Towards Faster Training of Global Covariance Pooling Networks by Iterative Matrix Square Root Normalization

Peihua Li, Jiangtao Xie, Qilong Wang, Zilin Gao
Dalian University of Technology
peihuali@dlut.edu.cn

Abstract

Global covariance pooling in convolutional neural networks has achieved impressive improvement over the classical first-order pooling. Recent works have shown matrix square root normalization plays a central role in achieving state-of-the-art performance. However, existing methods depend heavily on eigendecomposition (EIG) or singular value decomposition (SVD), suffering from inefficient training due to limited support of EIG and SVD on GPU. Towards addressing this problem, we propose an iterative matrix square root normalization method for fast end-to-end training of global covariance pooling networks. At the core of our method is a meta-layer designed with loop-embedded directed graph structure. The meta-layer consists of three consecutive nonlinear structured layers, which perform pre-normalization, coupled matrix iteration and post-compensation, respectively. Our method is much faster than EIG or SVD based ones, since it involves only matrix multiplications, suitable for parallel implementation on GPU. Moreover, the proposed network with ResNet architecture can converge in much less epochs, further accelerating network training. On large-scale ImageNet, we achieve competitive performance superior to existing counterparts. By finetuning our models pre-trained on ImageNet, we establish state-of-the-art results on three challenging fine-grained benchmarks. The source code and network models will be available at http://www.peihuali.org/iSQRT-COV.

1. Introduction

Deep convolutional neural networks (ConvNets) have made significant progress in the past years, achieving recognition accuracy surpassing human beings in large-scale object recognition [7]. The ConvNet models pre-trained on ImageNet [5] have been proven to benefit a multitude of other computer vision tasks, ranging from fine-grained visual categorization (FGVC) [25], object detection [28], semantic segmentation [26] to scene parsing [37], where labeled data are insufficient for training from scratch. The common layers such as convolution, non-linear rectification, pooling and batch normalization [11] have become off-the-shelf commodities, widely supported on devices including workstations, PCs and embedded systems.

Although the architecture of ConvNet has greatly evolved in the past years, its basic layers largely keep unchanged [19, 18]. Recently, researchers have shown increasing interests in exploring structured layers to enhance representation capability of networks [12, 25, 1, 22]. One particular kind of structured layer is concerned with global covariance pooling after the last convolution layer, which has shown impressive improvement over the classical first-order pooling, successfully used in FGVC [25], visual question answering [15] and video action recognition [34]. Very recent works have demonstrated that matrix square root normalization of global covariance pooling plays a key role in achieving state-of-the-art performance in both large-scale visual recognition [21] and challenging FGVC [24, 32].

For computing matrix square root, existing methods depend heavily on eigendecomposition (EIG) or singular value decomposition (SVD) [21, 32, 24]. However, fast implementation of EIG or SVD on GPU is an open problem, which is limitedly supported on NVIDIA CUDA platform, significantly slower than their CPU counterparts [12, 24]. As such, existing methods opt for EIG or SVD on CPU for computing matrix square root. Nevertheless, current implementations of meta-layers depending on CPU are far from ideal, particularly for multi-GPU configuration. Since GPUs with powerful parallel computing ability have to be interrupted and await CPUs with limited parallel ability, their concurrency and throughput are greatly restricted.

In [24], for the purpose of fast forward propagation (FP), Lin and Maji use Newton-Schulz iteration (called modified Denman-Beavers iteration therein) algorithm, which is proposed in [9], to compute matrix square-root. Unfortunately, for backward propagation (BP), they compute the gradient through Lyapunov equation solution which depends on the

The work was supported by National Natural Science Foundation of China (No. 61471082). Peihua Li is the corresponding author.

Method	Forward Prop. (FP)	Backward Prop. (BP)	CUDA support	Scalability to multi-GPUs	Large-scale (LS) or Small-scale (SS)
MPN-COV [21]	**EIG algorithm**	BP of EIG	limited	limited	LS only
G^2DeNet [32]	**SVD algorithm**	BP of SVD	limited	limited	SS only
Improved B-CNN [24]	Newton-Schulz Iter.	**BP by Lyapunov equation (SCHUR or EIG required)**	limited	limited	SS only
	SVD algorithm	BP of SVD			
iSQRT-COV (ours)	Newton-Schulz Iter.	BP of Newton-Schulz Iter.	good	good	LS+SS

Table 1. Differences between our iSQRT-COV and related methods. The bottleneck operations are marked with red, bold text.

GPU unfriendly Schur-decomposition (SCHUR) or EIG. Hence, the training in [24] is expensive though FP which involves only matrix multiplication runs very fast. Inspired by that work, we propose a fast end-to-end training method, called iterative matrix square root normalization of covariance pooling (iSQRT-COV), depending on Newton-Schulz iteration in both forward and backward propagations.

At the core of iSQRT-COV is a meta-layer with loop-embedded directed graph structure, specifically designed for ensuring both convergence of Newton-Schulz iteration and performance of global covariance pooling networks. The meta-layer consists of three consecutive structured layers, performing pre-normalization, coupled matrix iteration and post-compensation, respectively. We derive the gradients associated with the involved non-linear layers based on matrix backpropagation theory [12]. The design of sandwiching Newton-Schulz iteration using pre-normalization by Frobenius norm or trace and post-compensation is essential, which, as far as we know, did not appear in previous literature (e.g. in [9] or [24]). The pre-normalization guarantees convergence of Newton-Schulz (NS) iteration, while post-compensation plays a key role in achieving state-of-the-art performance with prevalent deep ConvNet architectures, e.g. ResNet [8]. The main differences between our method and other related works[1] are summarized in Tab. 1.

