This is the Pre-Published Version.

The following publication Q. Wang, P. Li and L. Zhang, "G2DeNet: Global Gaussian Distribution Embedding Network and Its Application to Visual Recognition," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 6507-6516 is available at https://doi.org/10.1109/CVPR.2017.689.

G^2DeNet: Global Gaussian Distribution Embedding Network and Its Application to Visual Recognition*

Qilong Wang[1], Peihua Li[1], Lei Zhang[2]
[1]Dalian University of Technology, [2]Hong Kong Polytechnic University
qlwang@mail.dlut.edu.cn, peihuali@dlut.edu.cn, cslzhang@comp.polyu.edu.hk

Abstract

Recently, plugging trainable structural layers into deep convolutional neural networks (CNNs) as image representations has made promising progress. However, there has been little work on inserting parametric probability distributions, which can effectively model feature statistics, into deep CNNs in an end-to-end manner. This paper proposes a Global Gaussian Distribution embedding Network (G^2DeNet) to take a step towards addressing this problem. The core of G^2DeNet is a novel trainable layer of a global Gaussian as an image representation plugged into deep CNNs for end-to-end learning. The challenge is that the proposed layer involves Gaussian distributions whose space is not a linear space, which makes its forward and backward propagations be non-intuitive and non-trivial. To tackle this issue, we employ a Gaussian embedding strategy which respects the structures of both Riemannian manifold and smooth group of Gaussians. Based on this strategy, we construct the proposed global Gaussian embedding layer and decompose it into two sub-layers: the matrix partition sub-layer decoupling the mean vector and covariance matrix entangled in the embedding matrix, and the square-rooted, symmetric positive definite matrix sub-layer. In this way, we can derive the partial derivatives associated with the proposed structural layer and thus allow backpropagation of gradients. Experimental results on large scale region classification and fine-grained recognition tasks show that G^2DeNet is superior to its counterparts, capable of achieving state-of-the-art performance.

1. Introduction

Modeling activations of convolutional layers or fully-connected layers of pre-trained deep convolutional neural networks (CNNs) as image representations has been very successful in a variety of computer vision tasks, such as object recognition [25], image retrieval [9] and texture classification [6]. However, these methods handle feature learning, image modeling and loss function (e.g., classifier) in separate stages. Recent researches have shown it is meaningful and helpful to plug modeling methods into deep CNN architectures as structural layers in an end-to-end manner [14, 24, 1, 37]. Compared with [9, 6], the end-to-end approaches can jointly leverage the power of learning features, representing images and training classifiers [1, 29].

To represent images, the probability distributions are widely used as they generally have capability to model abundant statistics of features, producing fixed size representations regardless of varying feature sizes [28, 32, 33, 40]. Unfortunately, there has been little work attempting to plug trainable probability distribution modeling layers into deep CNNs. Oliva *et al*. [29] make an effort to propose a deep mean maps (DMMs) method, which can plug a family of non-parametric distributions into deep CNNs. By exploiting the mean of random Fourier features [31] to approximate the mean map embeddings of distributions [11], the DMMs layer is decomposed into common operations of convolution, pixel-wise cosine and average pooling so that forward and backward propagations can be easily accomplished. It is reported that the DMMs layer improves the existing CNNs on several real-world datasets. However, the DMMs method does not consider special characteristics of individual distributions, for example, exponential distributions have specific geometric structures.

Although the DMMs method has been studied, combining parametric probability distributions modeling into deep CNNs still is an open problem. In this paper, we take a step forward towards addressing this problem. Specifically, as in [28, 33, 40], we use global Gaussians as image representations and propose a global Gaussian embedding layer to combine them in deep CNN architectures. In contrast to DMMs [29], we explicitly take advantage of the geometry of Gaussians by considering their parameters (i.e., the mean vectors and covariance matrices) rather than using approximated embeddings of distributions in DMMs. This renders

*P. Li, to whom correspondence should be addressed, was supported by National Natural Science Foundation of China (No. 61471082). L. Zhang was supported by National Natural Science Foundation of China (No. 61672446). We thank NVIDIA corporation for donating GPU.

