DropCov: A Simple yet Effective Method for Improving Deep Architectures

Qilong Wang[1,3], **Mingze Gao**[1], **Zhaolin Zhang**[1], **Jiangtao Xie**[2], **Peihua Li**[2], **Qinghua Hu**[1,3,*]

[1]Tianjin University, China, [2]Dalian University of Technology, China,
[3] Haihe Laboratory of Information Technology Application Innovation, Tianjin, China
qlwang@tju.edu.cn, gaomingze@tju.edu.cn, zzl9@tju.edu.cn
jiangtaoxie@mail.dlut.edu.cn, peihuali@dlut.edu.cn, huqinghua@tju.edu.cn

Abstract

Previous works show global covariance pooling (GCP) has great potential to improve deep architectures especially on visual recognition tasks, where post-normalization of GCP plays a very important role in final performance. Although several post-normalization strategies have been studied, these methods pay more close attention to effect of normalization on covariance representations rather than the whole GCP networks, and their effectiveness requires further understanding. Meanwhile, existing effective post-normalization strategies (e.g., matrix power normalization) usually suffer from high computational complexity (e.g., $O(d^3)$ for d-dimensional inputs). To handle above issues, this work first analyzes the effect of post-normalization from the perspective of training GCP networks. Particularly, we for the first time show that *effective post-normalization can make a good trade-off between representation decorrelation and information preservation for GCP, which are crucial to alleviate over-fitting and increase representation ability of deep GCP networks, respectively*. Based on this finding, we can improve existing post-normalization methods with some small modifications, providing further support to our observation. Furthermore, this finding encourages us to propose a novel pre-normalization method for GCP (namely DropCov), which develops an adaptive channel dropout on features right before GCP, aiming to reach trade-off between representation decorrelation and information preservation in a more efficient way. Our DropCov only has a linear complexity of $O(d)$, while being free for inference. Extensive experiments on various benchmarks (i.e., ImageNet-1K, ImageNet-C, ImageNet-A, Stylized-ImageNet, and iNat2017) show our DropCov is superior to the counterparts in terms of efficiency and effectiveness, and provides a simple yet effective method to improve performance of deep architectures involving both deep convolutional neural networks (CNNs) and vision transformers (ViTs).

1 Introduction

Global covariance pooling (GCP) has shown remarkable potential to improve performance of deep architectures in a variety of tasks, especially visual recognition [31, 23, 29, 30, 8, 48, 36, 47, 51]. One of core differences among various deep GCP methods is post-normalization for covariance representations, which plays a crucial role in final performance. The existing post-normalization approaches can be roughly categorized into element-wise post-normalization methods [31, 25, 2] and structure-wise ones [23, 29, 28, 25, 36, 47], which concentrate on normalizing entries and

[*] Qinghua Hu is the corresponding author and is with Engineering Research Center of City intelligence and Digital Governance, Ministry of Education of the People's Republic of China.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

eigenvalues of covariance matrices, respectively. Some researches (e.g., [30]) perform two kinds of normalizations simultaneously. Although several post-normalization strategies have been studied, behavior of different approaches seem vary significantly and there is a lack of an intuitive and unified interpretation. Therefore, the true mechanism lying behind these post-normalization methods (especially under deep architectures) is still an open problem.

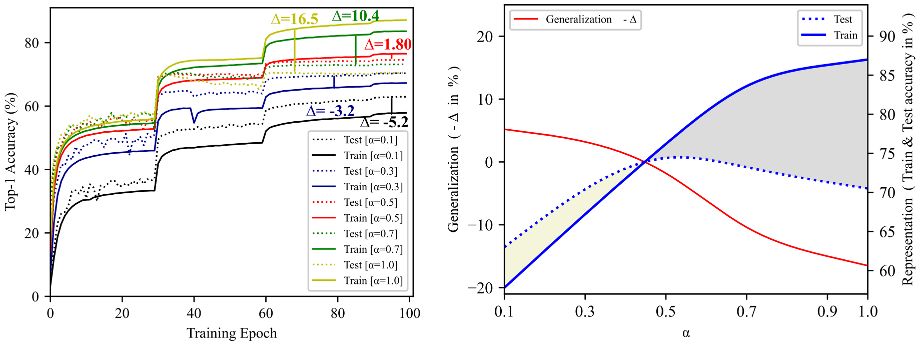
Existing works understand the effect of post-normalization methods of GCP from different aspects. Specifically, [31, 25] claim that their element-wise normalization strategies can suppress the burstiness of covariance representations as in the classical bag-of-words [37] or actually perform a kind of whitening on representations. DeepO$_2$P [23] and MPN-COV [29] develop post-normalization methods to exploit geometry of covariances by considering the fact that space of covariance forms a Riemannian manifold [1, 34]. Besides, MPN-COV shows that matrix square-root (sqrt) normalization accounts for a robust covariance estimation [45], while [30] claims spectral normalization compensates for burstiness of eigenvalues and shows effectiveness of matrix sqrt normalization via empirical comparisons. Although many efforts are made, the aforementioned researches try to explain why GCP works by studying how post-normalizations affect covariance representations, which lack a full view on the whole GCP networks. Therefore, this work analyzes effect of post-normalization approaches on optimizing deep GCP networks. Particularly, by taking matrix power normalization [29] as an example, we find that post-normalization tries to handle a paradox between representation decorrelation and information preservation for GCP, which are crucial to alleviate over-fitting and increase representation ability of deep GCP networks, respectively. Particularly, effective post-normalization approaches can achieve a good trade-off, and so lead to a better performance. Based on this finding, we introduce some modified versions of existing post-normalization methods (e.g., adaptive power normalization for MPN-COV, I-LogM for DeepO$_2$P, and linear transformation for B-CNN [31]), which achieve further improvement, providing support to our observation.

Another issue of existing post-normalization methods is high computational complexity. Particularly, effective structure-wise post-normalization strategies usually involve singular value decomposition (SVD) of covariance matrices or sequential matrix multiplications, resulting in the computational complexity of $O(d^3)$ for d-dimensional inputs. Inspired by finding above, this paper proposes an efficient pre-normalization method for GCP (namely DropCov) based on dropout [26], as it has the natural ability to perform feature decorrelation [4] and reduce over-fitting of training neural networks [26]. Specifically, our DropCov develops an adaptive channel dropout on features right before computation of covariance representations, where probability of dropout is adaptively decided by seeking a good trade-off between representation decorrelation and information preservation. Compared to existing post-normalization methods, our DropCov achieves superior or competitive performance with a linear computational complexity of $O(d)$, while it is only used for training.