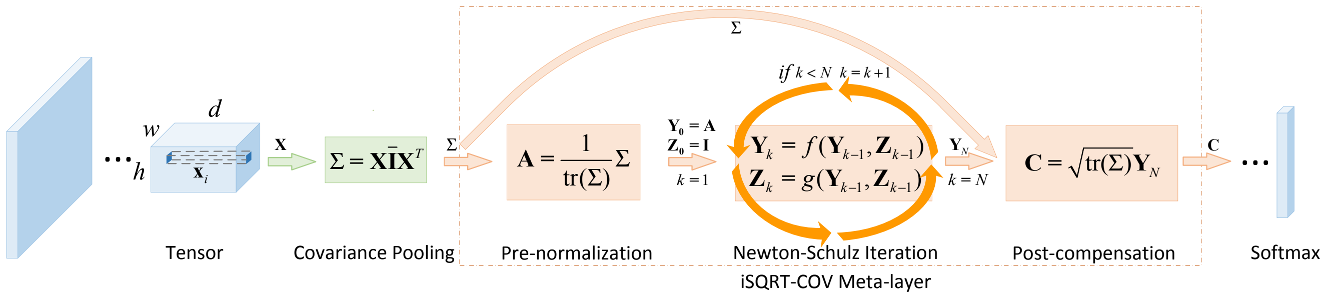
2. Related Work

B-CNN is one of the first end-to-end covariance pooling ConvNets [25, 12]. It performs element-wise square root normalization followed by ℓ_2-normalization for covariance matrix, achieving impressive performance in FGVC task. Improved B-CNN [24] shows that additional matrix square root normalization before element-wise square root and ℓ_2-normalization can further attain large improvement. In training process, they perform FP using Newton-Schulz iteration or using SVD, and perform BP by solving Lyapunov equation or compute gradients associated with SVD. In any case, improved B-CNN suffers from GPU unfriendly SVD, SCHUR or EIG and so network training is expensive. Our iSQRT-COV differs from [24] in three aspects. First, both FP and BP of our method are based on Newton-Schulz iteration, making network training very efficient as only GPU friendly matrix multiplications are involved. Second, we propose sandwiching Newton-Schulz iteration using pre-normalization and post-compensation which is essential and plays a key role in training extremely deep ConvNets. Finally, we evaluate extensively on both large-scale ImageNet and on three popular fine-grained benchmarks.

In [21], matrix power normalized covariance pooling method (MPN-COV) is proposed for large-scale visual recognition. It achieves impressive improvements over first-order pooling with AlexNet [18], VGG-Net [3, 29] and ResNet [8] architectures. MPN-COV has shown that, given a small number of high-dimensional features, matrix power is consistent with shrinkage principle of robust covariance estimation, and matrix square root can be derived as a robust covariance estimator via a von Neumann regularized maximum likelihood estimation [33]. It is also shown that matrix power normalization approximately yet effectively exploits geometry of the manifold of covariance matrices, superior to matrix logarithm normalization [12] for high-dimensional features. All computations of MPN-COV meta-layer are implemented with NVIDIA cuBLAS library running on GPU, except EIG which runs on CPU.

G^2DeNet [32] is concerned with inserting global Gaussian distributions into ConvNets for end-to-end learning. In G^2DeNet, each Gaussian is identified as square root of a symmetric positive definite matrix based on Lie group structure of Gaussian manifold [20]. The matrix square root plays a central role in obtaining the competitive performance [32, Tab. 1 & Tab. 5]. Compact bilinear pooling (CBP) [6] clarifies that bilinear pooling is closely related to the second-order polynomial kernel, and presents two compact representations via low-dimensional feature maps for kernel approximation. Kernel pooling [4] approximates Gaussian RBF kernel to a given order through compact explicit feature maps, aiming to characterize higher order feature interactions. Cai et al. [2] introduce a polynomial kernel based predictor to model higher-order statistics of convolutional features across multiple layers.

[1]It is worth noting that, after CVPR submission deadline, authors of [24] release code of improved B-CNN together with a scheme similar to ours, in which BP of Newton-Schulz iteration is implemented using Autograd package in PyTorch. We note that (1) that scheme is parallel to our work, and (2) they only provide pieces of code but do not train using BP of Newton-Schulz iteration on any real-world benchmarks.



w
d
h
$\mathbf{x}_i$
X
$\Sigma = \mathbf{X}\bar{\mathbf{I}}\mathbf{X}^T$
Σ
$\mathbf{A} = \frac{1}{\mathrm{tr}(\Sigma)}\Sigma$
$\mathbf{Y}_0 = \mathbf{A}$
$\mathbf{Z}_0 = \mathbf{I}$
$k=1$
if $k<N$ $k=k+1$
$\mathbf{Y}_k = f(\mathbf{Y}_{k-1}, \mathbf{Z}_{k-1})$
$\mathbf{Z}_k = g(\mathbf{Y}_{k-1}, \mathbf{Z}_{k-1})$
$\mathbf{Y}_N$
$k=N$
$\mathbf{C} = \sqrt{\mathrm{tr}(\Sigma)}\mathbf{Y}_N$
C
Tensor
Covariance Pooling
Pre-normalization
Newton-Schulz Iteration
Post-compensation
Softmax
iSQRT-COV Meta-layer

Figure 1. Proposed iterative matrix square root normalization of covariance pooling (iSQRT-COV) network. After the last convolution layer, we perform second-order pooling by estimating a covariance matrix. We design a meta-layer with loop-embedded directed graph structure for computing approximate square root of covariance matrix. The meta-layer consists of three nonlinear structured layers, performing pre-normalization, coupled Newton-Schulz iteration and post-compensation, respectively. See Sec. 3 for notations and details.

3. Proposed iSQRT-COV Network

In this section, we first give an overview of the proposed iSQRT-COV network. Then we describe matrix square root computation and its forward propagation. We finally derive the corresponding backward gradients.

3.1. Overview of Method

The flowchart of the proposed network is shown in Fig. 1. Let output of the last convolutional layer (with ReLU) be a $h \times w \times d$ tensor with spatial height h, width w and channel d. We reshape the tensor to a feature matrix $\mathbf{X}$ consisting of $n = wh$ features of $d-$dimension. Then we perform second-order pooling by computing the covariance matrix $\boldsymbol{\Sigma} = \mathbf{X}\bar{\mathbf{I}}\mathbf{X}^T$, where $\bar{\mathbf{I}} = \frac{1}{n}(\mathbf{I} - \frac{1}{n}\mathbf{1})$, $\mathbf{I}$ and $\mathbf{1}$ are the $n \times n$ identity matrix and matrix of all ones, respectively.

Our meta-layer is designed to have loop-embedded directed graph structure, consisting of three consecutive nonlinear structured layers. The purpose of the first layer (i.e., pre-normalization) is to guarantee the convergence of the following Newton-Schulz iteration, achieved by dividing the covariance matrix by its trace (or Frobenius norm). The second layer is of loop structure, repeating the coupled matrix equations involved in Newton-Schulz iteration a fixed number of times, for computing approximate matrix square root. The pre-normalization nontrivially changes data magnitudes, so we design the third layer (i.e., post-compensation) to counteract the adverse effect by multiplying trace (or Frobenius norm) of the square root of the covariance matrix. As the output of our meta-layer is a symmetric matrix, we concatenate its upper triangular entries forming an $d(d+1)/2$-dimensional vector, submitted to the subsequent layer of the ConvNet.