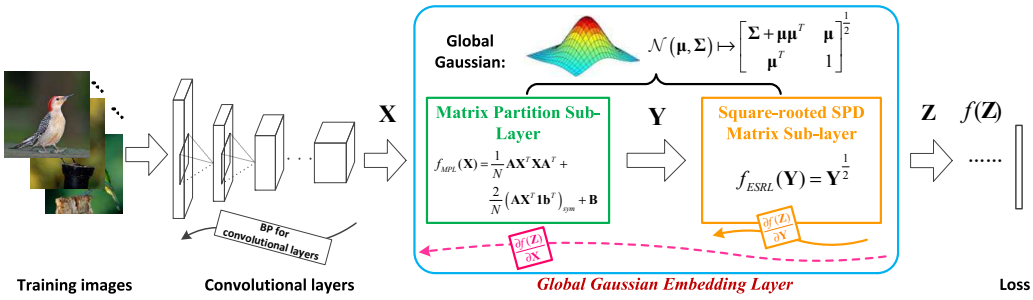

Figure 1. Overview of the proposed Global Gaussian Distribution embedding Network (G^2DeNet). The core of G^2DeNet is a novel layer of global Gaussians as image representations, inserted after the last convolutional layer in a deep CNN in an end-to-end manner. By first identifying a Gaussian as the square root of an SPD matrix based on Lie group theory, we decompose the layer into two sub-layers and develop a method to compute partial derivatives using matrix variations and SVD. For detailed mathematical notations, refer to Section 3.

our method more challenging since we need to consider the Riemannian structure of Gaussians as well as its forward and backward propagations in the deep CNNs.

To include Gaussian representation as a layer in the deep CNNs, we first exploit a Gaussian embedding strategy based on Lie group theory, where a Gaussian distribution is uniquely transformed to a square-rooted symmetric positive definite (SPD) matrix. To make our global Gaussian embedding layer trainable, we decompose it into two consecutive sub-layers: the matrix partition sub-layer and the square-rooted SPD matrix sub-layer. The first sub-layer decouples the mean vector and covariance matrix entangled in the embedding matrix so that it can be explicitly written as a function of features, while the second one is to compute the square root of an SPD matrix through the singular value decomposition (SVD). Then, we develop a method to compute the partial derivatives associated with the two sub-layers based on the theory of matrix variations. In this way, we can perform forward and backward propagations through the global Gaussian embedding layer. For convenience, hereafter, the proposed network is called Global Gaussian Distribution embedding Network (G^2DeNet), whose overview is illustrated in Figure 1. At the core of G^2DeNet is a trainable layer of a global Gaussian as an image representation, inserted after the last convolutional layer in the deep CNNs.

The contributions of this paper lie in three folds: (1) we propose a novel trainable structural layer, which can plug global Gaussian distributions into deep CNNs for powerful image representation. To our best knowledge, this is the first attempt to plug a parametric probability distribution into CNN architectures in an end-to-end form. (2) Technically, to make possible the forward and backward propagations on Gaussian manifold, we exploit a Gaussian embedding strategy based on Lie group theory and develop a structural backpropagation method. (3) The experiments are extensively conducted on large scale MS-COCO [23] and challenging fine-grained benchmarks [39, 27, 20], demonstrating superiority of the proposed method.