To verify the effectiveness of our proposed DropCov, we conduct experiments using various deep architectures including CNNs [17] and vision transformers (ViT) [43, 32, 50] on ImageNet-1K [7], ImageNet-C [18], ImageNet-A [19], Stylized-ImageNet [13], and long-tailed iNat2017 [20]. The contributions of this work are summarized as follows: (1) To our best knowledge, this work for the first time analyzes the effect of post-normalization on GCP from the perspective of optimizing neural networks, and we show that effective post-normalization methods can make a good trade-off between representation decorrelation (*w.r.t* over-fitting reduction) and information preservation (*w.r.t* increase of representation ability) for GCP (*w.r.t* deep GCP networks); (2) Based on our finding, we make some small modifications to existing post-normalization methods (Sec. 2.2), e.g., adaptive power normalization and I-LogM & linear transformation. The comparisons in Sec. 5.2 show our modifications can bring further improvement, giving support on our observation; (3) Profiting from our finding, we propose a novel DropCov method by developing an adaptive channel dropout as pre-normalization of GCP, which has a linear complexity of $O(d)$ and is only used for training. Extensive experiments show our DropCov is superior or competitive to existing post-normalization approaches and provides a simple yet effective method to improve deep architectures.

2 Effect of Post-normalization on Deep GCP Networks

In this section, we first analyze the effect of post-normalization on optimizing deep GCP networks by taking matrix power normalization as an example. To further verify our finding, we spread it to existing post-normalization approaches, and make some modifications according to our observations.



(a) Convergence curves of various α (b) Generalization *vs.* representation ability

Figure 1: Results of GCP networks with matrix power normalization (MPN) using the backbone of ResNet-18 ($d = 128$) on ImageNet-1K. (a) Convergence curves of various $\alpha \in (0, 1]$; (b) Comparison of generalization and representation abilities for various α. From them we can see that larger values of α result in higher training accuracies indicated by blue solid line in (b), *w.r.t* better representation or approximation ability, but suffer from heavier over-fitting (*w.r.t* worse generalization), aka larger differences of training accuracy over test one indicated by Δ. In contrast, smaller values of α have better generalization (aka smaller Δ), while obtaining lower classification accuracies (*w.r.t* weaker representation ability). Particularly, $\alpha = 0.5$ achieves the best trade-off and so leads the best results.

2.1 How Does Matrix Power Normalization Impact GCP

Matrix Power Normalization Both previous works [47, 36] and our experiments show structure-wise post-normalization methods are generally superior to element-wise ones, where matrix power normalization (MPN) usually performs the most competitively. Let $\mathbf{X} \in \mathbb{R}^{N \times d}$ be N d-dimensional features with zero-mean those outputted by the last convolution layer of deep CNNs or the last transformer block of ViT (i.e., final word tokens), GCP computes $\mathbf{X}^T\mathbf{X}$ with post-normalization to generate a covariance representation $\mathbf{Z}$. Specifically, GCP with MPN is calculated as

$$\mathbf{Z} = (\mathbf{X}^T\mathbf{X})^{\alpha} = \mathbf{U}\mathbf{\Lambda}^{\alpha}\mathbf{U}^T, \tag{1}$$

where $\mathbf{Z} \in \mathbb{R}^{d \times d}$ is used for final classification. $\alpha > 0$ is a parameter of power. $\mathbf{U}$ and $\mathbf{\Lambda}$ are the matrix of eigenvectors and the diagonal matrix of eigenvalues of $\mathbf{X}^T\mathbf{X}$, respectively.

Effect of Power For MPN, power α has a significant effect on the final results. Previous works [29, 30] conduct empirical comparison of α, and choose the best one (e.g., $\alpha = 0.5$). However, the effect brought by α is still not very clear. In this work, we first analyze the effect of $\alpha \in (0, 1]$ on optimizing deep GCP networks. It is clear from Eqn. (1) that α has the following effect on GCP from the mathematical perspective:

$$\begin{cases} \mathbf{Z} \longmapsto \mathbf{I}, \quad \text{if} \quad \alpha \longmapsto 0 \quad : \text{Representation Decorrelation \& Information Loss}, \\ \mathbf{Z} \longmapsto \mathbf{X}^T\mathbf{X}, \quad \text{if} \quad \alpha \longmapsto 1 \quad : \text{Information Preservation \& Strong Correlation}. \end{cases} \tag{2}$$

Specifically, when $\alpha \longmapsto 0$, the normalized representation $\mathbf{Z}$ will be decorrelated and tend to be an identity matrix $\mathbf{I}$. Such representation decorrelation can help optimization of neural networks [27, 22, 4] (in particular for combating over-fitting [5]), which, however, will lead to information loss for $\mathbf{X}^T\mathbf{X}$, hurting representation ability of covariances. Particularly, $\mathbf{Z}$ with MPN of $\alpha = 0$ becomes an identity matrix, and the information lying in $\mathbf{X}^T\mathbf{X}$ completely loses. When $\alpha \longmapsto 1$, information of $\mathbf{X}^T\mathbf{X}$ will be gradually preserved, maintaining the correlations as characterized in the covariance matrices, which makes training of neural networks difficult (e.g., over-fitting). To further verify above observation, we conduct experiments using ResNet-18 on ImageNet-1K (more results can refer to supplementary materials). By observing Figs. 1(a) and 1(b), we draw the following conclusion:

- When $\alpha < 0.5$, post-normalization (i.e., MPN) tends to decorrelate representation. According to red line of Fig. 1(b), smaller values of α reduce over-fitting more significantly and have better generalization (aka smaller differences of training accuracy over test one [3, 5]). Meantime, indicated by blue lines of Fig. 1(b), smaller α leads lower accuracies (*w.r.t* weaker representation or approximation ability [35, 47]) due to information loss.

- When $\alpha > 0.5$, post-normalization (i.e., MPN) tends to preserve information. The training accuracies (*w.r.t* representation or approximation ability) of deep GCP networks indicated by blue solid line of Fig. 1(b) increase along values of α enlarge, but heavier over-fitting (aka larger values of Δ) occur because effect of representation decorrelation decreases.

According to Eqn. (2) and above observation, we have

Corollary 1. *Effective post-normalization (e.g., matrix power normalization (MPN) with* $0 < \alpha \leq 1$*) can achieve a good trade-off between representation decorrelation and information preservation for GCP, which are crucial to alleviate over-fitting and increase representation ability of deep GCP networks, respectively. Particularly, MPN with* $\alpha = 0.5$ *achieves the best trade-off for* $\alpha \in (0, 1]$ *(without considering other factors), which is proved to be the widely used choice of* α *[29, 30].*

The Corollary 1 illuminates the effect of MPN on deep GCP networks from the perspective of model training, and gives an intuitive explanation about the choice of parameter α.

