolution of the last feature maps while resulting in larger computational cost, especially for object detection and instance segmentation where large-size input images are required. To handle this issue, we introduce two strategies: (1) we still use down-sampling as done in the original ResNet, and the method is indicated by GCP_D; (2) a max-pooling layer with a step size 2 is inserted before *conv5_1* for down-sampling, indicted by GCP_M.

For a fair comparison, all methods are implemented by

Backbone Model	Method	Detectors	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ResNet-50	GAP	Faster R-CNN	36.4	58.2	39.2	21.8	40.0	46.2
	GCP_D		36.6$_{(\uparrow 0.2)}$	58.4$_{(\uparrow 0.2)}$	39.5$_{(\uparrow 0.3)}$	21.3$_{(\downarrow 0.5)}$	40.8$_{(\uparrow 0.8)}$	47.0$_{(\uparrow 0.8)}$
	GCP_M		**37.1**$_{(\uparrow 0.7)}$	**59.1**$_{(\uparrow 0.9)}$	**39.9**$_{(\uparrow 0.7)}$	**22.0**$_{(\uparrow 0.2)}$	**40.9**$_{(\uparrow 0.9)}$	**47.6**$_{(\uparrow 1.4)}$
ResNet-101	GAP		38.7	60.6	41.9	22.7	43.2	50.4
	GCP_D		39.5$_{(\uparrow 0.8)}$	60.7$_{(\uparrow 0.1)}$	43.1$_{(\uparrow 1.2)}$	22.9$_{(\uparrow 0.2)}$	**44.1**$_{(\uparrow 0.9)}$	**51.4**$_{(\uparrow 1.0)}$
	GCP_M		**39.6**$_{(\uparrow 0.9)}$	**61.2**$_{(\uparrow 0.6)}$	**43.1**$_{(\uparrow 1.2)}$	**23.3**$_{(\uparrow 0.6)}$	43.9$_{(\uparrow 0.7)}$	51.3$_{(\uparrow 0.9)}$
ResNet-50	GAP	Mask R-CNN	37.2	58.9	40.3	22.2	40.7	48.0
	GCP_D		37.3$_{(\uparrow 0.1)}$	58.8$_{(\downarrow 0.1)}$	40.4$_{(\uparrow 0.1)}$	22.0$_{(\downarrow 0.2)}$	41.1$_{(\uparrow 0.4)}$	48.2$_{(\uparrow 0.2)}$
	GCP_M		**37.9**$_{(\uparrow 0.7)}$	**59.4**$_{(\uparrow 0.5)}$	**41.3**$_{(\uparrow 1.0)}$	**22.4**$_{(\uparrow 0.2)}$	**41.5**$_{(\uparrow 0.8)}$	**49.0**$_{(\uparrow 1.0)}$
ResNet-101	GAP		39.4	60.9	43.3	23.0	43.7	51.4
	GCP_D		40.3$_{(\uparrow 0.9)}$	61.5$_{(\uparrow 0.6)}$	44.0$_{(\uparrow 0.7)}$	**24.1**$_{(\uparrow 1.1)}$	44.7$_{(\uparrow 1.0)}$	52.5$_{(\uparrow 1.1)}$
	GCP_M		**40.7**$_{(\uparrow 1.3)}$	**62.0**$_{(\uparrow 1.1)}$	**44.6**$_{(\uparrow 1.3)}$	23.9$_{(\uparrow 0.9)}$	**45.2**$_{(\uparrow 1.5)}$	**52.9**$_{(\uparrow 1.5)}$

Table 6. Object detection results of various deep CNN models using Faster R-CNN and Mask R-CNN on COCO val2017.

Method	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
R-50 + GAP	34.1	55.5	36.2	16.1	36.7	50.0
R-50 + GCP_D	34.2	55.3	36.4	15.8	37.1	50.1
R-50 + GCP_M	**34.7**	**56.3**	**36.8**	**16.4**	**37.5**	**50.6**
R-101 + GAP	35.9	57.7	38.4	16.8	39.1	53.6
R-101 + GCP_D	36.5	58.2	38.9	17.3	39.9	53.5
R-101 + GCP_M	**36.7**	**58.7**	**39.1**	**17.6**	**39.9**	**53.7**

Table 7. Instance segmentation results of various deep CNN models using Mask R-CNN on COCO val2017.

Backbone	Method	Top-1	Top-5	AP	AP_{50}	AP_{75}
ResNet-18	w/o DS	**75.47**	**92.23**	30.0	50.7	31.4
	w/ DS	74.48	91.68	**30.3**	**51.0**	**32.2**
ResNet-50	w/o DS	**78.56**	**94.16**	36.6	58.4	39.5
	w/ DS	78.10	94.09	**36.8**	**58.5**	**39.7**
ResNet-101	w/o DS	**79.48**	**94.75**	39.5	60.7	43.1
	w/ DS	79.11	94.56	**39.6**	**60.9**	**43.4**

Table 8. Comparison of GCP with or without down-sampling (DS) on ImageNet (columns 3 and 4) and MS COCO (columns 5, 6, and 7). Here, Faster R-CNN is used for object detection.