3.2. Matrix Square Root and Forward Propagation

Square roots of matrices, particularly covariance matrices which are symmetric positive (semi)definite (SPD), find applications in a variety of fields including computer vision, medical imaging [38] and chemical physics [14]. It is well-known any SPD matrix has a unique square root which can be computed accurately by EIG or SVD. Briefly, let $\mathbf{A}$ be an SPD matrix and it has EIG $\mathbf{A} = \mathbf{U}\mathrm{diag}(\lambda_i)\mathbf{U}^T$, where $\mathbf{U}$ is orthogonal and $\mathrm{diag}(\lambda_i)$ is a diagonal matrix of eigenvalues λ_i of $\mathbf{A}$. Then $\mathbf{A}$ has a square root $\mathbf{Y} = \mathbf{U}\mathrm{diag}(\lambda_i^{1/2})\mathbf{U}^T$, i.e., $\mathbf{Y}^2 = \mathbf{A}$. Unfortunately, both EIG and SVD are not well supported on GPU.

Newton-Schulz Iteration Higham [9] studied a class of methods for iteratively computing matrix square root. These methods, termed as Newton-Padé iterations, are developed based on the connection between matrix sign function and matrix square root, together with rational Padé approximation. Specifically, for computing the square root $\mathbf{Y}$ of $\mathbf{A}$, given $\mathbf{Y}_0 = \mathbf{A}$ and $\mathbf{Z}_0 = \mathbf{I}$, for $k = 1, \cdots, N$, the coupled iteration takes the following form [9, Chap. 6.7]:

$$\begin{aligned}\mathbf{Y}_k &= \mathbf{Y}_{k-1}p_{lm}(\mathbf{Z}_{k-1}\mathbf{Y}_{k-1})q_{lm}(\mathbf{Z}_{k-1}\mathbf{Y}_{k-1})^{-1}\\ \mathbf{Z}_k &= p_{lm}(\mathbf{Z}_{k-1}\mathbf{Y}_{k-1})q_{lm}(\mathbf{Z}_{k-1}\mathbf{Y}_{k-1})^{-1}\mathbf{Z}_{k-1}, \end{aligned} \tag{1}$$

where p_{lm} and q_{lm} are polynomials, and l and m are non-negative integers. Eqn. (1) converges only locally: if $\|\mathbf{A} - \mathbf{I}\| < 1$ where $\|\cdot\|$ denotes any induced (or consistent) matrix norm, $\mathbf{Y}_k$ and $\mathbf{Z}_k$ quadratically converge to $\mathbf{Y}$ and $\mathbf{Y}^{-1}$, respectively. The family of coupled iteration is stable in that small errors in the previous iteration will not be amplified. The case of $l = 0, m = 1$ called *Newton-Schulz iteration* fits for our purpose as no GPU unfriendly matrix inverse is involved:

$$\begin{aligned}\mathbf{Y}_k &= \frac{1}{2}\mathbf{Y}_{k-1}(3\mathbf{I} - \mathbf{Z}_{k-1}\mathbf{Y}_{k-1})\\ \mathbf{Z}_k &= \frac{1}{2}(3\mathbf{I} - \mathbf{Z}_{k-1}\mathbf{Y}_{k-1})\mathbf{Z}_{k-1}. \end{aligned} \tag{2}$$

Clearly Eqn. (2) involves only matrix product, suitable for parallel implementation on GPU. Compared to *accurate* square root computed by EIG, one can only obtain *approximate* solution with a small number of iterations. We determine the number of iterations N by cross-validation. Interestingly, compared to EIG or SVD based methods, experiments on large-scale ImageNet show that we can obtain matching or marginally better performance under AlexNet architecture (Sec. 4.2) and better performance under ResNet architecture (Sec. 4.3), using no more than 5 iterations.

Pre-normalization and Post-compensation As Newton-Schulz iteration only converges locally, we pre-normalize $\mathbf{\Sigma}$ by trace or Frobenius norm, i.e.,

$$\mathbf{A} = \frac{1}{\mathrm{tr}(\mathbf{\Sigma})}\mathbf{\Sigma} \ \text{ or } \ \frac{1}{\|\mathbf{\Sigma}\|_F}\mathbf{\Sigma}. \tag{3}$$

Let λ_i be eigenvalues of $\mathbf{\Sigma}$, arranged in nondecreasing order. As $\mathrm{tr}(\mathbf{\Sigma}) = \sum_i \lambda_i$ and $\|\mathbf{\Sigma}\|_F = \sqrt{\sum_i \lambda_i^2}$, it is easy to see that $\|\mathbf{\Sigma} - \mathbf{I}\|_2$, which equals to the largest singular value of $\mathbf{\Sigma} - \mathbf{I}$, is $1 - \frac{\lambda_1}{\sum_i \lambda_i}$ and $1 - \frac{\lambda_1}{\sqrt{\sum_i \lambda_i^2}}$ for the case of trace and Frobenius norm, respectively, both less than 1. Hence, the convergence condition is satisfied.

The above pre-normalization of covariance matrix nontrivially changes the data magnitudes such that it produces adverse effect on network. Hence, to counteract this change, after the Newton-Schulz iteration, we accordingly perform post-compensation, i.e.,

$$\mathbf{C} = \sqrt{\mathrm{tr}(\mathbf{\Sigma})}\mathbf{Y}_N \ \text{ or } \ \mathbf{C} = \sqrt{\|\mathbf{\Sigma}\|_F}\mathbf{Y}_N. \tag{4}$$

An alternative scheme to counterbalance the influence incurred by pre-normalization is Batch Normalization (BN) [11]. One may even consider without using any post-compensation. However, our experiment on ImageNet has shown that, without post-normalization, prevalent ResNet [8] fails to converge, while our scheme outperforms BN by about 1% (see 4.3 for details).

3.3. Backward Propagation (BP)

The gradients associated with the structured layers are derived using matrix backpropagation methodology [13], which establishes the chain rule of a general matrix function by first-order Taylor approximation. Below we take *pre-normalization by trace* as an example, deriving the corresponding gradients.