2.2 Extension of Corollary 1 to Existing Post-normalization Methods

To further verify Corollary 1, we extend it to analyze behavior of existing post-normalization. Accompanied by these analyses, we can improve previous methods with some small modifications.

Adaptive Power Normalization (APN) Above all, Corollary 1 shows MPN with $\alpha = 0.5$ is the best trade-off for $\alpha \in (0, 1]$ if no other factor is considered. To further verify the effect of trade-off, we propose an adaptive power normalization (APN) by considering the effect of inputs (i.e., eigenvalues $\mathbf{\Lambda}$ of covariances). Specifically, we formulate the trade-off between representation decorrelation and information preservation as the following objective function:

$$\min_{\alpha} \log\left(\frac{\lambda_{max}^{\alpha}}{\lambda_{min}^{\alpha}}\right), \quad s.t. \ \max_{\alpha} \sum\nolimits_{i} \frac{\lambda_i^{\alpha}}{\sum_i \lambda_i^{\alpha}} \log\left(\frac{\lambda_i^{\alpha}}{\sum_i \lambda_i^{\alpha}}\right), \tag{3}$$

where λ_{max} and λ_{min} are maximum and minimum of $\mathbf{\Lambda}$, respectively. Particularly, we minimize the ratio of λ_{max}^{α} to λ_{min}^{α} (in logarithmic coordinates) to enforce representation decorrelation. Here, smaller values of the objective function indicate stronger decorrelation, and $\alpha = 0$ achieves minimum (i.e., zero). On the other hand, we maximize negative entropy of normalized eigenvalues to keep discrepancy among eigenvalues and so preserve information, which achieves maximum value (i.e., zero) for $\alpha \to +\infty$. By considering that previous works [29, 30] show relation between performance and α is generally convex, we exploit a grid search strategy to obtain a suboptimal solution for $\alpha \in (0, 1]$ with the interval of 0.01[2]. However, λ_{min} is usually nearby zero, leading to computational instability. Therefore, we optimize the objective function (3) as

$$\min_{\alpha} \big[\underbrace{\alpha(\log(\lambda_{max}) - \log(\max(C, \lambda_{min})))}_{\text{representation decorrelation}} \underbrace{-\tau \sum\nolimits_{i=1}^{K} (\lambda_i^{\alpha} / \sum\nolimits_{i=1}^{K} \lambda_i^{\alpha}) \log(\lambda_i^{\alpha} / \sum\nolimits_{i=1}^{K} \lambda_i^{\alpha})}_{\text{information preservation}} \big], \tag{4}$$

where we set minimum of λ_{min} to a constant C (e.g., 1e-2), and compute the entropy using Top-K eigenvalues. τ is a balance parameter, which is set to 1.2. Note that Eqn. (4) is computed efficiently for small K ($K \leq 3$ in our work) using about a hundred trials, given the eigenvalues $\mathbf{\Lambda}$.

MPN with $\alpha > 1$ Theoretically, α of MPN could be larger than one, which, however, violates the principle of representation decorrelation. Meanwhile, it disrupts equilibrium between eigenvalues, hurting information inherent in covariances. As illustrated in Fig. 2, MPN with $\alpha = 2$ suffers from heavy over-fitting, and is clearly worse than one with $\alpha = 0.5$.

Matrix Logarithm Normalization (LogM) LogM [23] is one another popular structure-wise post-normalization. Mathematically, logarithm has similar behavior with power function of $\alpha = 0.3$ when eigenvalues $\lambda > 1$, but reverses amplitudes of eigenvalues $\lambda < 1$ (i.e., neither decorrelation nor information preservation). However, small eigenvalues frequently occur in deep models [29], especially for large feature dimension (d). Accordingly, as shown in Fig. 2, LogM achieves similar

[2] Note that it is unnecessary to obtain exactly the optimal solution, since performance gap is negligible when difference of α is less than 0.01.

results with MPN of $\alpha = 0.3$ for $d = 128$. For $d = 256$, GCP network with LogM fails to converge due to side effect brought by small eigenvalues ($\lambda < 1$). *Therefore, we modified LogM with a shift operation (namely I-LogM) to make all inputs of LogM be larger than one, i.e.,* $\log(\lambda_i) \longrightarrow \epsilon \log((\lambda_i + \epsilon)/\epsilon)$. Our I-LogM brings promising gains when $d = 256$ (see Sec. 5.2 and Table 6).

Element-wise Normalization (EwN) EwN could be roughly written in a unified formulation: $\boldsymbol{\beta}\frac{f(\mathbf{z})-\mu}{\|f(\mathbf{z})-\mu\|_*} + \boldsymbol{\gamma}$, where $\mathbf{z}$ is vectorization of $\mathbf{X}^T\mathbf{X}$. f is a mapping function and $\|\mathbf{a}\|_*$ indicates a certain norm of $\mathbf{a}$. $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ perform a linear transformation (LT). Particularly, for element-wise power normalization (EPN) [31], f is a signed power function and $\mu = 0$. $\|\mathbf{a}\|_*$ is a ℓ_2 norm, and no LT is performed. For LN [2], f is an identity mapping and μ is mean of $f(\mathbf{z})$. $\|\mathbf{a}\|_*$ is a ℓ_2 norm. According to above formulation and Corollary 1, EwN performs representation decorrelation based on zero-mean (i.e., $f(\mathbf{z}) - \mu$) and standardization (i.e., normalized by $\|f(\mathbf{z}) - \mu\|_*$), while LT operation aims to recover information. Compared to MPN, element-wise normalization lacks an effective mechanism to balance representation decorrelation and information preservation. Particularly, as shown in Sec. 5.2, EPN does not work well, since only representation decorrelation is performed. *Therefore, we introduce a LT for EPN, and it achieves promising improvement* (see Table 6).