2. Related Work

Ionescu *et al*. [14] establish the theory and practice of global, structured matrix backpropagation in an end-to-end training framework. In particular, they propose theorems on variations of SVD or eigenvalue decomposition (EIG) and instantiate the $DeepO_2P$ model for region classification. At the heart of $DeepO_2P$ is a trainable O_2P layer plugged into the deep CNN architecture performing second-order pooling of convolutional features. The O_2P layer in $DeepO_2P$ leads to second-order, non-central moments which are SPD matrices and whose geometry is handled using Log-Euclidean metrics [2], resulting in backpropagation of logarithm of SPD matrices. The structural matrix backpropagation theory [14] motivates the backpropagation of our proposed method on Gaussian manifold. However, different from $DeepO_2P$, we attempt to insert a trainable Gaussian distribution layer, where the geometry of Gaussians is quite different from the geometry involved in $DeepO_2P$. Specifically, we introduce a Gaussian embedding strategy based on Lie group theory, which uniquely maps a Gaussian to a square-rooted SPD matrix. Note that the second-order moments can be seen as Gaussians of zero-mean. In Section 4.1, we show that our G^2DeNet outperforms $DeepO_2P$ by a margin while preserving a comparable complexity.

The bilinear CNN (BCNN) [24] model inserts a trainable bilinear pooling layer after the last convolutional layer in CNN architectures. This layer computes the outer products of features from two CNN models, and then performs sum-pooling and normalization. When the two CNN models are different, BCNN captures correlations of different sources of features. If the two CNN models are identical, the outer products plus sum-pooling leads to second-order, non-central moments, as in $DeepO_2P$; differently, BCNN performs power normalization followed by ℓ_2-normalization for the resulting SPD matrices rather than matrix logarithm used in $DeepO_2P$. In contrast to BCNN, our purpose is to propose image representations by parametric, Gaussian

distributions for end-to-end learning while respecting their structures of manifold and Lie group, which is distinct from BCNN in theory and implementation of both the forward and backward propagations. In addition, comparisons in Section 4.2 show that the proposed G^{2}DeNet is superior to BCNN in exactly the same experimental settings.

The other related works include NetVLAD [1] and FisherNet [37]. They both concern insertion into the CNN architectures of a trainable layer consisting of features encoding and pooling to form an orderless image representation. The NetVLAD accomplishes the trainable layer of the generalized vector of locally aggregated descriptors (VLAD) [17]. The FisherNet proposes a method to implement the Fisher vector (FV) [32] in an end-to-end learning manner. Regarding implementation, both of the two methods decompose (after appropriate modifications or simplifications) the inserted layers into typical operations of convolution, softmax and pooling so that off-the-shelf implementation of layers can be conveniently used. Different from them, in our G^{2}DeNet, the trainable layer is concerned with the Gaussian distribution which involves structural backpropagation on manifold, and the commodity operations in the classical CNN cannot be simply used.

3. Global Gaussian Distribution Embedding Network

In this section, we will introduce our global Gaussian distribution embedding network. To make Gaussian be integrated into CNN architectures, we first map a Gaussian to a square rooted SPD matrix. Then, we propose a novel global Gaussian embedding layer and develop a structural backpropagation method for our global Gaussian embedding layer. Finally, we give a brief introduction of G^{2}DeNet based on other Gaussian embedding methods.

3.1. Gaussian Embedding

In this paper, we use global Gaussians as image representations. Suppose we have a set of N d-dimensional features $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T \in \mathbb{R}^{N\times d}$, where T indicates matrix transpose. The Gaussian distribution of these features can be estimated as follows:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\big(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\big),$$

where $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$ and $\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\boldsymbol{\mu})(\mathbf{x}_i-\boldsymbol{\mu})^T$ are respectively mean vector and covariance matrix, and $|\cdot|$ indicates matrix determinant. The Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is determined by parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

We denote by $\mathcal{G}(d)$ the space of $d-$variate Gaussians. It has long been known [35] that this space is a Riemannian manifold having geometric structure. A recent work [22] has made advance, showing that $\mathcal{G}(d)$ can be endowed with a Lie group structure, i.e., it is not only a Riemannian manifold but is a smooth group. This paper exploits the embedding method in [22] to identify a Gaussian as a square-rooted SPD matrix. Let $UT^+(d+1)$ be the set of all positive definite upper triangular matrices of order $d+1$ which is a Lie group, and $\boldsymbol{\Sigma}^{-1} = \mathbf{L}\mathbf{L}^T$ be the Cholesky decomposition of the inverse of $\boldsymbol{\Sigma}$, where $\mathbf{L}$ is a lower triangular matrix of order d with positive diagonals. Through