BP of Post-compensation Given $\frac{\partial l}{\partial \mathbf{C}}$ where l is the loss function, the chain rule is of the form $\mathrm{tr}\big((\frac{\partial l}{\partial \mathbf{C}})^T \mathrm{d}\mathbf{C}\big) = \mathrm{tr}\big((\frac{\partial l}{\partial \mathbf{Y}_N})^T \mathrm{d}\mathbf{Y}_N + (\frac{\partial l}{\partial \mathbf{\Sigma}})^T \mathrm{d}\mathbf{\Sigma}\big)$, where $\mathrm{d}\mathbf{C}$ denotes variation of $\mathbf{C}$. After some manipulations, we have

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{Y}_N} &= \sqrt{\mathrm{tr}(\mathbf{\Sigma})}\frac{\partial l}{\partial \mathbf{C}} \\ \frac{\partial l}{\partial \mathbf{\Sigma}}\Big|_{\text{post}} &= \frac{1}{2\sqrt{\mathrm{tr}(\mathbf{\Sigma})}}\mathrm{tr}\Big(\Big(\frac{\partial l}{\partial \mathbf{C}}\Big)^T \mathbf{Y}_N\Big)\mathbf{I}.\end{aligned} \tag{5}$$

BP of Newton-Schulz Iteration Then we are to compute the partial derivatives of the loss function with respect to $\frac{\partial l}{\partial \mathbf{Y}_k}$ and $\frac{\partial l}{\partial \mathbf{Z}_k}$, $k = N-1, \ldots, 1$, given $\frac{\partial l}{\partial \mathbf{Y}_N}$ computed by Eqn. (5) and $\frac{\partial l}{\partial \mathbf{Z}_N} = 0$. As the covariance matrix $\mathbf{\Sigma}$ is symmetric, it is easy to see from Eqn. (2) that $\mathbf{Y}_k$ and $\mathbf{Z}_k$ are both symmetric. According to the chain rules (omitted hereafter for simplicity) of matrix backpropagation and after some manipulations, $k = N, \ldots, 2$, we can derive

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{Y}_{k-1}} =& \frac{1}{2}\Big(\frac{\partial l}{\partial \mathbf{Y}_k}\Big(3\mathbf{I} - \mathbf{Y}_{k-1}\mathbf{Z}_{k-1}\Big) - \mathbf{Z}_{k-1}\frac{\partial l}{\partial \mathbf{Z}_k}\mathbf{Z}_{k-1} \\ & - \mathbf{Z}_{k-1}\mathbf{Y}_{k-1}\frac{\partial l}{\partial \mathbf{Y}_k}\Big) \\ \frac{\partial l}{\partial \mathbf{Z}_{k-1}} =& \frac{1}{2}\Big(\Big(3\mathbf{I} - \mathbf{Y}_{k-1}\mathbf{Z}_{k-1}\Big)\frac{\partial l}{\partial \mathbf{Z}_k} - \mathbf{Y}_{k-1}\frac{\partial l}{\partial \mathbf{Y}_k}\mathbf{Y}_{k-1} \\ & - \frac{\partial l}{\partial \mathbf{Z}_k}\mathbf{Z}_{k-1}\mathbf{Y}_{k-1}\Big).\end{aligned} \tag{6}$$

The final step of this layer is concerned with the partial derivative with respect to $\frac{\partial l}{\partial \mathbf{A}}$, which is given by

$$\frac{\partial l}{\partial \mathbf{A}} = \frac{1}{2}\Big(\frac{\partial l}{\partial \mathbf{Y}_1}\Big(3\mathbf{I} - \mathbf{A}\Big) - \frac{\partial l}{\partial \mathbf{Z}_1} - \mathbf{A}\frac{\partial l}{\partial \mathbf{Y}_1}\Big). \tag{7}$$

BP of Pre-normalization Note that here we need to combine the gradient of the loss function l with respect to $\mathbf{\Sigma}$, backpropagated from the post-compensation layer. As such, by referring to Eqn. (3), we make similar derivations as before and obtain

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{\Sigma}} =& -\frac{1}{(\mathrm{tr}(\mathbf{\Sigma}))^2}\mathrm{tr}\Big(\Big(\frac{\partial l}{\partial \mathbf{A}}\Big)^T \mathbf{\Sigma}\Big)\mathbf{I} + \frac{1}{\mathrm{tr}(\mathbf{\Sigma})}\frac{\partial l}{\partial \mathbf{A}} \\ & + \frac{\partial l}{\partial \mathbf{\Sigma}}\Big|_{\text{post}}.\end{aligned} \tag{8}$$

If we adopt *pre-normalization by Frobenius norm*, the gradients associated with post-compensation become

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{Y}_N} &= \sqrt{\|\mathbf{\Sigma}\|_F}\frac{\partial l}{\partial \mathbf{C}} \\ \frac{\partial l}{\partial \mathbf{\Sigma}}\Big|_{\text{post}} &= \frac{1}{2\|\mathbf{\Sigma}\|_F^{3/2}}\mathrm{tr}\Big(\Big(\frac{\partial l}{\partial \mathbf{C}}\Big)^T \mathbf{Y}_N\Big)\mathbf{\Sigma},\end{aligned} \tag{9}$$

and that with respect to pre-normalization is

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{\Sigma}} =& -\frac{1}{\|\mathbf{\Sigma}\|_F^3}\mathrm{tr}\Big(\Big(\frac{\partial l}{\partial \mathbf{A}}\Big)^T \mathbf{\Sigma}\Big)\mathbf{\Sigma} + \frac{1}{\|\mathbf{\Sigma}\|_F}\frac{\partial l}{\partial \mathbf{A}} \\ & + \frac{\partial l}{\partial \mathbf{\Sigma}}\Big|_{\text{post}},\end{aligned} \tag{10}$$

while the backward gradients of Newton-Schulz iteration (6) keep unchanged.