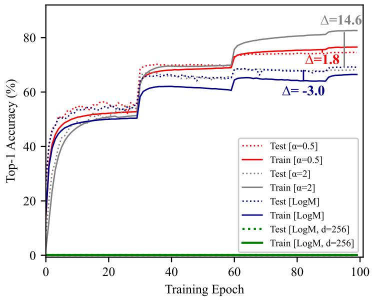


Figure 2: Results of $\alpha > 1$ & LogM

3 DropCov: A Simple yet Effective Normalization

3.1 Normalizing GCP via Dropout

Corollary 1 shows the core effect of effective post-normalization (e.g., MPN) lies in a good trade-off between representation decorrelation and information preservation for optimizing deep GCP networks. However, MPN suffers from high computational complexity (i.e., $O(d^3)$ of d-dimensional features), especially for high-dimensional inputs. Previous works [41, 5, 4] show that dropout can reduce correlation among features. Therefore, one question is naturally raised up: *Could we use efficient dropout to perform representation decorrelation for GCP?* Given probability ρ of dropout, $\rho \rightarrow 1$ leading to sparser, less correlated features but more information loss. On the contrary, $\rho \rightarrow 0$ preserves more information while performing less decorrelation. Above observations show consistence with Corollary 1. However, there exist two issues to apply dropout for normalizing GCP: (1) how to perform dropout and (2) how to choose probability ρ of dropout. A naive method is to perform dropout with a fixed ρ after covariance representations (as compared in Table 2), but it has several limitations: first, dropout after covariance representations will break their structures (e.g., row i of covariance means correlations among i-th channel and all ones); second, fixed ρ hardly achieves trade-off between representation decorrelation and information preservation; third, complexity of such strategy is quadratic to feature dimension d (i.e., $O(d^2)$), and is still high for large d.

3.2 Pre-Normalization via Adaptive Channel Dropout (ACD)

To handle above issues, this work proposes a pre-normalization method (namely DropCov) by performing channel dropout with an adaptive probability ρ on features $\mathbf{X}$ before GCP, which aims to reach a good trade-off between representation decorrelation and information preservation. Since DropCov performs representation decorrelation via an adaptive dropout operation on channel features before GCP, it can maintain structure of covariance representations, while having a more efficient, linear computational complexity of $O(d)$. To reach the trade-off with adaptive probability ρ, we compute it by considering two factors: (1) feature dimension (d) and (2) relationship between feature correlation and feature importance. Then, we have

$$\begin{aligned} \rho &= 1 - \frac{D}{\log(d)}\left(\frac{\boldsymbol{\omega}^T\boldsymbol{\pi}}{\|\boldsymbol{\omega}\|\|\boldsymbol{\pi}\|}\right), \\ \boldsymbol{\omega} &= \sigma(\mathrm{C1D}_3(\mathrm{GAP}(\mathbf{X}))), \quad \boldsymbol{\pi} = \mathrm{SUM}_{\mathrm{row}}(\mathbf{X}^T\mathbf{X}), \end{aligned} \tag{5}$$

where $\boldsymbol{\omega}$ and $\boldsymbol{\pi}$ represent feature importance and feature correlation, respectively. D is a scaling parameter, which is set to 10 throughout all experiments. Since self-attention mechanisms [21, 46] are

Table 1: Comparison of our DropCov with close related works in terms of formulation, computational complexity of post-normalization and usage mode. Please refer to Sec. 4 for the details of symbols.

	Method	Formulation	Complexity	Train	Infer.
Element-wise Post-Norm.	B-CNN [31]	$\ell_2(sqrt(\mathrm{V}(\mathbf{X}^T\mathbf{X})))$	$O(d^2)$	✓	✓
	SigmE [25]	$2/(1+e^{-\beta\cdot\mathrm{V}(\mathbf{X}^T\mathbf{X})})-1$	$O(d^2)$	✓	✓
	LN [2]	$\boldsymbol{\beta}\odot(\mathrm{V}(\mathbf{X}^T\mathbf{X})-\mu)/\sigma)\oplus\boldsymbol{\gamma}$	$O(d^2)$	✓	✓
Structure-wise Post-Norm.	DeepO$_2$P [23]	$\mathrm{V}(\mathrm{LogM}(\mathbf{X}^T\mathbf{X}))$	$O(d^3)$	✓	✓
	MPN-COV [29]	$\mathrm{V}((\mathbf{X}^T\mathbf{X})^{\alpha})$	$O(d^3)$	✓	✓
	IB-CNN [30]	$\ell_2(sqrt(\mathrm{V}(\mathbf{X}^T\mathbf{X})^{1/2}))$	$O(d^3)$	✓	✓
	iSQRT-COV [28]	$\mathrm{V}(\approx(\mathbf{X}^T\mathbf{X})^{1/2})$	$O(d^3)$	✓	✓
	MaxExp [25]	$\mathbf{I}-(\mathbf{I}-\mathbf{X}^T\mathbf{X}/(\mathbf{X}^T\mathbf{X}+\varepsilon))^{\alpha}$	$O(d^3)$	✓	✓
Ours	DropCov	$\mathrm{V}(\mathbf{Y}^T\mathbf{Y}),\ \mathbf{Y}=\delta_{\rho}(\mathbf{X})$	$O(d)$	✓	×

⁎: Note β and ε are parameters. $\odot$ and $\oplus$ indicate element-wise multiplication and addition, respectively.

usually used to search important features, we measure feature importance by modifying a light-weight channel attention module [46], which involves global average pooling (GAP), a 1D convolution with kernel size of 3 (C1D$_3$), and softmax function (σ). As covariance $\mathbf{X}^T\mathbf{X}$ naturally characterizes correlation among different features, we compute feature correlation by summarizing the elements along row of $\mathbf{X}^T\mathbf{X}$ (i.e., $\mathrm{SUM}_{\mathrm{row}}(\mathbf{X}^T\mathbf{X})$). Note that we restrict ρ of Eqn. (5) belongs to $(0, 1)$.

As for Eqn. (5), we adaptively decide probability ρ of channel dropout for reaching a good trade-off between representation decorrelation and information preservation, where $\frac{D}{\log(d)}$ and inner product of $\langle\boldsymbol{\omega},\boldsymbol{\pi}\rangle$ are designed to consider effect of feature dimension (d) and relationship between feature correlation and feature importance, respectively. In particular, larger feature dimension d usually requires larger dropout probability ρ. Meanwhile, feature correlation (i.e., $\boldsymbol{\pi}$) and feature importance (i.e., $\boldsymbol{\omega}$) are closely related to representation decorrelation and information preservation, respectively. Clearly, larger feature correlation results in stronger representation correlation, while features with larger channel weights contain more important information. Therefore, relationship between feature correlation and feature importance is good indicator to perform channel dropout. Intuitively, if feature correlation is closely related to (i.e., seriously decoupled with) feature importance, it is hard to select features for reaching a good trade-off between representation decorrelation and information preservation, and so we need to carefully adopt dropout for channel features under the random setting, leading to a small ρ. Otherwise, we can perform dropout more safely to achieve a trade-off, and adopt a large ρ. Note that we show behavior of ρ during training in supplementary materials. Based on Eqn. (5), our DropCov is achieved by

$$\mathbf{z}=\mathrm{V}(\mathbf{Y}^T\mathbf{Y}),\ \mathbf{Y}=\delta_{\rho}(\mathbf{X}), \tag{6}$$

where δ_ρ is a channel dropout operation with the probability of ρ. V is a vectorization operation followed by triangulation, leading to a $d(d+1)/2$-dimensional covariance representation $\mathbf{z}$. Note that our DropCov (6) is only adopted to train GCP networks, which can be easily implemented by standard back-propagation. During inference stage, we simply use $\mathrm{V}(\mathbf{X}^T\mathbf{X})$ for final classification.