$$\phi(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \mathbf{H}_{\boldsymbol{\mu},\mathbf{J}} \triangleq \begin{bmatrix}\mathbf{J} & \boldsymbol{\mu}\\ \mathbf{0}^T & 1\end{bmatrix}, \tag{1}$$

a Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is uniquely mapped to the matrix $\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}} \in UT^+(d+1)$, where $\mathbf{J} = \mathbf{L}^{-T}$. However, the embedding form (1) does not suit for backpropagation due to the Cholesky decomposition and matrix inverse.

The matrix $\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}}$ can be further mapped to a unique SPD matrix based on matrix polar decomposition and Lie group isomorphism. Let $\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}} = \mathbf{S}_{\boldsymbol{\mu},\mathbf{J}}\mathbf{Q}_{\boldsymbol{\mu},\mathbf{J}}$ be the left polar decomposition of $\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}}$, where $\mathbf{S}_{\boldsymbol{\mu},\mathbf{J}}$ and $\mathbf{Q}_{\boldsymbol{\mu},\mathbf{J}}$ be an $(d+1)\times(d+1)$ SPD matrix and an orthogonal matrix of determinant one, respectively. The mapping can be written as

$$\psi(\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}}) = \mathbf{S}_{\boldsymbol{\mu},\mathbf{J}} = \begin{bmatrix}\boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu}\\ \boldsymbol{\mu}^T & 1\end{bmatrix}^{\frac{1}{2}}, \tag{2}$$

and $\mathbf{Q}_{\boldsymbol{\mu},\mathbf{J}}$ is the closest orthogonal matrix to $\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}}$, i.e.,

$$\mathbf{Q}_{\boldsymbol{\mu},\mathbf{J}} = \min_{\mathbf{R}\in O(d+1)} \|\mathbf{H}_{\boldsymbol{\mu},\mathbf{J}} - \mathbf{R}\|_F,$$

where F indicates the Frobenius norm and $O(d+1)$ denotes the set of $(d+1)\times(d+1)$ orthogonal matrices. Through the above consecutive mappings, our introduced Gaussian embedding can be represented as follows:

$$(\psi\circ\phi)(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \begin{bmatrix}\boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu}\\ \boldsymbol{\mu}^T & 1\end{bmatrix}^{\frac{1}{2}}. \tag{3}$$

Most works study Gaussian embedding based on the structure of Riemannian manifold of Gaussians. Nakayama *et al*. [28] embed Gaussians in a flat manifold by taking an affine coordinate system. In [8], Gaussian is mapped to a unique positive definite lower triangular affine transform (PDLTAT) matrix whose space forms an affine group. The methods in Calvo *et al*. [4] and Lovri'c *et al*. [26] respectively embed the space of Gaussian in the Siegel group and the Riemannian symmetric space, identifying a Gaussian as a unique SPD matrix. Note that, different from the aforementioned methods which only consider the Riemannian manifold structure of $\mathcal{G}(d)$, our introduced embedding method (3) makes use of the Lie group structure, i.e., the geometric structure of Riemannian manifold and the algebraic structure of smooth group. The Gaussian embedding strategy (3) is not only suitable for backpropagation but also produces better performances as compared in Section 4.3.2.

3.2. Global Gaussian Embedding Layer

Next, we will construct our global Gaussian embedding layer according to the embedding form (3). To facilitate implementation of this layer, we decompose it into two sub-layers: matrix partition sub-layer and square rooted SPD matrix sub-layer, as illustrated in Figure 1.