Finally, given $\frac{\partial l}{\partial \mathbf{\Sigma}}$, one can derive the gradient of the loss function l with respect to input matrix $\mathbf{X}$, which takes the following form [21]:

$$\frac{\partial l}{\partial \mathbf{X}} = \bar{\mathbf{I}}\mathbf{X}\Big(\frac{\partial l}{\partial \mathbf{\Sigma}} + \Big(\frac{\partial l}{\partial \mathbf{\Sigma}}\Big)^T\Big). \tag{11}$$

4. Experiments

We evaluate the proposed method on both large-scale image classification and challenging fine-grained visual categorization tasks. We make experiments using two PCs each of which is equipped with a 4-core Intel i7-4790k@4.0GHz CPU, 32G RAM, 512GB Samsung PRO SSD and two Titan Xp GPUs. We implement our networks using MatConvNet [30] and Matlab2015b, under Ubuntu 14.04.5 LTS.

4.1. Datasets and Our Meta-layer Implementation

Datasets For large-scale image classification, we adopt *ImageNet LSVRC2012 dataset* [5] with 1,000 object categories. The dataset contains 1.28M images for training, 50K images for validation and 100K images for testing (without published labels). As in [11, 8], we report the results on the validation set. For fine-grained categorization, we use three popular *fine-grained benchmarks*, i.e., CUB-200-2011(Birds) [31], FGVC-aircraft (Aircrafts) [27] and Stanford cars (Cars) [17]. The Birds dataset contains 11,788 images from 200 species, with large intra-class variation but small inter-class variation. The Aircrafts dataset includes 100 aircraft classes and a total of 10,000 images with small background noise but higher inter-class similarity. The Cars dataset consists of 16,185 images from 196 classes. For all datasets, we adopt the provided training/test split, using neither bounding boxes nor part annotations.

Implementation of iSQRT-COV Meta-layer We encapsulate our code in three computational blocks, which implement forward&backward computation of pre-normalization layer, Newton-Schulz iteration layer and post-compensation layer, respectively. The code is written in C++ based on NVIDIA cuBLAS on top of CUDA toolkit 8.0. In addition, we write code in C++ based on cuBLAS for computing covariance matrices. We create MEX files so that the above subroutines can be called in Matlab environment. For AlexNet, we insert our meta-layer after the last convolution layer (with ReLU), which outputs an $13 \times 13 \times 256$ tensor. For ResNet architecture, as suggested [21], we do not perform downsampling for the last set of convolutional blocks, and add one 1×1 convolution with $d = 256$ channels after the last sum layer (with ReLU). The added 1×1 convolution layer outputs an $14 \times 14 \times 256$ tensor. Hence, with both architectures, the covariance matrix $\mathbf{\Sigma}$ is of size 256×256 and our meta-layer outputs an $d(d + 1)/2 \approx$ 32K-dimensional vector as the image representation.

4.2. Evaluation with AlexNet on ImageNet

In the first part of experiments, we analyze, with AlexNet architecture, the design choices of our iSQRT-COV method, including the number of Newton-Schulz iterations, time and memory usage, and behaviors of different pre-normalization methods. We select AlexNet because it runs faster with shallower depth, and the results can extrapolate to deeper networks which mostly follow its architecture design.

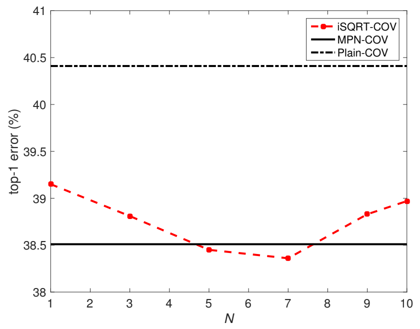


Figure 2. Impact of number N of Newton-Schulz iterations on iSQRT-COV with AlexNet architecture on ImageNet.

We follow [21] for color augmentation and weight initialization, adopting BN and no dropout. We use SGD with a mini-batch of 128, unless otherwise stated. The momentum is 0.9 and weight decay is 0.0005. We train iSQRT-COV networks from scratch in 20 epochs where learning rate follows exponential decay $10^{-1.1} \rightarrow 10^{-5}$. All training and test images are uniformly resized with shorter sides of 256. During training we randomly crop a 224×224 patch from each image or its horizontal flip. We make inference on one single 224×224 center crop from a test image.

Impact of Number N of Newton-Schulz Iterations Fig. 2 shows top-1 error rate as a function of number of Newton-Schulz iterations in Eqn. (2). Plain-COV indicates simple covariance pooling without any normalization. With one single iteration, our method outperforms Plain-COV by 1.3%. As iteration number grows, the error rate of iSQRT-COV gradually declines. With 3 iterations, iSQRT-COV is comparable to MPN-COV, having only 0.3% higher error rate, while performing marginally better than MPN-COV between 5 and 7 iterations. After $N = 7$, the error rate consistently increases, indicating growth of iteration number is not helpful for improving accuracy. As larger N incurs higher computational cost, to balance efficiency and accuracy, we set N to 5 in the remaining experiments. Notably, the approximate square root normalization improves a little over the accurate one obtained via EIG. This interesting problem will be discussed in Sec. 4.3, where iSQRT-COV is further evaluated on substantially deeper ResNets.

Time and Memory Analysis We compare time and memory consumed by single meta-layer of different methods. We use public code for MPN-COV, G^{2}DeNet and improved B-CNN released by the respective authors. As shown in Tab. 2(a), iSQRT-COV ($N = 3$) and iSQRT-COV ($N = 5$) are 3.1x faster and 1.8x faster than MPN-COV, respectively. Furthermore, iSQRT-COV ($N = 5$) is five times more efficient than improved B-CNN and

(a) Time of FP+BP (ms) taken and memory (MB) used by single meta-layer. Numbers in parentheses indicate FP time.

Method		Language	bottleneck	Time	Memory
iSQRT-COV (N=3)		C++	N/A	**0.81 (0.26)**	0.627
iSQRT-COV (N=5)				**1.41 (0.41)**	**1.129**
MPN-COV [21]		C++&M	EIG	2.58 (2.41)	0.377
Impro. B-CNN [24]	FP and BP based on SVD	M	SVD or EIG	13.51 (11.19)	0.501
	FP by NS Iter., BP by Lyap.			13.91 (2.09)	
G^{2}DeNet [32]		M	SVD	8.56 (4.76)	0.505

(b) Time (ms) taken by matrix decomposition (single precision arithmetic)

Algorithm	CUDA cuSOLVER	Matlab (CPU function)	Matlab (GPU function)
EIG	21.3	1.8	9.8
SVD	52.2	4.1	11.9

Table 2. Comparison of time and memory usage with AlexNet architecture. The size of covariance matrix is 256 $\times$ 256.