4 Related Works

In this section, we review some works closely related to our DropCov. Table 1 gives a detailed comparison of our DropCov with existing post-normalization methods in terms of formulation, computational complexity of post-normalization and usage mode. For element-wise post-normalization methods, B-CNN[3] [31] performs element-wise signed square-root ($sqrt$) followed by a ℓ_2 normalization on covariance representations. [25] proposes to use some surrogate functions (e.g., SigmE) to handle issue of negative evidence inherent in power function. Layer normalization (LN) [2] is proposed to normalize each feature with its mean (μ) and variance (γ) followed by a linear transformation. Since these

[3] Note there exist small differences between GCP and bilinear pooling of B-CNN. Specifically, GCP computes covariance of inputs $\mathbf{X}$ with mean of $\boldsymbol{\mu}$, while bilinear pooling computes outer product of inputs $\mathbf{X}$ and $\mathbf{Y}$. When $\mathbf{X}$ and $\mathbf{Y}$ are shared and zero-mean, bilinear pooling captures the same information with one of GCP.

methods perform normalization on each element of covariance representations, they have the computational complexity of $O(d^2)$ for d-dimensional features $\mathbf{X}$. For structure-wise post-normalization methods, DeepO$_2$P and MPN-COV [29] perform matrix logarithm (LogM) and matrix power function for covariance representations, respectively. IB-CNN [30] combines matrix square-root normalization with element-wise one, while iSQRT-COV [28] proposes an approximate matrix square-root normalization via fast iterative matrix multiplications. Song et al. further analyze the effect of iSQRT-COV [38] and develop a faster variant [39]. [25] introduces several spectral power functions (e.g., MaxExp) for normalizing covariance matrices. These structure-wise approaches involve SVD of covariances or sequential matrix multiplications, having the computational complexity of $O(d^3)$. ReDro [36] proposes a relation dropout scheme to divide a large-sized covariance into a group of small-sized ones, aiming to mitigate the computational issue of matrix normalization [29, 30, 28, 11]. ReDro reduces computational complexity of structure-wise approaches from $O(d^3)$ to $O(d^3/G^2)$, where G is the number of groups. Besides, all existing works perform post-normalization methods during both training and inference (Infer.) stages. Different from aforementioned works, our DropCov performs adaptive channel dropout δ on features $\mathbf{X}$ right before GCP, which has a linear complexity $O(d)$ and is free during inference stage. Experimental results in Sec. 5.3 show our DropCov with much less computational complexity is superior or competitive to previous works.

5 Experiments

In this section, we conduct experiments to evaluate effectiveness of our DropCov in improving deep architectures on image classification tasks. Specifically, we first describe implementation details, and then make ablation study on ImageNet-1K (IN-1K) [7] using backbone of ResNet-18. Additionally, we apply our DropCov to both ResNets [17] and ViT models [43, 32, 50], while comparing them on several benchmarks (i.e., IN-1K, ImageNet-C (IN-C) [18], ImageNet-A (IN-A) [19] and Stylized-ImageNet (Sty.-IN) [13]) to assess generalization and robustness of our DropCov. Finally, we assess generalization of DropCov models by transferring them to long-tailed species classification [20].

Table 2: Comparisons (% in Top-1 accuracy) of adaptive normalization methods with the fixed ones using ResNet-18 on ImageNet-1K.

Method	$d = 64$	$d = 128$	$d = 256$
DropElement ($\rho = 0.5$)	73.4	74.6	74.0
DropChannel ($\rho = 0.5$)	70.1	73.1	75.1
ACD (Ours)	**73.5**	**75.0**	**75.2**
MPN ($\alpha = 0.5$)	73.1	74.4	74.9
APN (Ours)	73.3	74.5	75.0

Table 3: Comparisons (% in Top-1 accuracy) of several specific dropout strategies using ResNet-18 on ImageNet-1K.

Method	$d = 64$		$d = 256$	
	Element	Channel	Element	Channel
Large ($\rho = 0.5$)	72.2	71.3	71.4	70.9
Small ($\rho = 0.5$)	68.4	71.2	66.4	71.5
Uniform ($\rho = 0.5$)	72.8	72.1	74.0	74.4
ACD (Ours)	**73.5**		**75.2**	

Table 4: Comparisons (% in Top-1 accuracy) of various dropout variants using ResNet-18 on ImageNet-1K.

Method	dim=64	dim=256
Maxout [15]	72.1	73.7
Dropconnect [44]	70.6	72.5
Decov [5]	72.7	74.0
Maxdropout [9]	72.0	70.1
DropBlock [14]	72.3	74.1
ACD(Ours)	**73.5**	**75.2**

Table 5: Comparisons (% in Top-1 accuracy) of DropElement and DropChannel with various drop rates (ρ) using ResNet-18 on ImageNet-1K.

	Channel Dropout		Element Dropout	
ρ	dim = 64	dim = 256	dim = 64	dim = 256
0.1	72.5	71.9	72.3	70.8
0.3	72.8	74.7	72.8	72.3
0.5	70.1	75.1	73.4	74.0
0.7	65.7	72.1	72.1	74.2
0.9	20.5	54.7	68.2	73.8
ACD (Ours)	**73.5**	**75.2**	**73.5**	**75.2**

5.1 Implementation Details

In this work, we apply our DropCov to several ResNets [17] and ViT models [43, 32, 50]. Specifically, we construct DropCov models based on ResNet following exactly the same schemes in [29, 47]. Given a CNN backbone, we first introduce a 1×1 convolution after the last convolution layer for reducing dimension of features to d, and then replace the original GAP by our DropCov followed by a fully-connected (FC) layer and softmax for classification. All models are optimized using the same hyper-parameter settings and data augmentations as suggested in [17, 29], where stochastic gradient descent (SGD) with initial learning rate (lr) of 0.1 is used to train the networks within 100 epochs and lr is decayed by 10 every 30 epochs. We combine DropCov with ViT models as in [49], where one FC layer is employed to reduce dimension of final word tokens, and our DropCov replaces GAP and classification token for Swin Transformers [32] and DeiT/T2T-ViT [43, 50], respectively. These models are trained by using the same hyper-parameter settings in [43, 32, 50]. All programs run a sever equipped with eight Nvidia RTX 3090 GPUs and 128G RAM. Source code (e.g., PyTorch [33], PaddlePaddle and Mindspore) will be available at `https://github.com/mingzeG/DropCov`.