3.2.1 Matrix Partition Sub-layer

We denote $\mathbf{Y} = f_{MPL}(\mathbf{X}) \triangleq \begin{bmatrix} \boldsymbol{\Sigma} + \boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu} \\ \boldsymbol{\mu}^T & 1 \end{bmatrix}$. Obviously the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ are entangled. The purpose of this sub-layer is to decouple $\mathbf{Y}$ and explicitly write it as the function of input features $\mathbf{X}$. We first note that there exists the identity $\boldsymbol{\Sigma} = \frac{1}{N}\mathbf{X}^T\mathbf{X} - \boldsymbol{\mu}\boldsymbol{\mu}^T$. After some elementary manipulations, we have

$$\begin{aligned} \mathbf{Y} =& f_{MPL}(\mathbf{X}) \\ =& \frac{1}{N}\mathbf{A}\mathbf{X}^T\mathbf{X}\mathbf{A}^T + \frac{2}{N}\left(\mathbf{A}\mathbf{X}^T\mathbf{1}\mathbf{b}^T\right)_{sym} + \mathbf{B}. \end{aligned} \tag{4}$$

In the above equation, $\mathbf{A} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0}^T \end{bmatrix}$ where $\mathbf{I}$ is the $d \times d$ identity matrix and $\mathbf{0}$ is $d-$dimensional zero vector, $\mathbf{b} = [0, \ldots, 0, 1]$ is $(d+1)-$dimensional vector with all elements being zero except the last one which is equal to one, $\mathbf{1}$ is $N-$dimensional vector with all elements being one, and finally $\mathbf{B} = \begin{bmatrix} \mathbf{O} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$ where $\mathbf{O}$ is $d \times d$ zero matrix. The notation $\mathbf{P}_{sym} = \frac{1}{2}(\mathbf{P} + \mathbf{P}^T)$ denotes the symmetrization of $\mathbf{P}$. After such manipulations, the derivative of $\mathbf{Y}$ with respect to $\mathbf{X}$ is straightforward.

3.2.2 Square-rooted SPD Matrix Sub-layer

The purpose of this sub-layer is to compute the square root of SPD matrix $\mathbf{Y}$, i.e., $\mathbf{Z} = f_{ESRL}(\mathbf{Y}) \triangleq \mathbf{Y}^{\frac{1}{2}}$. It is well-known that an SPD matrix is diagonalizable by SVD and the diagonal elements are positive real numbers. Specifically, $\mathbf{Y}$ has SVD

$$\mathbf{Y} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T, \tag{5}$$

where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \cdots, \lambda_{d+1})$ is the diagonal matrix of the eigenvalues λ_i in decreasing order and $\mathbf{U} = [\mathbf{u}_1 \ \cdots \ \mathbf{u}_{d+1}]$ is an orthogonal matrix whose columns consist of normalized eigenvectors $\mathbf{u}_i$ corresponding to the eigenvalues λ_i. As such the square root of $\mathbf{Y}$ can be computed conveniently as follows:

$$\mathbf{Z} = f_{ESRL}(\mathbf{Y}) = \mathbf{U}\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{U}^T, \tag{6}$$

where $\boldsymbol{\Lambda}^{\frac{1}{2}} = \operatorname{diag}(\lambda_1^{\frac{1}{2}}, \cdots, \lambda_{d+1}^{\frac{1}{2}})$ is computed as element-wise square root of the eigenvalues. Combining matrix partition sub-layer (4) with square-rooted SPD matrix sub-layer (6), we can accomplish the Gaussian embedding (3). Next, we will show backpropagation for the proposed global Gaussian embedding layer.

3.3. Backpropagation for Global Gaussian Embedding Layer

To implement backpropagation for global Gaussian embedding layer, we need to compute $\frac{\partial f(\mathbf{Z})}{\partial \mathbf{X}}$, where $f(\mathbf{Z})$ denotes a sub-network of G^{2}DeNet whose input and output are $\mathbf{Z}$ and loss function, respectively. In this paper, $\frac{\partial f(\mathbf{Z})}{\partial \mathbf{X}}$ can be achieved by two steps. In the first step, we compute $\frac{\partial f(\mathbf{Z})}{\partial \mathbf{Y}}$. For brevity, we use f instead of $f(\mathbf{Z})$ in the following.