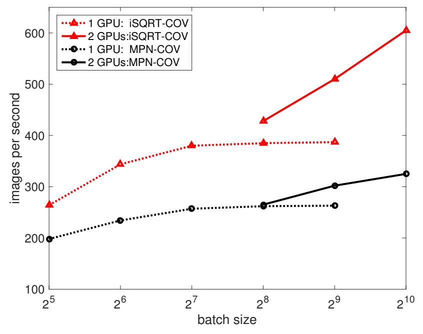


Figure 3. Images per second (FP+BP) of network training with AlexNet architecture.

G^{2}DeNet. For improved B-CNN, the forward computation of Newton-Schulz (NS) iteration is much faster than that of SVD, but the total time of two methods is comparable. The authors of improved B-CNN also proposed two other implementations, i.e., FP by NS iteration plus BP by SVD and FP by SVD plus BP by Lyapunov (Lyap.), which take 15.31 (2.09) and 12.21 (11.19), respectively. We observe that, in any case, the forward+backward time taken by single meta-layer of improved B-CNN is significant as GPU unfriendly SVD or EIG cannot be avoided, even though the forward computation is very efficient when NS iteration is used. Tab. 2(b) presents running time of EIG and SVD of an 256 $\times$ 256 covariance matrix. Matlab (M) built-in CPU functions and GPU functions deliver over 10x and 2.1x speedups over their CUDA counterparts, respectively. Our method needs to store $\mathbf{Y}_k$ and $\mathbf{Z}_k$ in Eqn. (2) which will be used in backpropagation, taking up more memory than EIG or SVD based ones. Among all, our iSQRT-COV ($N = 5$) takes up the largest memory of 1.129MB, which is insignificant compared to 12GB memory on a Titan Xp. Note that for network inference only, our method takes 0.125MB memory as it is unnecessary to store $\mathbf{Y}_k$ and $\mathbf{Z}_k$.

Method	Top-1 Error	Top-5 Error	Time
AlexNet [18]	41.8	19.2	1.32 (0.77)
MPN-COV [21]	38.51	17.60	3.89 (2.59)
B-CNN [25]	39.89	18.32	1.92 (0.83)
DeepO$_2$P [12]	42.16	19.62	11.23 (7.04)
Impro. B-CNN*[24]	40.75	18.91	15.48 (13.04)
G^{2}DeNet [32]	38.71	17.66	9.86 (5.88)
iSQRT-COV(Frob.)	38.78	17.67	2.56 (0.81)
iSQRT-COV(trace)	**38.45**	**17.52**	2.55 (0.81)

Table 3. Error rate (%) and time of FP+BP (ms) per image of different covariance pooling methods with AlexNet on ImageNet. Numbers in parentheses indicate FP time. *Following [24], improved B-CNN successively performs matrix square root, element-wise square root and ℓ_2 normalizations.

Next, we compare in Fig. 3 speed of network training between MPN-COV and iSQRT-COV with both one-GPU and two-GPU configurations. For one-GPU configuration, the speed gap vs. batch size between the two methods keeps nearly constant. For two-GPU configuration, their speed gap becomes more significant when batch size gets larger. As can be seen, the speed of iSQRT-COV network continuously grows with increase of batch size while that of MPN-COV tends to saturate when batch size is larger than 512. Clearly our iSQRT-COV network can make better use of computing power of multiple GPUs than MPN-COV.

Pre-normalization by Trace vs. by Frobenius Norm Sec. 3 describes two pre-normalization methods. Here we compare them in Tab. 3 (bottom rows), where iSQRT-COV (trace) indicates pre-normalization by trace. We can see that pre-normalization by trace produces 0.3% lower error rate than that by Frobenius norm, while taking similar time with the latter. Hence, in all the remaining experiments, we adopt trace based pre-normalization method.

Comparison with Other Covariance Pooling Methods We compare iSQRT-COV with other covariance pooling methods, as shown in Tab. 3. The results of MPN-COV, B-CNN and DeepO$_2$P are duplicated from [21]. We train from scratch G^{2}DeNet and improved B-CNN on ImageNet. We use the most efficient implementation of improved B-CNN, i.e., FP by SVD and BP by Lyap., and we mention all implementations of improved B-CNN produce similar results. Our iSQRT-COV using pre-normalization by trace is marginally better than MPN-COV. All matrix square root normalization methods except improved B-CNN outperform B-CNN and DeepO$_2$P. Since improved B-CNN is identical to MPN-COV if element-wise square root normalization and ℓ_2-normalization are neglected, its unsatisfactory performance suggests that, after matrix square root normalization, further element-wise square root normalization and ℓ_2-normalization hurt large-scale ImageNet classifica-

Pre-normalization	Post-compensation	Top-1 Err.	Top-5 Err.
Trace	w/o	N/A	N/A
	w/ BN [11]	23.12	6.60
	w/ Trace	**22.14**	**6.22**

Table 4. Impact of post-compensation on iSQRT-COV with ResNet-50 architecture on ImageNet.

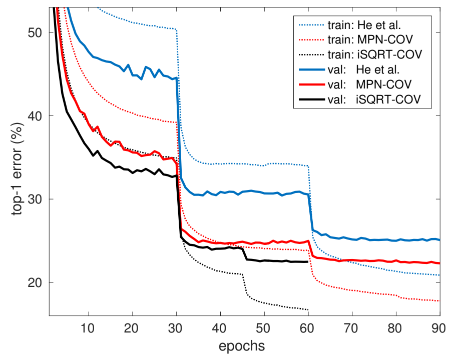


Figure 4. Convergence curves of different networks trained with ResNet-50 architecture on ImageNet.

tion. This is consistent with the observation in [21, Tab. 1], where after matrix power normalization, additional normalization by Frobenius norm or matrix ℓ_2−norm makes performance decline.

4.3. Results on ImageNet with ResNet Architecture

This section evaluates iSQRT-COV with ResNet architecture [8]. We follow [21] for color augmentation and weight initialization. We rescale each training image with its shorter side randomly sampled on $[256, 512]$ [29]. The fixed-size 224×224 patch is randomly cropped from the rescaled image or its horizontal flip. We rescale each test image with a shorter side of 256 and evaluate a single 224×224 center crop for inference. We use SGD with a mini-batch size of 256, a weight decay of 0.0001 and a momentum of 0.9. We train iSQRT-COV networks from scratch in 60 epochs, initializing the learning rate to $10^{-1.1}$ which is divided by 10 at epoch 30 and 45, respectively.

