Is Second-order Information Helpful for Large-scale Visual Recognition?

Peihua Li[1], Jiangtao Xie[1], Qilong Wang[1], Wangmeng Zuo[2]
[1]Dalian University of Technology, [2]Harbin Institute of Technology
peihuali@dlut.edu.cn,{jiangtaoxie,qlwang}@mail.dlut.edu.cn,wmzuo@hit.edu.cn

Abstract

By stacking layers of convolution and nonlinearity, convolutional networks (ConvNets) effectively learn from low-level to high-level features and discriminative representations. Since the end goal of large-scale recognition is to delineate complex boundaries of thousands of classes, adequate exploration of feature distributions is important for realizing full potentials of ConvNets. However, state-of-the-art works concentrate only on deeper or wider architecture design, while rarely exploring feature statistics higher than first-order. We take a step towards addressing this problem. Our method consists in covariance pooling, instead of the most commonly used first-order pooling, of high-level convolutional features. The main challenges involved are robust covariance estimation given a small sample of large-dimensional features and usage of the manifold structure of covariance matrices. To address these challenges, we present a Matrix Power Normalized Covariance (MPN-COV) method. We develop forward and backward propagation formulas regarding the nonlinear matrix functions such that MPN-COV can be trained end-to-end. In addition, we analyze both qualitatively and quantitatively its advantage over the well-known Log-Euclidean metric. On the ImageNet 2012 validation set, by combining MPN-COV we achieve over 4%, 3% and 2.5% gains for AlexNet, VGG-M and VGG-16, respectively; integration of MPN-COV into 50-layer ResNet outperforms ResNet-101 and is comparable to ResNet-152. The source code will be available on the project page: http://www.peihuali.org/MPN-COV.

1. Introduction

Since outperforming significantly the classical, shallow classification framework, deep convolutional networks (ConvNets) [14] have triggered fast growing interests and achieved great advance in large-scale visual recognition [27, 29, 9]. The ConvNet architecture [17] renders learning of features, representations and classification in an end-to-end manner, superior to the classical Bag of Words (BoW) [16] architecture where these components are separately optimized, independent of each other. The large-scale, labeled ImageNet dataset [24] and high computing capability of GPUs contribute to successful training of increasingly wider and deeper ConvNets.

The ConvNet model, which starts from the raw color images as inputs, learns progressively the low-level, middle-level and high-level features from bottom, intermediate to top convolutional (conv.) layers [33], obtaining discriminative representations connected to fully-connected (FC) layers. The gradient backprogation algorithm enables the classifier to learn decision boundaries delineating thousands of classes in the space of large-dimensional features generally with complex distributions. Hence, for realizing full potentials of ConvNets, it is important to adequately consider feature distributions. However, most ConvNets concentrate only on designing wider or deeper architectures, rarely exploring statistical information higher than first-order. In the traditional classification paradigm where sufficient labeled data are not available, high-order methods combined with ConvNet models pretrained on ImageNet dataset have achieved impressive recognition accuracies [6, 31]. In the small-scale classification scenarios, researchers have studied end-to-end methods, including DeepO^{2}P [11] and B-CNN [22], for exploiting second-order statistics in deep ConvNets [11, 22]. As such, one interesting problem arising naturally is whether statistics higher than first-order is helpful for large-scale visual recognition.

In this paper, we take a step towards addressing this problem. Motivated by [11, 22], we perform covariance pooling of the last convolutional features rather than the commonly used first-order pooling, producing covariance matrices as global image representations. The main challenges involved are robust covariance estimation provided only with a small sample of large-dimensional features and usage of the manifold structure of the covariance matrices. Existing methods can not well address the two problems, producing unsatisfactory improvement in the large-scale setting. DeepO^{2}P adopts Log-Euclidean (Log-E) metric [1] for exploiting geometry of covariance spaces, which

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

however brings side effect on covariance representations. B-CNN performs element-wise normalization, without considering the manifold of covariance matrices. For tackling the challenges, we propose a Matrix Power Normalized Covariance (MPN-COV) method. We show that MPN-COV amounts to robust covariance estimation; it also approximately exploits the geometry of covariance space while circumventing the downside of the well-known Log-E metric [1]. As MPN-COV involves nonlinear matrix functions whose backpropagation is not straightforward, we develop the gradients associated with MPN-COV based on the matrix propagation methodology [12] for end-to-end learning.

Our main contributions are summarized as follows. Firstly, we are among the first who attempt to exploit statistics higher than first-order for improving the large-scale classification. We propose matrix power normalized covariance method for more discriminative representations, and develop the forward and backward propagation formulas for the nonlinear matrix functions, achieving end-to-end MPN-COV networks. Secondly, we provide interpretations of MPN-COV from statistical, geometric and computational points of view, explaining the underlying mechanism that MPN-COV can address the aforementioned challenges. Thirdly, on the ImageNet 2012 dataset, we thoroughly evaluate MPN-COV for validating our mathematical derivation and understandings, obtaining competitive improvements over its first-order counterparts under a variety of ConvNet architectures.

2. Related Work

The statistics higher than first-order has been successfully used in both classical and deep learning based classification scenarios. In the area of low-level patch descriptors, local Gaussian descriptors has demonstrated better performance than descriptors exploiting zeroth- or first-order statistics [20]. Fisher Vector (FV) makes use of the first- and second-order statistics, reporting state-of-the-art results based on hand-crafted features [25]. The locality-constrained affine subspace coding [19] proposed to use Fisher Information matrix for improving classification performance. By adopting features computed from pretrained ConvNets, FV considerably improves recognition accuracy over using hand-crafted features on small-scale datasets [6]. Wang et al. [31] present global Gaussian distributions as image representations for material recognition using the convolutional features from pretrained ConvNets. In [6, 31], feature design, image representation and classifier training are not jointly optimized. Different from them, we propose end-to-end deep learning to exploit the second-order statistics for improving large-scale visual recognition.