Compute $\frac{\partial f}{\partial \mathbf{Y}}$ Note that $\mathbf{Y}$ is an SPD matrix, and its SVD can be written as $\mathbf{Y} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$. The chain rule of this step is given by

$$\frac{\partial f}{\partial \mathbf{Y}} : d\mathbf{Y} = \frac{\partial f}{\partial \mathbf{U}} : d\mathbf{U} + \frac{\partial f}{\partial \boldsymbol{\Lambda}} : d\boldsymbol{\Lambda}, \tag{7}$$

where $\mathbf{U} : \mathbf{V} = \operatorname{tr}(\mathbf{U}^T\mathbf{V})$ denotes the trace of $\mathbf{U}^T\mathbf{V}$, and $d\mathbf{U}$ denotes the variation of $\mathbf{U}$. By taking variation of $\mathbf{Y}$ we have $d\mathbf{Y} = d\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T + \mathbf{U}d\boldsymbol{\Lambda}\mathbf{U}^T + \mathbf{U}\boldsymbol{\Lambda}d\mathbf{U}^T$. Note that $\mathbf{U}$ is orthogonal, and after some manipulations, we can derive

$$\begin{aligned} d\mathbf{U} &= 2\mathbf{U}(\mathbf{K}^T \odot (\boldsymbol{\Lambda}^T\mathbf{U}^T d\mathbf{Y}\mathbf{U})_{sym}), \\ d\boldsymbol{\Lambda} &= (\mathbf{U}^T d\mathbf{Y}\mathbf{U})_{diag}, \end{aligned} \tag{8}$$

where $(\cdot)_{diag}$ indicates matrix diagonalization, $\odot$ is the Hadamard product, and $\mathbf{K}$ is a square matrix with its element $\mathbf{K}_{ij} = 1/(\lambda_i^2 - \lambda_j^2)$ if $i \neq j$ and $\mathbf{K}_{ij} = 0$ otherwise. Substituting Eq. (8) into Eq. (7), we achieve

$$\frac{\partial f}{\partial \mathbf{Y}} = \mathbf{U}\Big(2\boldsymbol{\Lambda}\Big(\mathbf{K}^T \odot \Big(\mathbf{U}^T\frac{\partial f}{\partial \mathbf{U}}\Big)\Big)_{sym} + \Big(\frac{\partial f}{\partial \boldsymbol{\Lambda}}\Big)_{diag}\Big)\mathbf{U}^T. \tag{9}$$

The derivation of Eq. (9) is first given in [14, Prop. 1] and readers may refer to [15] for more details.

We proceed to compute $\frac{\partial f}{\partial \mathbf{U}}$ and $\frac{\partial f}{\partial \boldsymbol{\Lambda}}$. Here the chain rule is give by

$$\frac{\partial f}{\partial \mathbf{Z}} : d\mathbf{Z} = \frac{\partial f}{\partial \mathbf{U}} : d\mathbf{U} + \frac{\partial f}{\partial \boldsymbol{\Lambda}} : d\boldsymbol{\Lambda}. \tag{10}$$

We substitute the variation $d\mathbf{Z} = 2(d\mathbf{U}\boldsymbol{\Lambda}^{-\frac{1}{2}}\mathbf{U}^T)_{sym} + \frac{1}{2}\mathbf{U}\boldsymbol{\Lambda}^{-\frac{1}{2}}d\boldsymbol{\Lambda}\mathbf{U}^T$ into the above equation and can derive

$$\frac{\partial f}{\partial \mathbf{U}} = 2\Big(\frac{\partial f}{\partial \mathbf{Z}}\Big)_{sym}\mathbf{U}\boldsymbol{\Lambda}^{\frac{1}{2}}, \ \frac{\partial f}{\partial \boldsymbol{\Lambda}} = \frac{1}{2}\boldsymbol{\Lambda}^{-\frac{1}{2}}\mathbf{U}^T\frac{\partial f}{\partial \mathbf{Z}}\mathbf{U}. \tag{11}$$