Significance of Post-compensation Rather than our post-compensation scheme, one may choose Batch Normalization (BN) [11] or simply do nothing (i.e., without post-compensation). Tab. 4 summarizes impact of different schemes on iSQRT-COV network with ResNet-50 architecture. Without post-compensation, iSQRT-COV network fails to converge. Careful observations show that in this case the gradients are very small (on the order of 10^{-5}), and largely tuning of learning rate helps little. Option of BN helps the network converge, but producing about 1% higher top-1 error rate than our post-compensation scheme. The comparison above suggests that our post-compensation scheme is essential for achieving state-of-the-art results.

Method	Model	Top-1 Err.	Top-5 Err.
He et al. [8]	ResNet-50	24.7	7.8
FBN [23]		24.0	7.1
SORT [35]		23.82	6.72
MPN-COV [21]		22.73	6.54
iSQRT-COV		**22.14**	**6.22**
He et al. [8]	ResNet-101	23.6	7.1
iSQRT-COV		**21.21**	**5.68**
He et al. [8]	ResNet-152	23.0	6.7

Table 5. Error (%) comparison of second-order networks with first-order ones on ImageNet.

Fast Convergence of iSQRT-COV Network We compare convergence of iSQRT-COV and MPN-COV with ResNet-50 architecture, as well as the original ResNet-50 [8] in which global average pooling is performed after the last convolution layer. Fig. 4 presents the convergence curves. Compared to the original ResNet-50, the convergence of both iSQRT-COV and MPN-COV is significantly faster. We observe that iSQRT-COV can converge well within 60 epochs, achieving top-1 error rate of 22.14%, $\sim$0.6% lower than MPN-COV. We also trained iSQRT-COV with 90 epochs using same setting with MPN-COV, obtaining top-5 error of 6.12%, slightly lower than that with 60 epochs (6.22%). This indicates iSQRT-COV can converge in less epochs, so further accelerating training, as opposed to MPN-COV. The fast convergence property of iSQRT-COV is appealing. As far as we know, previous networks with ResNet-50 architecture require at least 90 epochs to converge to competitive results.

Comparison with State-of-the-arts In Tab. 5, we compare our method with other second-order networks, as well as the original ResNets. With ResNet-50 architecture, all the second-order networks improve over the first-order one while our method performing best. MPN-COV and iSQRT-COV, both of which involve square root normalization, are superior to FBN [23] which uses no normalization and SORT [35] which introduces dot product transform in the linear sum of two-branch module followed by element-wise normalization. Moreover, our iSQRT-COV outperforms MPN-COV by 0.6% in top-1 error. Note that our 50-layer iSQRT-COV network achieves lower error rate than much deeper ResNet-101 and ResNet-152, while our 101-layer iSQRT-COV network outperforming the original ResNet-101 by 2.4% and ResNet-152 by 1.8%, respectively.

Why Approximate Square Root Performs Better Fig. 2 shows that more iterations which lead to more accurate square root is not helpful for iSQRT-COV with AlexNet. From Tab. 5, we observe that iSQRT-COV with ResNet computing approximate square root performs better than MPN-COV which can obtain exact square root by EIG. Recall that, for covariance pooling ConvNets, we face the

Method	d	Dim.	Top-1 Err.	Top-5 Err.	Time
He et al. [8]	N/A	2K	24.7	7.8	8.08 (1.93)
iSQRT-COV	64	2K	23.73	6.99	9.86 (2.39)
	128	8K	22.78	6.43	10.75 (2.67)
	256	32K	22.14	6.22	11.33 (2.89)

Table 6. Error rate (%) and time of FP+BP (ms) per image vs. d (or representation dimension) of compact iSQRT-COV with ResNet-50 on ImageNet. Numbers in parentheses indicate FP time.

problem of small sample of large dimensionality, and matrix square root is consistent with general shrinkage principle of robust covariance estimation [21]. Hence, we conjuncture that approximate matrix square root may be a better robust covariance estimator than the exact square root. Despite this analysis, we think this problem is worth future research.

Compactness of iSQRT-COV Our iSQRT-COV outputs 32k-dimensional representation which is high. Here we consider to compress this representation. Compactness by PCA [25] is not viable since obtaining the principal components on ImageNet is too expensive. CBP [6] is not applicable to our iSQRT-COV as well, as it does not *explicitly* estimate the covariance matrix. We propose a simple scheme, which decreases the dimension (dim.) of covariance representation by lowering the number d of channels of 1×1 convolutional layer before our covariance pooling. Tab. 6 summarizes results of compact iSQRT-COV. The recognition error increases slightly ($\uparrow$ 0.64%) when d decreases from 256 to 128 (correspondingly, dim. of image representation 32K $\rightarrow$ 8K). The error rate is 23.73% if the dimension is compressed to 2K, still outperforming the original ResNet-50 which performs global average pooling.

4.4. Fine-grained Visual Categorization (FGVC)

Finally, we apply iSQRT-COV models pre-trained on ImageNet to FGVC. For fair comparison, we follow [25] for experimental setting and evaluation protocol. On all datasets, we crop 448×448 patches as input images. We replace 1000-way softmax layer of a pre-trained iSQRT-COV model by a k-way softmax layer, where k is number of classes in the fine-grained dataset, and finetune the network using SGD with momentum of 0.9 for 50$\sim$100 epochs with a small learning rate (lr=$10^{-2.1}$) for all layers except the fully-connected layer, which is set to $5 \times lr$. We use horizontal flipping as data augmentation. After finetuning, the outputs of iSQRT-COV layer are ℓ_2-normalized before inputted to train k one-vs-all linear SVMs with hyperparameter $C = 1$. We predict the label of a test image by averaging SVM scores of the image and its horizontal flip.

Tab. 7 presents classification results of different methods, where column 3 lists the dimension of the corresponding representation. With ResNet-50 architecture, KP performs much better than CBP, while iSQRT-COV (8K) respectively outperforms KP (14K) by about 2.6%, 3.8% and 0.6% on Birds, Aircrafts and Cars, and iSQRT-COV (32K) further improves accuracy. Note that KP combines first-order up to fourth-order statistics while iSQRT-COV only exploits second-order one. With VGG-D, iSQRT-COV (32k) matches or outperforms state-of-the-art competitors, but inferior to iSQRT-COV (32k) with ResNet-50.