In image classification the second-order pooling known as O^2P is proposed in [2]. The O^2P computes non-central, second-order moments which is subject to matrix logarithm

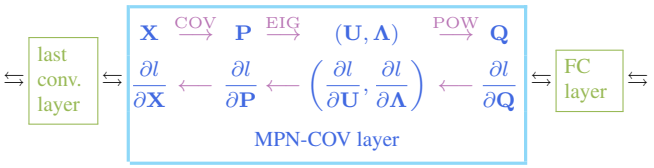


Figure 1. Illustration of forward and backward propagations of ConvNets with MPN-COV. The proposed MPN-COV as a layer is inserted between the last conv. layer and FC layer, trainable end-to-end. See text for notations and in-depth description.

for representing free-form regions. In the context of classical image classification, Koniusz et al. [13] propose second- and third-order pooling of hand-crafted features or their coding vectors. For the goal of counteracting correlated burstiness due to non-i.i.d. data, they apply power normalization of eigenvalues (ePN) to autocorrelation matrices or to the core tensors [15] of the autocorrelation tensors. In [5], Higher-order Kernel (HoK) descriptor is proposed for action recognition in videos. HoK concerns pooling of higher-order tensors of probability scores from pretrained ConvNets in video frames, which are subject to ePN and then fed to SVM classifiers. Our main differences from [13, 5] are (1) we develop an end-to-end MPN-COV method in deep ConvNet architecture, and verify that statistics higher than first-order is helpful for large-scale recognition; (2) we provide statistical, geometric and computational interpretations, explaining the mechanism underlying matrix power normalization.

Ionescu et al. [11] present the theory of matrix backprogation which makes possible inclusion of structured, global layers into deep ConvNets. Furthermore, they propose DeepO^{2}P for end-to-end, second-order pooling in deep ConvNets by Singular Value Decomposition (SVD). B-CNN [22] aggregates the outer products of convolutional features from two networks. The resulting matrices undergo element-wise power normalization. Note that B-CNN produces second-order, non-central moments when the two ConvNets involved share the same configuration. Our MPN-COV is similar to DeepO^{2}P and B-CNN but having clear distinctions. We show that matrix power normalization plays a key role for the second-order pooling to achieve competitive performance, instead of matrix logarithm [1] or element-wise power normalization [25]. Moreover, we provide the rationale why MPN-COV well addresses the two challenges of the covariance pooling. Finally, DeepO^{2}P and B-CNN have not been evaluated on challenging, large-scale ImageNet dataset.

3. The Proposed MPN-COV

For an input image, MPN-COV produces a normalized covariance matrix as a representation, which characterizes the correlations of feature channels and actually designates the shape of feature distribution. Fig. 1 illustrates the for-

ward and backward propagations of MPN-COV. Given the responses $\mathbf{X}$ of the last conv. layer as features, we first compute the sample covariance matrix $\mathbf{P}$ of $\mathbf{X}$. Then we perform eigenvalue decomposition (EIG) of $\mathbf{P}$ to obtain the orthogonal matrix $\mathbf{U}$ and diagonal matrix $\mathbf{\Lambda}$, through which the matrix power $\mathbf{Q} \triangleq \mathbf{P}^\alpha$ can be transformed to the power of eigenvalues of $\mathbf{P}$. Finally, $\mathbf{Q}$ will be inputted to the subsequent, top FC layer. Accordingly, in backward pass, given the partial derivative $\frac{\partial l}{\partial \mathbf{Q}}$ of loss function l with respect to $\mathbf{Q}$ propagated from the top FC layer, we need to compute in reverse order the associated partial derivatives.

3.1. Forward Propagation

Let $\mathbf{X} \in \mathbb{R}^{d \times N}$ be a matrix whose columns consist of a sample of N features of dimension d. The sample covariance matrix $\mathbf{P}$ of $\mathbf{X}$ is computed as

$$\mathbf{X} \mapsto \mathbf{P}, \quad \mathbf{P} = \mathbf{X}\bar{\mathbf{I}}\mathbf{X}^T. \tag{1}$$

Here $\bar{\mathbf{I}} = \frac{1}{N}(\mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T)$, where $\mathbf{I}$ is the $N \times N$ identity matrix, $\mathbf{1} = [1, \ldots, 1]^T$ is a $N-$dimensional vector, and T denotes the matrix transpose. The sample covariance matrix $\mathbf{P}$ is symmetric positive semi-definite, which has eigenvalue decomposition as follows:

$$\mathbf{P} \mapsto (\mathbf{U}, \mathbf{\Lambda}), \quad \mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T, \tag{2}$$

where $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$ is a diagonal matrix and $\lambda_i, i = 1, \ldots, d$ are eigenvalues arranged in non-increasing order; $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$ is an orthogonal matrix whose column $\mathbf{u}_i$ is the eigenvector corresponding to λ_i. Through EIG we can convert matrix power to the power of eigenvalues. Hence, we have

$$(\mathbf{U}, \mathbf{\Lambda}) \mapsto \mathbf{Q}, \quad \mathbf{Q} \triangleq \mathbf{P}^\alpha = \mathbf{U}\mathbf{F}(\mathbf{\Lambda})\mathbf{U}^T. \tag{3}$$

Here α is a positive real number and $\mathbf{F}(\mathbf{\Lambda}) = \mathrm{diag}(f(\lambda_1), \ldots, f(\lambda_d))$, where $f(\lambda_i)$ is the power of eigenvalues

$$f(\lambda_i) = \lambda_i^\alpha, \quad \text{for MPN.} \tag{4}$$

Inspired by the element-wise power normalization technique [25], we can further perform, right after MPN, normalization by matrix ℓ_2-norm (M-ℓ_2) or by matrix Frobenius norm (M-Fro). The matrix ℓ_2-norm (also known as the *spectral norm*) of a matrix $\mathbf{P}$, denoted by $\|\mathbf{P}\|_2$, is defined as the largest singular value of $\mathbf{P}$, which equals the largest eigenvalue if $\mathbf{P}$ is a covariance matrix. The matrix Frobenius norm of $\mathbf{P}$ can be defined in various ways such as $\|\mathbf{P}\|_F = (\mathrm{tr}(\mathbf{P}^T\mathbf{P}))^{\frac{1}{2}} = (\sum_i \lambda_i^2)^{\frac{1}{2}}$, where λ_i are singular values of $\mathbf{P}$. As such, we have

$$f(\lambda_i) = \begin{cases} \lambda_i^\alpha \big/ \lambda_1^\alpha & \text{for MPN+M-}\ell_2 \\ \lambda_i^\alpha \big/ (\sum_k \lambda_k^{2\alpha})^{\frac{1}{2}} & \text{for MPN+M-Fro} \end{cases} \tag{5}$$

Note that, in (5), when $\alpha = 1$ the first and second identities reduce to separate M-ℓ_2 and M-Fro normalizations, respectively.