Compute $\frac{\partial f}{\partial \mathbf{X}}$ In the second step, we compute the partial derivative associated with the matrix partition sub-layer. The chain rule involved is

$$\frac{\partial f}{\partial \mathbf{X}} : d\mathbf{X} = \frac{\partial f}{\partial \mathbf{Y}} : d\mathbf{Y}. \tag{12}$$

Method	Gaussian Embedding
Nakayama *et al.* [28]	$\mathbf{z} = [\, vec(\boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T), \boldsymbol{\mu}^T \,]^T$
Calvo *et al.* [4] or Lovrić *et al.* [26]	$\mathbf{Z} = \begin{bmatrix} \boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu} \\ \boldsymbol{\mu}^T & 1 \end{bmatrix}$
[4, 26] + Log-Euclidean [2]	$\mathbf{Z} = \log \begin{bmatrix} \boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu} \\ \boldsymbol{\mu}^T & 1 \end{bmatrix}$
Ours	$\mathbf{Z} = \begin{bmatrix} \boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu} \\ \boldsymbol{\mu}^T & 1 \end{bmatrix}^{\frac{1}{2}}$

Table 1. Comparison of different Gaussian embedding methods. vec indicates the vectorization operation of a matrix.

We take the variation of $\mathbf{Y}$ with respect to $\mathbf{X}$ and substitute it into Eq. (12). After some arrangements, we achieve

$$\frac{\partial f}{\partial \mathbf{X}} = \frac{2}{N}\Big(\mathbf{X}\mathbf{A}^T + \mathbf{1}\mathbf{b}^T\Big)\Big(\frac{\partial f}{\partial \mathbf{Y}}\Big)_{sym}\mathbf{A}. \tag{13}$$

In summary, for the proposed global Gaussian embedding layer, the forward propagation can be performed via Eqs. (4) and (6), while the backpropagation can be achieved by Eq. (13), Eq. (9) and Eq. (11). Our layer can be plugged into various CNN architectures (e.g., AlexNet [21] and VGG-VD-Net [34]) in an end-to-end manner. In practice, we insert our layer after the last convolutional layer (with ReLU operation).

3.4. G^{2}DeNet Based Other Embedding Methods

Finally, we introduce three other embedding methods which can be used in our G^{2}DeNet methodology. The comparison of embedding forms of different Gaussian embedding methods are listed in Table 1.

The backpropagation rule for [28] is given by

$$\frac{\partial f}{\partial \mathbf{X}} = \frac{1}{N}\Big(2\mathbf{X}\Big(mat\left(\frac{\partial f}{\partial \mathbf{z}}\right)_{1:d^2}\Big)_{sym} + \mathbf{1}\left(\frac{\partial f}{\partial \mathbf{z}}\right)^T_{d^2+1:d^2+d}\Big) \tag{14}$$

where $\mathbf{y}_{1:i}$ denotes the vector formed by entries $1, \ldots, i$ in vector $\mathbf{y}$ and $mat(\mathbf{y})$ denotes reshaping of vector $\mathbf{y}$ to a square matrix which has the same number of elements in $\mathbf{y}$. The partial derivative associated with [4, 26] is

$$\frac{\partial f}{\partial \mathbf{X}} = \frac{2}{N}\Big(\mathbf{X}\mathbf{A}^T + \mathbf{1}\mathbf{b}^T\Big)\Big(\frac{\partial f}{\partial \mathbf{Z}}\Big)_{sym}\mathbf{A}. \tag{15}$$