5.2 Ablation Study

In this subsection, we conduct experiments on ImageNet-1K using lightweight ResNet-18 to assess effect of adaptive probability ρ of dropout on our method, while compare with several dropout strategies, and finally assess effect of our modifications on existing post-normalization approaches.

Fixed ρ *vs.* ACD. Our DropCov performs pre-normalization for GCP by using an adaptive channel dropout (ACD) method, which automatically determines probability ρ of dropout by considering trade-off between representation decorrelation and information preservation. We compare with two kinds of baseline dropout methods using fixed $\rho = 0.5$, i.e., element-wise dropout for covariance representations (DropElement) and channel dropout for input features (DropChannel). As shown in the upper part of Table 2, our ACD achieves the best results for various feature dimensions (d) and clearly outperforms the naive DropElement in Sec. 3.1. Besides, we compare with DropElement and DropChannel with various dropout ratios. As compared in Table 5, our ACD is superior to DropElement and DropChannel for all dropout ratios. Particularly, the best dropout ratios are quite different for two methods and various feature dimensions. In contrast, our ACD can adaptively determine probability ρ of dropout by balancing representation decorrelation and information preservation, while always achieving the best performance, clearly verifying ACD can achieve a better trade-off.

Comparison of Various Dropout Strategies. To verify the effectiveness of our ACD, we compare with several dropout strategies. First, we perform element-wise (Element) dropout on some specific entries of covariance with $\rho = 0.5$. Specifically, we rank all entries of covariance in a descending order, and then perform dropout on top/bottom/uniform half of the ordered entries (namely Large/Small/Uniform). We adopt similar strategies to feature channels (Channel), where channel indicators are decided by attention module of our ACD. Note that 'Element' dropout preforms dropout on elements of covariance representations based on values of elements ($[\mathbf{X}^T\mathbf{X}]_{ij}$), which indicates correlation between i-th channel and j-th channel. Therefore, it only considers feature correlation. For 'Channel' dropout, it preforms dropout on feature channels based on their weights, which are obtained by attention module of ACD (i.e., $\boldsymbol{\omega}$). Therefore, it only considers feature importance. Based on above discussion, 'Element' and 'Channel' dropout methods can be regarded as special cases of ACD, where only feature correlation or feature importance are considered, respectively. As compared in Table 3, uniform dropout clearly outperforms those only focus on largest and smallest entries, indicating equilibrium of feature correlation or importance is crucial to final performance. Furthermore, our ACD is superior to uniform dropout by a clear margin. Besides, we compare with several existing Dropout variants, including Maxout [15], DropConnect [44], Decov [5], maxdropout [9] and DropBlock [14]. As shown in Table 4, our ACD clearly outperforms them. Although these methods can prevent overfitting, they are not good at making a balance between representation decorrelation and information preservation. These results further demonstrate the effectiveness of our ACD.

Effect of Modifications. According to Corollary 1, we make several modifications to existing post-normalization approaches. Beyond MPN, we propose an APN method by considering eigenvalues $\mathbf{\Lambda}$ of inputs (see Eqns. (3) and(4)). The results are compared in the bottom part of Table 2, where our APN determines α by considering effect of inputs while average values of α are 0.53, 0.49 and 0.45 for $d = 256$, $d = 128$ and $d = 64$, respectively. Clearly, average values of α achieved by APN are nearby 0.5, which further account for why 0.5 is the widely used choice of α for MPN. Besides, APN brings 0.1%∼0.2% gains over MPN with $\alpha = 0.5$ by considering effect of inputs. We would like to clarify they are non-trivial gains on ImageNet over strong MPN, and the recent works [38, 40] also bring similar gains (0.1%∼0.3%) over MPN. Besides, as discussed in Sec.2.2, we introduce an I-LogM and a linear transform (LT) based on BN [22] for improving LogM normalization (DeepO$_2$P)

Table 6: Comparison of DropCov with several post-normalization in terms of Top-1 accuracy (Acc.) and running speed using ResNet-18 on ImageNet-1K. *: According to Sec.2.2 and Corollary 1, we modify B-CNN and IB-CNN with a linear transform (LT), while modifying DeepO$_2$P with I-LogM.

Method	$d = 64$			$d = 256$		
	Top-1 Acc. (%)	Train (µs)	Test (µs)	Top-1 Acc. (%)	Train (µs)	Test (µs)
Plain GCP	71.1	3.45	1.04	70.0	35.08	11.89
B-CNN [31]	38.3	4.31	1.23	41.1	44.97	14.38
B-CNN + LT*	68.3	4.45	1.28	73.2	47.37	15.24
LN [2]	71.7	4.10	1.13	70.2	40.79	13.59
DeepO$_2$P [23]	70.1	922.50	910.95	*Not Converge*	9731.71	9638.71
I-LogM*	71.2	922.53	910.98	72.0	9732.46	9639.65
MPN-COV [29]	73.1	925.16	913.87	74.9	9735.11	9642.43
iSQRT-COV [28]	73.4	12.35	4.94	75.2	193.65	77.46
IB-CNN [30]	*Not Converge*	13.93	5.16	36.1	202.38	80.55
IB-CNN + LT*	70.0	14.16	5.21	72.8	207.48	82.98
DropCov (Ours)	**73.5**	**3.54**	**1.04**	**75.2**	**36.20**	**11.89**

$\diamondsuit$: Note we compute running speed (µs) of single GCP module with post-normalization on a 2080Ti GPU.
$\sharp$: The original ResNet-18 with GAP achieves 70.2% in Top-1 accuracy.

and element-wise post-normalization (i.e., B-CNN and IB-CNN), respectively. As shown in Table 6, DeepO$_2$P performs poorly troubled by small eigenvalues (especially for $d = 256$), while our I-LogM obtains promising gains for both low- and high-dimensional features. Furthermore, B-CNN and IB-CNN achieve very unsatisfied results; in contrast, our simple modification brings significant improvement for B-CNN and IB-CNN. Above results clearly verify our finding in Corollary 1.