3.2. Backward Propagation

We use the methodology of matrix backpropagation, formulated in [11, 12], to compute the partial derivative of loss function l with respect to the input matrix of some layer. It is built on the theory of matrix calculus, enabling inclusion of structured, nonlinear matrix functions in neural networks while considering the invariants involved such as orthogonality, diagonality and symmetry.

Let us consider derivation of $\frac{\partial l}{\partial \mathbf{U}}$ and $\frac{\partial l}{\partial \mathbf{\Lambda}}$, given $\frac{\partial l}{\partial \mathbf{Q}}$ propagated from the top FC layer. The expression of the chain rule is

$$\mathrm{tr}\Big(\Big(\frac{\partial l}{\partial \mathbf{U}}\Big)^T \mathrm{d}\mathbf{U} + \Big(\frac{\partial l}{\partial \mathbf{\Lambda}}\Big)^T \mathrm{d}\mathbf{\Lambda}\Big) = \mathrm{tr}\Big(\Big(\frac{\partial l}{\partial \mathbf{Q}}\Big)^T \mathrm{d}\mathbf{Q}\Big), \tag{6}$$

where $\mathrm{d}\mathbf{Q}$ denotes variation of matrix $\mathbf{Q}$. From Eq. (3), we have $\mathrm{d}\mathbf{Q} = \mathrm{d}\mathbf{U}\mathbf{F}\mathbf{U}^T + \mathbf{U}\mathrm{d}\mathbf{F}\mathbf{U}^T + \mathbf{U}\mathbf{F}\mathrm{d}\mathbf{U}^T$. We note that $\mathrm{d}\mathbf{F} = \mathrm{diag}\big(\alpha\lambda_1^{\alpha-1}, \ldots, \alpha\lambda_d^{\alpha-1}\big)\mathrm{d}\mathbf{\Lambda}$. After some arrangements, we obtain

$$\begin{aligned} \frac{\partial l}{\partial \mathbf{U}} &= \Big(\frac{\partial l}{\partial \mathbf{Q}} + \Big(\frac{\partial l}{\partial \mathbf{Q}}\Big)^T\Big)\mathbf{U}\mathbf{F} \\ \frac{\partial l}{\partial \mathbf{\Lambda}} &= \alpha\Big(\mathrm{diag}\Big(\lambda_1^{\alpha-1}, \ldots, \lambda_d^{\alpha-1}\Big)\mathbf{U}^T\frac{\partial l}{\partial \mathbf{Q}}\mathbf{U}\Big)_{\mathrm{diag}}, \end{aligned} \tag{7}$$

where $\mathbf{A}_{\mathrm{diag}}$ denotes the operation preserving the diagonal entries of $\mathbf{A}$ while setting all non-diagonal entries to zero. For MPN+M-ℓ_2 and MPN+M-Fro, $\frac{\partial l}{\partial \mathbf{\Lambda}}$ takes respectively the following forms:

$$\begin{aligned} \frac{\partial l}{\partial \mathbf{\Lambda}} =& \frac{\alpha}{\lambda_1^\alpha}\Big(\mathrm{diag}\Big(\lambda_1^{\alpha-1}, \ldots, \lambda_d^{\alpha-1}\Big)\mathbf{U}^T\frac{\partial l}{\partial \mathbf{Q}}\mathbf{U}\Big)_{\mathrm{diag}} \\ &- \mathrm{diag}\Big(\frac{\alpha}{\lambda_1}\mathrm{tr}\Big(\mathbf{Q}\frac{\partial l}{\partial \mathbf{Q}}\Big), 0, \ldots, 0\Big) \end{aligned} \tag{8}$$

and

$$\begin{aligned} \frac{\partial l}{\partial \mathbf{\Lambda}} =& \frac{\alpha}{\sqrt{\sum_k \lambda_k^{2\alpha}}}\Big(\mathrm{diag}\Big(\lambda_1^{\alpha-1}, \ldots, \lambda_d^{\alpha-1}\Big)\mathbf{U}^T\frac{\partial l}{\partial \mathbf{Q}}\mathbf{U}\Big)_{\mathrm{diag}} \\ &- \frac{\alpha}{\sum_k \lambda_k^{2\alpha}}\mathrm{tr}\Big(\mathbf{Q}\frac{\partial l}{\partial \mathbf{Q}}\Big)\mathrm{diag}\Big(\lambda_1^{2\alpha-1}, \ldots, \lambda_d^{2\alpha-1}\Big). \end{aligned} \tag{9}$$

Next, given $\frac{\partial l}{\partial \mathbf{U}}$ and $\frac{\partial l}{\partial \mathbf{\Lambda}}$, let us compute $\frac{\partial l}{\partial \mathbf{P}}$ associated with EIG (2). The chain rule is $\mathrm{tr}((\frac{\partial l}{\partial \mathbf{P}})^T\mathrm{d}\mathbf{P}) = \mathrm{tr}((\frac{\partial l}{\partial \mathbf{U}})^T\mathrm{d}\mathbf{U} + (\frac{\partial l}{\partial \mathbf{\Lambda}})^T\mathrm{d}\mathbf{\Lambda})$. Note that $\mathbf{U}$ should satisfy the orthogonal constraint. After some arrangements, we have

$$\frac{\partial l}{\partial \mathbf{P}} = \mathbf{U}\Big(\Big(\mathbf{K}^T \circ \Big(\mathbf{U}^T\frac{\partial l}{\partial \mathbf{U}}\Big)\Big) + \Big(\frac{\partial l}{\partial \mathbf{\Lambda}}\Big)_{\mathrm{diag}}\Big)\mathbf{U}^T, \tag{10}$$

where $\circ$ denotes matrix Kronecker product. The matrix $\mathbf{K} = \{K_{ij}\}$ where $K_{ij} = 1/(\lambda_i - \lambda_j)$ if $i \neq j$ and $K_{ij} = 0$

otherwise. We refer readers to [12, Proposition 2] for in-depth derivation of Eq. (10).

Finally, given $\frac{\partial l}{\partial \mathbf{P}}$, we derive the gradient of the loss function with respect to the input matrix $\mathbf{X}$, which takes the following form:

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

4. The Mechanism Underlying MPN-COV

This section explains the mechanism underlying MPN-COV. We provide interpretations from the statistical and geometric points of view, and make qualitative analysis from computational perspective.