	Method	Dim.	Birds	Aircrafts	Cars
ResNet-50	iSQRT-COV	32K	**88.1**	**90.0**	**92.8**
		8K	87.3	89.5	91.7
	CBP [6]	14K	81.6	81.6	88.6
	KP [4]	14K	84.7	85.7	91.1
VGG-D	iSQRT-COV	32K	87.2	90.0	92.5
	NetVLAD [1]	32K	81.9	81.8	88.6
	CBP [6]	8K	84.3	84.1	91.2
	KP [4]	13K	86.2	86.9	92.4
	LRBP [16]	10K	84.2	87.3	90.9
	Improved B-CNN[24]	262K	85.8	88.5	92.0
	G^{2}DeNet [32]	263K	87.1	89.0	92.5
	HIHCA [2]	9K	85.3	88.3	91.7
iSQRT-COV with ResNet-101		32K	**88.7**	**91.4**	**93.3**

Table 7. Comparison of accuracy (%) on fine-grained benchmarks. Our method uses neither bounding boxes nor part annotations.

On all fine-grained datasets, KP and CBP with 16-layer VGG-D perform better than their counterparts with 50-layer ResNet, despite the fact that ResNet-50 significantly outperforms VGG-D on ImageNet [8]. The reason may be that the last convolution layer of pre-trained ResNet-50 outputs 2048-dimensional features, much higher than 512-dimensional one of VGG-D, which are not suitable for existing second- or higher-order pooling methods. *Different from all existing methods which use models pre-trained on ImageNet with first-order information, our pre-trained models are of second-order.* Using pre-trained iSQRT-COV models with ResNet-50, we achieve recognition results superior to all the compared methods, and furthermore, establish state-of-the-art results on three fine-grained benchmarks using iSQRT-COV model with ResNet-101.

5. Conclusion

We presented an iterative matrix square root normalization of covariance pooling (iSQRT-COV) network which can be trained end-to-end. Compared to existing works depending heavily on GPU unfriendly EIG or SVD, our method, based on coupled Newton-Schulz iteration [9], runs much faster as it involves only matrix multiplications, suitable for parallel implementation on GPU. We validated our method on both large-scale ImageNet dataset and challenging fine-grained benchmarks. Given efficiency and promising performance of our iSQRT-COV, we hope global covariance pooling will be a promising alternative to global average pooling in other deep network architectures, e.g., ResNeXt [36], Inception [11] and DenseNet [10].

References

[1] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *CVPR*, 2016. 1, 8

[2] S. Cai, W. Zuo, and L. Zhang. Higher-order integration of hierarchical convolutional activations for fine-grained visual categorization. In *ICCV*, Oct 2017. 2, 8

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014. 2

[4] Y. Cui, F. Zhou, J. Wang, X. Liu, Y. Lin, and S. Belongie. Kernel pooling for convolutional neural networks. In *CVPR*, July 2017. 2, 8

[5] 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. 1, 5

[6] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell. Compact bilinear pooling. In *CVPR*, June 2016. 2, 8

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *ICCV*, 2015. 1

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 4, 5, 7, 8

[9] N. J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, Philadelphia, PA, USA, 2008. 1, 2, 3, 8

[10] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, July 2017. 8

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 1, 4, 5, 7, 8

[12] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015. 1, 2, 6

[13] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv*, abs/1509.07838, 2015. 4

[14] B. Jansík, S. Høst, P. Jørgensen, J. Olsen, and T. Helgaker. Linear-scaling symmetric square-root decomposition of the overlap matrix. *J. of Chemical Physics*, pages 124104–124104, 2007. 3

[15] K. Kafle and C. Kanan. An analysis of visual question answering algorithms. In *ICCV*, Oct 2017. 1

[16] S. Kong and C. Fowlkes. Low-rank bilinear pooling for fine-grained classification. In *CVPR*, July 2017. 8

[17] J. Krause, M. Stark, J. Deng, and F. F. Li. 3D Object representations for fine-grained categorization. In *ICCV Workshops*, pages 554–561, 2013. 5

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 1, 2, 6

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[20] P. Li, Q. Wang, H. Zeng, and L. Zhang. Local Log-Euclidean multivariate Gaussian descriptor and its application to image classification. *IEEE TPAMI*, 2017. 2

[21] P. Li, J. Xie, Q. Wang, and W. Zuo. Is second-order information helpful for large-scale visual recognition? In *ICCV*, Oct 2017. 1, 2, 4, 5, 6, 7, 8

[22] Y. Li, M. Dixit, and N. Vasconcelos. Deep scene image classification with the MFAFVNet. In *ICCV*, Oct 2017. 1

[23] Y. Li, N. Wang, J. Liu, and X. Hou. Factorized bilinear models for image recognition. In *ICCV*, 2017. 7

[24] T.-Y. Lin and S. Maji. Improved bilinear pooling with CNNs. In *BMVC*, 2017. 1, 2, 6, 8

[25] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *ICCV*, 2015. 1, 2, 6, 8

[26] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, June 2015. 1

[27] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *HAL - INRIA*, 2013. 5

[28] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. In *CVPR*, July 2017. 1

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2, 7

[30] A. Vedaldi and K. Lenc. MatConvNet – convolutional neural networks for MATLAB. In *ACM on Multimedia*, 2015. 5

[31] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds200-2011 Dataset. *California Institute of Technology*, 2011. 5

[32] Q. Wang, P. Li, and L. Zhang. G^{2}DeNet: Global Gaussian distribution embedding network and its application to visual recognition. In *CVPR*, July 2017. 1, 2, 6, 8

[33] 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. 2

[34] Y. Wang, M. Long, J. Wang, and P. S. Yu. Spatiotemporal pyramid network for video action recognition. In *CVPR*, July 2017. 1

[35] Y. Wang, L. Xie, C. Liu, S. Qiao, Y. Zhang, W. Zhang, Q. Tian, and A. Yuille. SORT: Second-order response transform for visual recognition. In *ICCV*, 2017. 7

[36] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *CVPR*, July 2017. 8

[37] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ADE20K dataset. In *CVPR*, 2017. 1

[38] D. Zhou, I. L. Dryden, A. A. Koloydenko, K. M. Audenaert, and L. Bai. Regularisation, interpolation and visualisation of diffusion tensor images using non-Euclidean statistics. *Journal of Applied Statistics*, 43(5):943–978, 2016. 3