MMDetection toolkit [6] with the same (default) settings. Specifically, the shorter side of input images are resized to 800, and SGD is used to optimize the networks with a weight decay of 1e-4, a momentum of 0.9 and a mini-batch size of 8. All detectors are trained within 12 epochs on train2017 of COCO, where the learning rate is initialized to 0.01 and is decreased by a factor of 10 after 8 and 11 epochs, respectively. The results on val2017 are reported for comparison. As listed in Table 6, GCP_M outperforms GCP_D while both of them are superior to GAP. Specifically, for ResNet-50, GCP_M improves GAP by 0.7% in terms of AP using Faster R-CNN and Mask R-CNN as detectors. For ResNet-101, GCP_M outperforms GAP by 0.9% and 1.3% for Faster R-CNN and Mask R-CNN, respectively. For results of instance segmentation in Table 7, GCP_M improves GAP by 0.6% and 0.8% using ResNet-50 (R-50) and ResNet-101 (R-101) as backbone models, respectively. Note that GCP brings more improvement when backbone model and detector are stronger. These results show the pre-trained networks with GCP on large-scale datasets can well generalize to different vision tasks, indicating the networks with GCP can provide better initialization models.

As described above, integration of GCP into ResNets discards down-sampling (DS) operation in *conv5_1* to obtain more sampling features for more promising classification performance. However, it decreases the resolution of *conv5_x* and increases computing cost, especially for large-size input images. Here, we assess its effect on performance of GCP. Specifically, we employ ResNet-18, ResNet-50 and ResNet-101 as backbone models, and compare the results of GCP with or without DS (i.e., $conv5\_1$ with a stride of 2). For object detection, we use Faster R-CNN as the basic detector. As shown in Table 8, GCP with DS is inferior to one without DS on ImageNet classification for each model, but achieves better performance on object detection. These results suggest that DS can be introduced for balancing classification accuracy, performance of object detection and model complexity for the networks trained with GCP.

5. Conclusion

In this paper, we made an attempt to analyze the effectiveness of GCP on deep CNNs from an optimization perspective. Specifically, we showed that GCP has the ability to improve Lipschitzness of loss and predictiveness of gradient in context of deep CNNs, and discussed the connection between GCP and second-order optimization. Our findings can account for several merits of GCP for training deep CNNs that have not been recognized previously or fully explored, including significant acceleration of network convergence, stronger robustness to distorted examples, and good generalization ability to different vision tasks. The extensive experimental results provide strong support to our findings. Our work provides an inspiring view to understand GCP and may help researchers explore more merits of GCP in context of deep CNNs. In future, we will investigate the theoretical proofs on smoothing effect of GCP and the rigorous connection to second-order optimization.

References

[1] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *J. Mach. Learn. Res.*, 18:116:1–116:40, 2017.

[2] Johan Bjorck, Carla P. Gomes, Bart Selman, and Kilian Q. Weinberger. Understanding batch normalization. In *NeurIPS*, 2018.

[3] Sijia Cai, Wangmeng Zuo, and Lei Zhang. Higher-order integration of hierarchical convolutional activations for fine-grained visual categorization. In *ICCV*, 2017.

[4] Nicholas Carlini, Guy Katz, Clark W. Barrett, and David L. Dill. Ground-truth adversarial examples. *arXiv*, 1709.10207, 2017.

[5] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Free-form region description with second-order pooling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(6):1177–1189, 2015.

[6] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv:1906.07155*, 2019.

[7] Moustapha Cissé, Piotr Bojanowski, Edouard Grave, Yann N. Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *ICML*, 2017.

[8] Yin Cui, Feng Zhou, Jiang Wang, Xiao Liu, Yuanqing Lin, and Serge Belongie. Kernel pooling for convolutional neural networks. In *CVPR*, 2017.

[9] Xiyang Dai, Joe Yue-Hei Ng, and Larry S. Davis. FASON: First and second order information fusion network for texture recognition. In *CVPR*, 2017.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[11] Ali Diba, Vivek Sharma, and Luc Van Gool. Deep temporal linear encoding networks. In *CVPR*, 2017.

[12] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *EMNLP*, 2016.

[13] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. In *CVPR*, 2016.

[14] Mengran Gou, Fei Xiong, Octavia Camps, and Mario Sznaier. Monet: Moments embedding network. In *CVPR*, 2018.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[17] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

[18] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[20] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015.

[21] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv*, 1509.07838, 2015.

[22] Shu Kong and Charless Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *CVPR*, 2017.

[23] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[25] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *CVPR*, 2018.

[26] Peihua Li, Jiangtao Xie, Qilong Wang, and Wangmeng Zuo. Is second-order information helpful for large-scale visual recognition? In *ICCV*, 2017.

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollr, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[28] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with CNNs. In *BMVC*, 2017.

[29] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear CNN models for fine-grained visual recognition. In *ICCV*, 2015.

[30] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

[31] James Martens. Deep learning via hessian-free optimization. In *ICML*, 2010.

[32] James Martens and Roger B. Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, 2015.

[33] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *ICLR*, 2017.

[34] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

[35] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks. In *CVPR*, 2019.

[36] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017.

[37] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[38] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *NeurIPS*, 2018.

[39] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[40] Qilong Wang, Peihua Li, and Lei Zhang. G^{2}DeNet: Global Gaussian distribution embedding network and its application to visual recognition. In *CVPR*, 2017.

[41] Kaicheng Yu and Mathieu Salzmann. Statistically-motivated second-order pooling. In *ECCV*, 2018.

[42] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[43] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

[44] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(9):2131–2145, 2019.