4.1. MPN-COV Amounts to Robust Covariance Estimation

The sample covariance amounts to the solution to the Maximum Likelihood Estimation (MLE) of normally distributed random vectors. Though MLE is widely used to estimate covariances, it is well known that it performs poorly when the sample of data is of large dimension with small size [4, 32]. This is just what our covariance pooling faces: in most state-of-the-art ConvNets [27, 10, 9], the last convolutional layer outputs less than 200 features of dimension larger than 512, and so the sample covariances are always rank-deficient, rendering robust estimation critical.

The robust estimation of large-dimensional covariances with small sample size has been of great interest in statistics [28], signal processing [4] and biology [32]. Stein [28] for the first time proposes the shrinkage principle for eigenvalues of sample covariances. Ledioit and Wolf [18] has shown that the largest eigenvalues are systematically biased upwards while the smallest ones are biased downwards, and thus introduced the optimal linear shrinkage estimator, where the estimated covariance matrix $\mathbf{Q}$ is a linear combination of the sample covariance $\mathbf{P}$ with the identity matrix (i.e., $\mathbf{Q} = \alpha_1\mathbf{P} + \alpha_2\mathbf{I}$). This method with α_i decided by cross-validation is widely used to counteract the ill-conditioning of covariance matrices. Our MPN-COV closely conforms to the shrinkage principle [28, 18], i.e., shrinking the largest sample eigenvalues and stretching the smallest ones, as will be shown later in Sec. 4.3. It only depends on the sample covariance, delivering an individualized shrinkage intensity to each eigenvalue.

A number of researchers propose various regularized MLE methods for robust covariance estimation (see [32] and references therein). An important conclusion we can draw is that MPN-COV can be deemed a robust covariance estimator, explicitly derived from a regularized MLE called vN-MLE, according to our previous work [31]. Specifically, we have

Proposition 1 *MPN-COV with $\alpha = \frac{1}{2}$ is the unique solution to the regularized MLE of covariance matrix, i.e.,*

$$\mathbf{P}^{\frac{1}{2}} = \arg\min_{\mathbf{\Sigma}} \log|\mathbf{\Sigma}| + \mathrm{tr}(\mathbf{\Sigma}^{-1}\mathbf{P}) + D_{\mathrm{vN}}(\mathbf{I}, \mathbf{\Sigma}), \tag{12}$$

where $\mathbf{\Sigma}$ is constrained to be positive semidefinite, and $D_{vN}(\mathbf{A}, \mathbf{B}) = \mathrm{tr}(\mathbf{A}(\log(\mathbf{A}) - \log(\mathbf{B})) - \mathbf{A} + \mathbf{B})$ is matrix von Neumann divergence.

Proposition 1 follows immediately by setting to one the regularizing parameter in [31, Theorem 1]. Note that the classical MLE only includes the first two terms on the right-hand side of Eq. (12), while the robust vN-MLE estimator introduces the third term, constraining the covariance matrix be similar to the identity matrix. It has been shown [31] that the vN-MLE outperforms other shrinkage methods [28, 18, 4] and regularized MLE method [32].

4.2. MPN-COV Approximately Exploits Riemannian Geometry

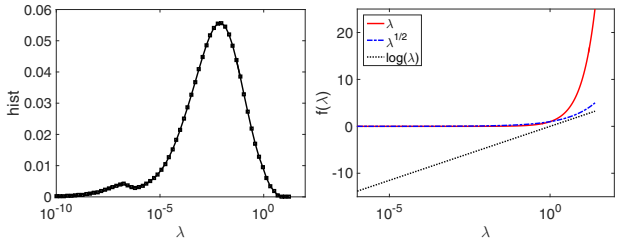
As the space of $d \times d$ covariance matrices, denoted by Sym_d^+, is a Riemannian manifold, it is appropriate to consider the geometrical structure when operating on this manifold. There are mainly two kinds of Riemannian metrics, i.e., the affine Riemannian metric [23] and the Log-E metric [1]. The former metric is affine-invariant, but is computationally inefficient and is coupled, not scalable to large-scale setting. In contrast, the most often used Log-E metric is similarity-invariant, efficient to compute and scalable to large-scale problems as it is a decoupled metric.

The metric for MPN-COV corresponds to the Power Euclidean (Pow-E) metric [7]. It has close connection with the Log-E metric, as presented in the following proposition:

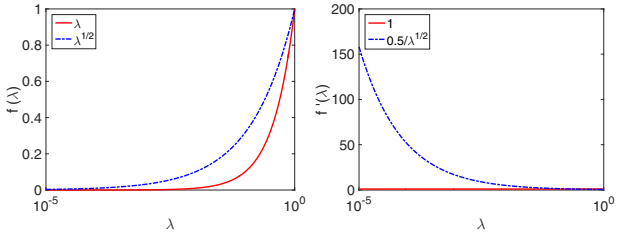
Proposition 2 *For any two covariance matrices $\mathbf{P}$ and $\widetilde{\mathbf{P}}$, the limit of the Pow-E metric $d_\alpha(\mathbf{P}, \widetilde{\mathbf{P}}) = \frac{1}{\alpha}\left\|\mathbf{P}^\alpha - \widetilde{\mathbf{P}}^\alpha\right\|_F$ as $\alpha > 0$ approaches 0 equals the Log-E metric, i.e., $\lim_{\alpha\to 0} d_\alpha(\mathbf{P}, \widetilde{\mathbf{P}}) = \left\|\log(\mathbf{P}) - \log(\widetilde{\mathbf{P}})\right\|_F$.*

This conclusion was first mentioned in [7] but without proof. Here we briefly prove this claim. Note that $d_\alpha(\mathbf{P}, \widetilde{\mathbf{P}}) = \left\|\frac{1}{\alpha}(\mathbf{P}^\alpha - \mathbf{I}) - \frac{1}{\alpha}(\widetilde{\mathbf{P}}^\alpha - \mathbf{I})\right\|_F$. Based on the eigenvalue decomposition of $\mathbf{P}$ we have $\frac{1}{\alpha}(\mathbf{P}^\alpha - \mathbf{I}) = \mathbf{U}\mathrm{diag}(\frac{\lambda_1^\alpha - 1}{\alpha}, \ldots, \frac{\lambda_n^\alpha - 1}{\alpha})\mathbf{U}^T$. The identity about the limit in Proposition 2 follows immediately by recalling $\lim_{\alpha\to 0}\frac{\lambda^\alpha - 1}{\alpha} = \log(\lambda)$.