5.3 Comparisons with Counterparts

Here, we conduct experiments on ImageNet-1K using ResNet-18 for comparing with existing post-normalization methods in terms of Top-1 accuracy and running speed, including B-CNN [31], LN [2], DeepO$_2$P [23], MPN-COV [29] with $\alpha = 0.5$, iSQRT-COV [28] and IB-CNN [30]. The results of all compared methods are listed in Table 6, where we can see that our DropCov outperforms plain GCP (i.e., GCP without normalization) by 2.4% (5.2%) for $d = 64$ ($d = 256$) with similar running time. Compared with element-wise post-normalization (i.e., LN, B-CNN and IB-CNN), our DropCov achieves significant performance gains while requiring less running time. For structure-wise post-normalization, our DropCov is remarkably superior to matrix logarithm normalization (DeepO$_2$P) in terms of both accuracy and running speed. Meanwhile, DropCov is superior or comparable to MPN-COV and iSQRT-COV using much less running time, especially during inference stage. Above results show our DropCov performs better or on par with the counterparts in terms of efficiency and effectiveness, providing an very promising normalization method for deep GCP networks.

5.4 Comparisons with Various Deep Architectures

CNNs. Here we apply our DropCov with $d = 128$ to various CNNs, including ResNet-34, ResNet-50 and ResNet-101. Meanwhile, we compare DropCov with the original CNNs on IN-1K, IN-C, IN-A and Sty.-IN. As shown in the upper part of Table 7, DropCov respectively brings about 2.6%, 2.2% and 1.8% gains over ResNet-34, ResNet-50 and ResNet-101 on IN-1K, while showing stronger robustness on IN-C, IN-A and Sty.-IN. Particularly, DropCov with ResNet-50 and ResNet-101 are respectively superior to the original ResNet-101 and ResNet-152, but having less model complexity.

ViT. For ViT models, we apply our DropCov with $d = 128$ to DeiT-S [43], Swin-T [32] and T2T-ViT-14 [50], and compare the original models as well as ConViT [6]. As shown in the bottom part of Table 7, DropCov brings about 2.6%, 1.3% and 1.2% gains over DeiT-S, Swin-T and T2T-ViT-14 on IN-1K, respectively. Meanwhile, it shows stronger robustness on IN-C, IN-A and Sty.-IN. Particularly, DropCov with DeiT-S is superior to DeiT-B and ConViT-B, but having less model complexity. Above results show our DropCov provides a simple yet effective method to improve deep architectures, and deep architectures with DropCov achieve better trade-off between accuracy and model complexity.

Table 7: Comparison of our DropCov with various ResNet and ViT models on four datasets, where results in terms of mCE and Top-1 accuracy (%) are reported on IN-C and reminding ones, receptively.

Method	Params.	FLOPs	IN-1K (↑)	IN-C (↓)	IN-A (↑)	Sty.-IN (↑)
ResNet-34 [17]	21.8 M	3.66 G	74.19	77.9	1.63	7.59
ResNet-50 [17]	25.6 M	3.86 G	76.02	76.7	2.47	7.15
ResNet-101 [17]	44.6 M	7.57 G	77.67	70.3	4.15	9.51
ResNet-152 [17]	60.2 M	11.28 G	78.13	69.3	5.98	10.09
ResNet-34+DropCov (Ours)	29.6 M	5.56 G	$76.81_{(2.62)}$	$71.1_{(6.8)}$	$3.45_{(1.82)}$	$11.16_{(3.57)}$
ResNet-50+DropCov (Ours)	32.0 M	6.19 G	$78.19_{(2.17)}$	$69.8_{(6.9)}$	$5.08_{(2.61)}$	$9.90_{(2.75)}$
ResNet-101+DropCov (Ours)	51.0 M	9.90 G	**79.51**$_{(1.84)}$	**65.8**$_{(4.5)}$	**7.54**$_{(3.39)}$	**11.41**$_{(1.90)}$
DeiT-S [43]	22.1 M	4.6 G	79.8	54.6	18.9	14.91
Swin-T [32]	28.3 M	4.5 G	81.2	62.0	21.6	13.40
T2T-ViT-14 [50]	21.5 M	5.2 G	81.5	53.2	23.9	15.80
DeiT-B [43]	86.6 M	17.6 G	82.0	48.5	27.4	17.94
ConViT-B [6]	86.5 M	17.7 G	82.4	**46.9**	29.0	**19.67**
DeiT-S+DropCov (Ours)	25.6 M	5.5 G	$82.4_{(2.6)}$	$52.6_{(2.0)}$	$31.2_{(12.3)}$	$17.10_{(2.19)}$
Swin-T+DropCov (Ours)	31.6 M	6.0 G	$82.5_{(1.3)}$	$54.8_{(7.2)}$	$33.1_{(11.5)}$	$14.13_{(0.73)}$
T2T-ViT-14+DropCov (Ours)	24.9 M	5.4 G	**82.7**$_{(1.2)}$	$52.1_{(1.1)}$	**31.7**$_{(7.8)}$	$18.81_{(3.01)}$

5.5 Comparisons on Long-tailed Benchmark

Table 8: Comparisons (%) on iNat2017.

Method	Top-1	Top-5
ResNet-101	62.4	84.1
ResNet-152	64.2	85.5
IncV3 SE	66.3	86.7
ViT-B_16 (IN-21K) [10]	68.7	N/A
TransFG (IN-21K) [16]	71.7	N/A
DropCov (ResNet-101)	67.9	87.3
DropCov (DeiT-S)	**72.4**	**90.3**

Finally, we transfer our DropCov models to iNat2017 [20], and compare with several deep architectures, i.e., ResNet-101, ResNet-152, Inception V3 (IncV3) [42] with SE [21] as well as the recently proposed ViT [10] and TransFG [16]. Specifically, we perform fine-tuning of our DropCov models with backbones of ResNet-101 and DeiT-S on iNat2017, namely DropCov (ResNet-101) and DropCov (DeiT-S). As shown in Table 8, our DropCov (ResNet-101) obtains 5.5% gains over the original ResNet-101, while outperforming ResNet-152 and IncV3 SE. Besides, our DropCov (DeiT-S) respectively outperforms ViT-B and TransFG by 3.7% and 0.7%, which are pre-trained on ImageNet-21K. The results clearly verify generalization of DropCov.