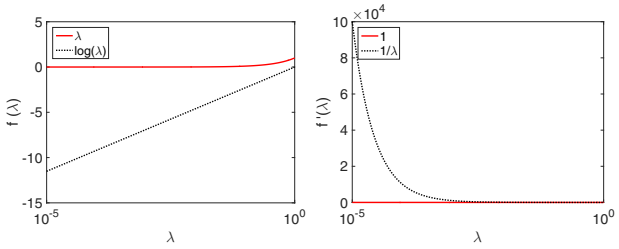
Hence, the proposed MPN-COV can be viewed as approximately exploiting the Riemannian geometry of Sym_d^+. It might seem that the Log-E metric is a better option than the Pow-E metric, since the former measures the true geodesic distance but the latter one only measures it approximately. We argue that this is not the case of our problem for two reasons. First, the Log-E metric requires the eigenvalues involved to be *strictly positive* [1, 34] while the Pow-E



(a) Eigenvalue histogram and normalization functions



(b) $\lambda^{\frac{1}{2}}$ and its derivative zoomed on $[10^{-5}, 1]$



(c) $\log(\lambda)$ and its derivative zoomed on $[10^{-5}, 1]$

Figure 2. Illustration of empirical distribution of eigenvalues and normalization functions. The identity $f(\lambda) = \lambda$ (no normalization) and its derivative are also plotted for reference. $\lambda^{\frac{1}{2}}$ conforms to the general shrinkage principle as suggested in [28, 18], which shrinks the largest eigenvalues and stretches the smallest ones, while preserving the order of eigenvalue significances. In contrast, $\log(\lambda)$ over-stretches the smallest eigenvalues, reversing the order of significance such that, after normalization, the smallest eigenvalues play more crucial roles than the largest ones.

metric allows *non-negative* eigenvalues [7, 34]. For Log-E the common method is to add a small positive number ϵ to eigenvalues for improving numerical stability. Although ϵ can be decided by cross-validation, it is difficult to seek a particular ϵ well suitable for a huge number of images. For example, [11] suggest $\epsilon = 10^{-3}$, which will smooth out eigenvalues less than 10^{-3}. Above all, the distributions of high-level, convolutional features are such that the logarithm brings side effect, which will be qualitatively analyzed in the next subsection. We will also quantitatively compare the two metrics by experiments in Sec. 5.2.

4.3. Qualitative Analysis

This section qualitatively analyzes, from the computational perspective, the impact of matrix power and logarithm on the eigenvalues of sample covariances. The matrix logarithm can be regarded as a kind of normalization, nonlinearly applied to the eigenvalues: $\mathbf{Q} \triangleq \log(\mathbf{P}) = \mathbf{U}\mathrm{diag}(\log(\lambda_1), \ldots, \log(\lambda_d))\mathbf{U}^T$. Below we will concentrate on power function $f(\lambda) = \lambda^{\frac{1}{2}}$ and logarithm $f(\lambda) = \log(\lambda)$.

We first examine the empirical distribution of eigenvalues of sample covariances. We randomly select 300,000 images from the training set of ImageNet 2012. For each image, we extract the output of the 5th conv. (Conv5) layer (with ReLU) using AlexNet model pretrained on ImageNet 2012, estimate the sample covariance $\mathbf{P}$, and then compute its eigenvalues using EIG in single-precision floating-point format. For a training image of 227×227, Conv5 outputs 13×13 features with 256 channels, reshaped to a matrix $\mathbf{X} \in \mathbb{R}^{256\times 169}$. As the rank of $\mathbf{P}$ is less than 169, $\mathbf{P}$ has less than 169 non-zeros eigenvalues. We mention that very small eigenvalues obtained by EIG may be inaccurate due to machine precision. The histogram of eigenvalues is shown in Fig. 2(a)(left), where zero eigenvalues are excluded for better view. Fig. 2(a)(right) shows the two normalization functions over $[10^{-5}, 10]$. The graphs of $\lambda^{\frac{1}{2}}$ & its derivative and $\log(\lambda)$ & its derivative, both zoomed on $[10^{-5}, 1]$, are shown in Fig. 2(b) and Fig. 2(c), respectively.

The function $\log(\lambda)$ considerably changes the eigenvalue magnitudes, reversing the order of eigenvalue significances, e.g., a significant eigenvalue $\lambda = 50 \mapsto \log(\lambda) \approx 3.9$ but an insignificant one $\lambda = 10^{-3} \mapsto \log(\lambda) \approx -6.9$. From the forward propagation formula $\mathbf{P} = \sum_i \lambda_i \mathbf{u}_i \mathbf{u}_i^T \mapsto \mathbf{Q} = \sum_i \log(\lambda_i)\mathbf{u}_i\mathbf{u}_i^T$ it can be seen that the smallest eigenvalues will play more crucial roles than the largest ones. This effect is also obvious if we consider the backprogation formula for the gradient $\frac{\partial l}{\partial \lambda_i}$ before and after normalization, i.e., $\mathbf{u}_i^T \frac{\partial l}{\partial \mathbf{Q}} \mathbf{u}_i \mapsto \frac{1}{\lambda_i}\mathbf{u}_i^T \frac{\partial l}{\partial \mathbf{Q}} \mathbf{u}_i, i = 1, \ldots, d$. For example, the derivative of $\log(\lambda)$ at $\lambda = 10^{-3}$ is 10^3 but at $\lambda = 50$ is 2×10^{-2}. Since significant eigenvalues are generally more important in that they capture the statistics of principal directions along which the feature variances are larger, matrix logarithm will deteriorate the covariance representations.

Now let us consider $\lambda^{\frac{1}{2}}$. It nonlinearly shrinks the eigenvalues larger than one, and the larger, the more shrunk, while stretching those less than one, and the smaller, the more stretched. This kind of normalization conforms to the general shrinkage principle as suggested in [28, 18]. Contrary to $\log(\lambda)$, it does not change the order of eigenvalue significances– significant (resp. insignificant) eigenvalues maintain significant (resp. insignificant). For example, $\lambda = 50 \mapsto \lambda^{\frac{1}{2}} \approx 7.1$ while $\lambda = 10^{-3} \mapsto \lambda^{\frac{1}{2}} \approx 0.032$. From the forward propagation formula $\mathbf{S} = \sum_i \lambda_i \mathbf{u}_i \mathbf{u}_i^T \mapsto \mathbf{Q} = \sum_i \lambda^{\frac{1}{2}} \mathbf{u}_i \mathbf{u}_i^T$, we see that the order of amount of contributions made by individual eigenvalues keep unchanged. Similar conclusion can be drawn if we consider the back-propagation formula of $\frac{\partial l}{\partial \lambda_i}$: $\mathbf{u}_i^T \frac{\partial l}{\partial \mathbf{Q}} \mathbf{u}_i \mapsto \frac{1}{2\sqrt{\lambda_i}} \mathbf{u}_i^T \frac{\partial l}{\partial \mathbf{Q}} \mathbf{u}_i$.

5. Experiments

We make experiments on ImageNet 2012 classification dataset [24], which consists of 1,000 classes, including roughly 1.28 million training images, 50k validation images, and 100k testing ones. We do not adopt extra training images. Following the common practice, we report top-1 and top-5 error rates on the validation set as measures of recognition performance. We develop programs based on MatConvNet [30] and Matlab 2015b under 64-bit Windows 7.0. The programs run on six workstations each of which is equipped with a Intel i7-4790k@4.0Ghz CPU and 32G RAM. Two NVIDA Titan X with 12 GB memory and four NVIDA GTX 1080 with 8 GB memory are used, one graphics card per workstation.

5.1. Implementation of MPN-COV Networks

To implement MPN-COV layer, we adopt the EIG algorithm on CPU in single-precision floating-point format, as its GPU version provided on the CUDA platform is several times slower. Except for EIG, all other operations in forward and backward propagations are performed on GPU. Since MPN-COV allows non-negative eigenvalues, we truncate to zeros the eigenvalues smaller than $\mathrm{eps}(\lambda_1)$, which indicates the positive distance from the maximum eigenvalue λ_1 to its next larger floating-point number. Our MPN-COV pooling replaces the common first-order, max/average pooling after the last conv. layer, producing a global, $d(d+1)/2-$dimensional image representation by concatenation of the upper triangular part of one covariance matrix. In state-of-the-art ConvNets, the feature dimension d of the last conv. layer gets much larger. For such architectures, we add a 1×1 conv. layer of 256 channels after the last conv. layer, so that the dimension of features inputted to the MPN-COV layer is fixed to 256 (see Sec. 5.3 and Sec. 5.4). As such, we alleviate the problem of small sample of large-dimensional features while decreasing the computational cost of the MPN-COV layer.

We adopt the standard color jittering technique [14] for training set augmentation. For AlexNet [14] and VGG-M [3], we follow the default setting in MatConvNet [30] where each training image is rescaled such that its shorter side is of 256 pixels. For VGG-16 [27] and ResNet [9], following [27], we rescale isotropically each training image with shorter side randomly sampled on $[256, 512]$. Then, we sample a fixed size patch at random from the resized image or its mirror, and subtract the mean RGB value from each pixel. In testing stage, we first isotropically resize each test image with short side 256, then adopt the commonly used 1-crop prediction or 10-crop prediction for performance evaluation. Following [10], we adopt batch normalization right after every convolution and before ReLU and no drop out.

We use mini-batch stochastic gradient descent with momentum (set to 0.9 throughout the experiments) for training. For AlexNet, VGG-M and VGG-16, we set the weight decay to 5×10^{-4}, and their mini-batch sizes are set to 128, 100 and 32, respectively. For training from scratch, the filter weights are initialized with a normal distribution $\mathcal{N}(0, 0.01)$ with mean 0 and variance 0.01 and the biases are initialized with zero [27]; ConvNets are trained up to 20 epochs, where the learning rates follow exponential decay, changing from 10^{-1} to 10^{-4} and $10^{-1.2}$ to 10^{-5} for the ConvNets with first-order pooling and those with MPN-COV pooling, respectively. For ResNets, following [9], we use a weight decay of 10^{-4} and a mini-batch size of 256, and initialize the biases with zero and the filter weights with $\mathcal{N}(0, 2/n)$, where n is the product of the size and #channels of filters. The ResNet-50 with MPN-COV is trained up to 90 epochs with learning rate initialized to $10^{-1.2}$ and divided by 10 when the error plateaus.

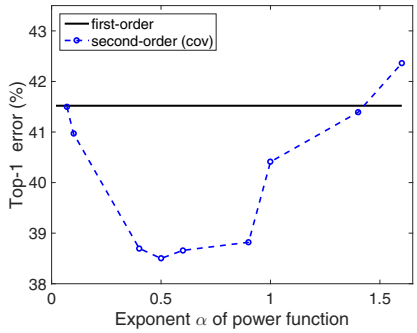


Figure 3. Impact of α on second-order cov pooling under AlextNet Architecture. Top-1 errors (*1-crop*) are reported. The bold line indicates the result of the AlexNet with first-order pooling.

5.2. Evaluation of MPN-COV Under AlexNet Architecture

In the first part of experiments, we evaluate MPN-COV by selecting AlexNet architecture [14], since it is shallower and runs faster than its variants. As recently proposed deeper ConvNets [27, 9] follow the basic architecture of AlexNet, our analysis here can extrapolate to them. We study the impact of exponent α of power function and various matrix normalization methods on cov pooling. We also compare with two existing end-to-end, second-order pooling methods, i.e., DeepO^{2}P [11] concerned with matrix normalization by logarithm and B-CNN [22] which performs element-wise power normalization.

Impact of Exponent α of Power Function. We first evaluate covariance (cov) pooling against the exponent α of power function. Fig. 3 shows top-1 errors versus α using single-crop prediction. We first note that the *plain cov pooling* ($\alpha = 1$, no normalization) produces an error rate of 40.41%, about 1.1% less than first-order max pooling. When $\alpha < 1$, the normalization function shrinks eigenvalues larger 1.0 and stretches those less than 1.0. As α (less than 1.0) decreases, the error rate continuously gets smaller

method	MPN	M-Fro	M-ℓ_2	init.	top-1 err.
First-order (ours)	–	–	–	Random	41.52
cov pool. (ours)	No	No	No	Random	40.41
	Yes	No	No	Random	**38.51**
	No	Yes	No	Random	39.87
	No	No	Yes	Random	39.65
	Yes	Yes	No	Random	39.93
	Yes	No	Yes	Random	39.62
	Yes	No	No	Warm	**37.35**

Table 1. Impact of various matrix normalizations under AlexNet architecture. We measure top-1 error (%, *1-crop*) for MPN, M-Fro and M-ℓ_2 as well as combinations of them.

method	top-1 error	top-5 error
MPN-COV (AlexNet) (ours)	**33.84**	**14.01**
Krizhevsky et al. [14]	40.7	18.2
VGG-F [3]	39.11	16.77

Table 2. Error (%, *10-crop*) comparison of MPN-COV (AlexNet) with ConvNets having similar architecture.

until the smallest value at around $\alpha = \frac{1}{2}$. With further decline of α, however, we observe the error rate grows consistently and soon is larger than that of the plain cov pooling. Note that over the interval $[0.4, 0.9]$ the performance of cov pooling varies insignificantly. When $\alpha > 1$, the effect of normalization is contrary, i.e., eigenvalues less than 1.0 are shrunk while those larger than 1.0 are stretched, which is not beneficial for covariance representations as indicated by the consistent growth of the error rates. In all the following experiments, we set $\alpha = \frac{1}{2}$.