6 Conclusion

In this paper, we first analyze the effect of post-normalization on GCP from the perspective of optimizing deep GCP networks. Particularly, we find that effective post-normalization methods have a good ability to balance representation decorrelation and information preservation, which can reduce over-fitting and increase representation ability of deep GCP networks, respectively. According to our finding, we introduce some strategies to improve existing normalization methods, further verifying our finding. More importantly, we propose a novel pre-normalization for GCP (namely DropCov) based on adaptive channel dropout, which achieves very competitive performance with a linear complexity and training-only mode. Extensive experiments identify that our DropCov provides a simple yet effective solution to improve existing deep architectures on image classification tasks.

GCP normally produces a high-dimensional covariance representation, bringing a certain amount of computational cost, but our work makes use of high-dimensional GCP more flexible. Meanwhile, our DropCov can achieve clear gains for low-dimensional GCP with little extra computational cost (refer to Table 6). Additionally, our DropCov is potentially applicable to compact GCP methods [12, 24] and encourages exploration of other effective dropout strategies, which will be studied in future work.

Acknowledgment

The work was sponsored by National Natural Science Foundation of China (Grant No.s 62276186, 61925602, 61971086 and 61732011), CCF-Baidu Open Fund (NO.2021PP15002000), CAAIXSJLJJ-2022-010C and Haihe Lab of ITAI (NO. 22HHXCJC00002).

References

[1] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Fast and simple calculus on tensors in the Log-Euclidean framework. In *MICCAI*, 2005.

[2] L. J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[3] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[4] G. Chen, P. Chen, Y. Shi, C. Hsieh, B. Liao, and S. Zhang. Rethinking the usage of batch normalization and Dropout in the training of deep neural networks. *CoRR*, abs/1905.05928, 2019.

[5] M. Cogswell, F. Ahmed, R. B. Girshick, L. Zitnick, and D. Batra. Reducing overfitting in deep networks by decorrelating representations. In *ICLR*, 2016.

[6] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun. ConViT: Improving vision transformers with soft convolutional inductive biases. In *ICML*, 2021.

[7] 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.

[8] A. Diba, V. Sharma, and L. Van Gool. Deep temporal linear encoding networks. In *CVPR*, 2017.

[9] C. F. G. dos Santos, D. Colombo, M. Roder, and J. P. Papa. Maxdropout: Deep neural network regularization based on maximum output values. In *ICPR*, 2020.

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[11] M. Engin, L. Wang, L. Zhou, and X. Liu. DeepKSPD: Learning kernel-matrix-based SPD representation for fine-grained image recognition. In *ECCV*, 2018.

[12] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. In *CVPR*, 2016.

[13] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *ICLR*, 2019.

[14] G. Ghiasi, T. Lin, and Q. V. Le. DropBlock: A regularization method for convolutional networks. In *NeurIPS*, 2018.

[15] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

[16] J. He, J.-N. Chen, S. Liu, A. Kortylewski, C. Yang, Y. Bai, C. Wang, and A. Yuille. TransFG: A transformer architecture for fine-grained recognition. In *AAAI*, 2022.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[18] D. Hendrycks and T. G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

[19] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song. Natural adversarial examples. In *CVPR*, 2021.

[20] G. V. Horn, O. M. Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. J. Belongie. The iNaturalist species classification and detection dataset. In *CVPR*, 2018.

[21] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(8):2011–2023, 2020.

[22] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[23] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015.

[24] S. Kong and C. C. Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *CVPR*, 2017.

[25] P. Koniusz, H. Zhang, and F. Porikli. A deeper look at power normalizations. In *CVPR*, 2018.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[27] Y. LeCun, I. Kanter, and S. A. Solla. Second order properties of error surfaces. In *NIPS*, 1990.

[28] P. Li, J. Xie, Q. Wang, and Z. Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *CVPR*, 2018.

[29] P. Li, J. Xie, Q. Wang, and W. Zuo. Is second-order information helpful for large-scale visual recognition? In *ICCV*, 2017.

[30] T. Lin and S. Maji. Improved bilinear pooling with CNNs. In *BMVC*, 2017.

[31] T. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *ICCV*, 2015.

[32] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[34] X. Pennec, P. Fillard, and N. Ayache. A Riemannian framework for tensor computing. *IJCV*, 66(1):41–66, 2006.

[35] T. Poggio, A. Banburski, and Q. Liao. Theoretical issues in deep networks. *Proceedings of the National Academy of Sciences (PNAS)*, 117(48):30039–30045, 2020.

[36] S. Rahman, L. Wang, C. Sun, and L. Zhou. ReDro: Efficiently learning large-sized SPD visual representation. In *ECCV*, 2020.

[37] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher Vector: Theory and practice. *IJCV*, 105(3):222–245, 2013.

[38] Y. Song, N. Sebe, and W. Wang. Why approximate matrix square root outperforms accurate SVD in global covariance pooling? In *ICCV*, 2021.

[39] Y. Song, N. Sebe, and W. Wang. Fast differentiable matrix square root. In *ICLR*, 2022.

[40] Y. Song, N. Sebe, and W. Wang. Improving covariance conditioning of the SVD meta-layer by orthogonality. In *ECCV*, 2022.

[41] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[43] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

[44] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

[45] Q. Wang, P. Li, W. Zuo, and L. Zhang. RAID-G: Robust estimation of approximate infinite dimensional Gaussian with application to material recognition. In *CVPR*, 2016.

[46] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu. ECA-Net: Efficient channel attention for deep convolutional neural networks. In *CVPR*, 2020.

[47] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li. Deep CNNs meet global covariance pooling: Better representation and generalization. *IEEE TPAMI*, 43(8), 2020.

[48] T. Winterbottom, S. Xiao, A. McLean, and N. A. Moubayed. Trying bilinear pooling in Video-QA. *arXiv preprint arXiv:2012.10285*, 2020.

[49] J. Xie, R. Zeng, Q. Wang, Z. Zhou, and P. Li. So-ViT: Mind visual tokens for vision transformer. *CoRR*, abs/2104.10935, 2021.

[50] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan. Tokens-to-Token ViT: Training vision transformers from scratch on ImageNet. In *ICCV*, 2021.

[51] J. Zhang, L. Wang, L. Zhou, and W. Li. Beyond Covariance: SICE and kernel based visual feature representation. *Int. J. Comput. Vis.*, 129(2):300–320, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Abstract and Section 1.
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 5.1 and Supplemental Material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.1 and Supplemental Material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We share the same settings with compared methods.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
    (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5.
    (b) Did you mention the license of the assets? [No] All of them are publicly available.
    (c) Did you include any new assets either in the supplemental material or as a URL? [No] We use publicly available datasets.
    (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] All of them are publicly available.
    (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use publicly available datasets.
5. If you used crowdsourcing or conducted research with human subjects...
    (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
    (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
    (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]