Impact of Various Normalization Methods. We mainly compare three kinds of normalizations (i.e., MPN, M-Fro and M-ℓ_2) as well as their combinations. Table 1 summarizes the comparison results. Compared to first-order pooling, the plain cov pooling and cov pooling with normalizations decrease the errors by a gap of over 1.1%, which clearly indicates that the second-order pooling is better than the most commonly used, first-order pooling. When used separately, all of the three matrix normalizations improve over the plain cov pooling. MPN outperforms M-Fro and M-ℓ_2 by $\sim 1.3\%$ and $\sim 1.1\%$, respectively, but combinations of M-Fro (or MPN-ℓ_2) with MPN degrade the performance of separate MPN. We also made experiments by performing element-wise power normalization right after MPN , which, however, degraded the performance by over 1%. As our MPN-COV explores the second-order statistics, which is more complex than the first-order one and may complicate minimization of the loss function, we consider warm initialization. Specifically, the weights of all layers before the cov pooling are initialized with the trained network with max pooling and their learning rates are set to $\text{logspace}[-2, -5, 20]$, while those after cov pooling are randomly initialized with learning rates doubled. Compared to random initialization (training from scratch), the warm initialization decreases error by $\sim 1.1\%$, which indicates it helps MPN-COV network converge to a better local minimum of the loss function. In the end, we compare in Table 2 our MPN-COV network with two ConvNets with similar architectures, i.e., VGG-F [3] and Krizhevsky et al. [14]. The result of VGG-F is obtained by us using the model released on the MatConvNet website. Our MPN-COV network performs much better than both of them.

method	init.	top-1 error	top-5 error
Plain COV (ours)	random	40.41	18.94
MPN-COV (ours)	random	**38.51**	**17.60**
B-CNN [22]	random	39.89	18.32
DeepO$_2$P [11]	random	42.16	19.62

Table 3. Error (%, *1-crop*) comparison of MPN-COV with two existing second-order pooling methods under AlexNet architecture.

Comparison with Existing Second-order Methods. Here we compare with DeepO$_2$P [11] and B-CNN [22], two existing end-to-end, second-order pooling methods, neither of which has been previously evaluated on ImageNet dataset. For DeepO$_2$P, Ionescu et al. suggested implementation of the nonlinear, matrix logarithm using SVD and double-precision floating-point format for both the forward and backward propagations. We adopted the code released by them. As suggested, we add $\epsilon = 10^{-3}$ to the eigenvalues for numerical stability. For a matrix $\mathbf{P} = [p_{ij}]$, B-CNN computes $\mathbf{Q} = \left[\text{sign}(p_{ij})(|p_{ij}| + \epsilon)^{\beta}\right]$ where sign is the signum function. For implementation, we adopted the code released by the authors of B-CNN, where $\beta = 0.5$ and $\epsilon = 10^{-5}$ as suggested. Table 3 presents the comparison results. We can see that DeepO$_2$P is inferior to the plain cov pooling (no normalization). As analyzed in Sec. 4.3, we attribute this to the fact that logarithm is not suitable for the convolutional features given the distribution as shown in Fig. 2(a) (left), as it changes the order of eigenvalue significances. B-CNN slightly improves the plain cov pooling by $\sim 0.5\%$, but outperformed by MPN-COV by 1.4%. We mention that we tuned ϵ for DeepO$_2$P and β and ϵ for B-CNN but achieved trivial improvement.

5.3. MPN-COV Under VGG-Net Architectures

In this section, we combine MPN-COV with two VGG networks, i.e. VGG-M [3] and VGG-16 [27]. We slightly modify VGG-M by presenting two configurations (config.). In config. a, we add an additional $1 \times 1 \times 256$ conv. layer (filter size: 1×1, channel: 256) right after Conv5, and for config. b, the numbers of channels of Conv3 and Conv4 are both raised from 512 to 640 while that of Conv5 is reduced to 256. Then our MPN-COV layer follows. Note that the 1×1 convolution has been used for dimensionality reduction and introducing nonlinearity [21, 29]. Either config. produces a sample of features $\mathbf{X} \in \mathbb{R}^{256 \times 169}$ for

method	init.	config. a	config. b
First-order (ours)	Random	37.07	37.31
MPN-COV (ours)	Random	34.60	35.27
	warm	**33.44**	34.25

Table 4. Top-1 error (%, *1-crop*) under VGG-M architecture.

method	init.	top-1 error	top-5 error
First-order (ours)	random	29.62	10.81
MPN-COV (ours)	random	**26.55**	**8.94**

Table 5. Error rates (%, *1-crop*) under VGG-16 architecture.

method	top-1 error	top-5 error
MPN-COV (VGG-M) (ours)	**30.39**	**11.43**
VGG-M [3]	34.00	13.49
Zeiler & Fergus [33]	37.5	16.0
OverFeat [26]	35.60	14.71

Table 6. Error (%, *10-crop*) comparison of MPN-COV (VGG-M) with ConvNets sharing similar architecture.

method	#layers	top-1 err.	top-5 err.
MPN-COV(VGG-16)(ours)	17	**24.68**	**7.75**
VGG-16 [27]	16	27.41	9.20
GoogleNet [29]	22	–	9.15
PReLU-net B [8]	22	25.53	8.13

Table 7. Error (%, *10-crop*) comparison of MPN-COV (VGG-16) with two ConvNets having comparable number of conv. layers.

the MPN-COV layer. As seen from Table 4, under config. a with random initialization MPN-COV outperforms the first-order, max pooling by $\sim$2.5% and with warm initialization the gap increases to $\sim$3.6%. For config. b the gains over max pooling are a little less than Config. a, i.e., $\sim$2% and $\sim$3% under random and warm initialization, respectively. For VGG-16, we add a $1 \times 1 \times 256$ convolution after the last conv. layer, obtain the feature matrix $\mathbf{X} \in \mathbb{R}^{256\times 196}$ for cov pooling. The results are shown in Table 5, from which we see that MPN-COV can bring large improvement for VGG-16 architecture.

Table 6 presents comparison of MPN-COV (VGG-M) with the original VGG-M [3], Zeiler & Fergus [33] and OverFeat [26], all sharing similar network architecture. Our MPN-COV (VGG-M) shows much better performance than them. In Table 7, we can see that in terms of top-1 error, MPN-COV (VGG-16) outperforms the original VGG-16 [27] by $\sim$2.7%, and in terms of top-5 error, it performs better than GoogleNet [29] and PReLU-net B [8] by $\sim$1.4% and $\sim$0.4%, respectively. PReLU-net C is similar to PReLU-net B but significantly increases #channels of every filter [8], producing slightly better performance than ours. Note that as the authors did not report the 10-crop results of the original VGG-M and VGG-16, we obtain them by using the best-performing models, i.e., matconvnet-vgg-m.mat and vgg-verydeep-16.mat, respectively.

method	init.	top-1 error	top-5 error
First-order (ours)	random	24.95	7.52
MPN-COV (ours)	random	**22.73**	**6.54**

Table 8. Error rates (%,*1-crop*) under ResNet-50 architecture.

method	top-1 error	top-5 error
MPN-COV (ResNet-50) (ours)	**21.20**	5.74
ResNet-50 [9]	22.85	6.71
ResNet-101 [9]	21.75	6.05
ResNet-152 [9]	21.43	**5.71**

Table 9. Error (%, *10-crop*) comparison of MPN-COV (ResNet-50) with the original ResNets.

5.4. MPN-COV Under ResNet Architecture

Finally, we integrate MPN-COV into ResNet-50 (baseline). To retain as many number of features as possible, we do not perform downsampling in conv5_1, as done in the original network, for the last set of building blocks (i.e. conv5_x). Then we connect the last addition layer (with ReLU) to a $1 \times 1 \times 256$ conv. layer, followed by the MPN-COV layer. As such, we have a sample of features $\mathbf{X} \in \mathbb{R}^{256\times 196}$ for covariance estimation. Regarding the time (ms) taken per image, MPN-COV network vs baseline are 18.21 vs 14.37 for training and 5.8 vs 3.5 for inference, respectively. We observe MPN-COV network converges faster: training/validation error rates (%) of MPN-COV vs baseline reach 37.0/34.3 vs 50.35/44.55 at epoch 30 and 18.02/23.19 vs 25.98/25.76 at epoch 60. Table 8 shows that MPN-COV produces $\sim$2.2% top-1 error (1-crop) less than the first-order average pooling. Table 9 shows that, compared to the original ResNets, with 10-crop prediction our MPN-COV network performs 1.65% better than ResNet-50, while outperforming ResNet-101 and being comparable to ResNet-152. By exploiting second-order statistics we achieve performance matching extremely deep ConvNets with much shallower one.

6. Conclusion

This paper proposed a matrix normalized covariance (MPN-COV) method for exploring the second-order statistics in large-scale classification. MPN-COV amounts to robust covariance estimation given a small number of large-dimensional features. It also approximately exploits the geometry of the space of covariance matrices, while circumventing the downside of the well-known Log-Euclidean metric. Extensive experiments on ImageNet 2012 dataset showed that our MPN-COV networks achieved competitive gains over its counterparts using only first-order information. In future we will combine MPN-COV with the Inception architecture [29], and study applications of MPN-COV to visual tasks such as object detection, scene categorization and fine-grained visual recognition.

References

[1] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM J. on Matrix Analysis and Applications*, 2006. 1, 2, 4

[2] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, pages 430–443, 2012. 2

[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. 6, 7, 8

[4] Y. Chen, A. Wiesel, Y. C. Eldar, and A. O. Hero. Shrinkage algorithms for MMSE covariance estimation. *IEEE TSP*, 58(10):5016–5029, 2010. 4

[5] A. Cherian, P. Koniusz, and S. Gould. Higher-order pooling of CNN features via kernel linearization for action recognition. In *WACV*, 2017. 2

[6] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and segmentation. In *CVPR*, 2015. 1, 2

[7] I. L. Dryden, A. Koloydenko, and D. Zhou. Non-Euclidean statistics for covariance matrices, with applications to diffusion tensor imaging. *The Annals of Applied Statistics*, 3(3):1102–1123, 2009. 4, 5

[8] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 8

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 4, 6, 8

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4, 6

[11] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015. 1, 2, 3, 5, 6, 7

[12] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv*, abs/1509.07838, 2015. 2, 3, 4

[13] P. Koniusz, F. Yan, P. H. Gosselin, and K. Mikolajczyk. Higher-order occurrence pooling for bags-of-words: Visual concept detection. *IEEE TPAMI*, 39(2):313–326, Feb 2017. 2

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 6, 7

[15] L. D. Lathauwer, B. D. Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM. J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000. 2

[16] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 1

[17] 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

[18] O. Ledoit and M. Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *J. Multivariate Analysis*, 88(2):365–411, 2004. 4, 5

[19] P. Li, X. Lu, and Q. Wang. From dictionary of visual words to subspaces: Locality-constrained affine subspace coding. In *CVPR*, 2015. 2

[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] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014. 7

[22] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *ICCV*, 2015. 1, 2, 6, 7

[23] X. Pennec, P. Fillard, and N. Ayache. A Riemannian framework for tensor computing. *IJCV*, pages 41–66, 2006. 4

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 1, 6

[25] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher vector: Theory and practice. *IJCV*, 105(3):222–245, 2013. 2, 3

[26] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014. 8

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 4, 6, 7, 8

[28] C. Stein. Lectures on the theory of estimation of many parameters. *Journal of Soviet Mathematics*, 34(1):1373–1403, 1986. 4, 5

[29] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1, 7, 8

[30] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for MATLAB. In *ACM on Multimedia*, 2015. 6

[31] 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. 1, 2, 4

[32] E. Yang, A. Lozano, and P. Ravikumar. Elementary estimators for sparse covariance matrices and other structured moments. In *ICML*, 2014. 4

[33] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pages 818–833, 2014. 1, 8

[34] 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. 4, 5