The derivation of backpropagation formulas for [4, 26] plus Log-Euclidean framework [2] is similar to those described in Section 3.3. The partial derivatives $\frac{\partial f}{\partial \mathbf{X}}$ and $\frac{\partial f}{\partial \mathbf{Y}}$ are the same as Eq. (13) and Eq. (9), respectively, but $\frac{\partial f}{\partial \mathbf{U}}$ and $\frac{\partial f}{\partial \boldsymbol{\Lambda}}$ take different forms as follows:

$$\frac{\partial f}{\partial \mathbf{U}} = 2\Big(\frac{\partial f}{\partial \mathbf{Z}}\Big)_{sym}\mathbf{U}\log(\boldsymbol{\Lambda}), \frac{\partial f}{\partial \boldsymbol{\Lambda}} = \boldsymbol{\Lambda}^{-1}\mathbf{U}^T\frac{\partial f}{\partial \mathbf{Z}}\mathbf{U}. \tag{16}$$

4. Experiments

In this section, we conduct two parts of experiments to evaluate our method: large-scale region classification on MS-COCO 2014 dataset [23] and challenging fine-grained recognition on Birds-200-2011 [39], FGVC-Aircraft [27] and FGVC-Cars [20]. We also verify the effects of different training methods and Gaussian embedding strategies on the proposed method. We implement our G^{2}DeNet by using the MatConvNet package [38], and run the programs on a PC equipped with a single NVIDIA Titan X GPU and 64G RAM. As suggested in [14], we use SVD rather than EIG for computing square-rooted SPD matrix because SVD is numerically more stable, and implement the global Gaussian embedding layer on CPU in double precision due to limited support of current GPU library for SVD or EIG, and less accurate gradients of the structured layer induced by the single precision. For numerical stability, we add a small positive number 1e-3 throughout the paper to the diagonal entries of covariance matrices. More implementation details are described in the following subsections.

4.1. Region Classification on MS-COCO

The MS-COCO dataset used for region classification task includes more than 890k segmented instances from 80 classes, divided into about 600k training instances and 290k validation ones. In this part of experiments, we mainly compare our G^{2}DeNet with its counterpart DeepO$_2$P [14]. For fair comparison, we exploit identical experimental settings with [14] and use the code released by the authors [1], where we replace the global O$_2$P layer with the proposed global Gaussian embedding layer.

We implement G^{2}DeNet which indicates the proposed layer is directly connected to a softmax layer, and G^{2}DeNet-FC indicating the proposed layer connects to two fully-connected layers followed by a softmax layer, as in AlexNet. Both of the two networks are initialized with AlexNet model pre-trained on ImageNet dataset [7]. We also implement G^{2}DeNet-FC with random initialization (training from scratch), which is called G^{2}DeNet-FC (S). We compare them with the corresponding counterparts which use the global O$_2$P layer. As in DeepO$_2$P, the cropped images are resized to have the largest sides of 200 pixels, and translation jittering and random, horizontal flipping are used. We perform training using stochastic gradient descent with a momentum of 0.9 and a batch size of 100. The G^{2}DeNet, G^{2}DeNet-FC and G^{2}DeNet-FC (S) are trained with 15, 20 and 50 epoches where learning rates are set as ones in DeepO^{2}P. The classification errors of various methods on the validation set are reported for comparison.

The convergence curve of the proposed G^{2}DeNet-FC is

[1]The code is available at http://www.maths.lth.se/matematiklth/personal/sminchis/code/

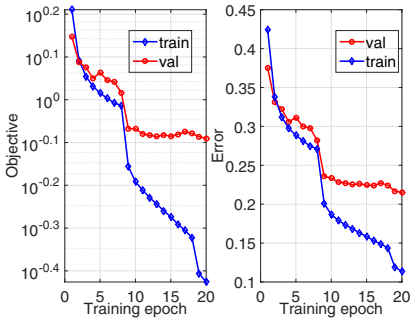

Figure 2. Convergence curve of our G^{2}DeNet-FC on MS-COCO.

	AlexNet-FC	DeepO$_2$P [14]	DeepO$_2$P-FC (S) [14]	DeepO$_2$P-FC [14]
Err.	25.3	28.6	28.9	25.2
	DMMs-FC [29]	G^{2}DeNet (Ours)	G^{2}DeNet-FC (S) (Ours)	G^{2}DeNet-FC (Ours)
Err.	24.6	24.4	22.6	**21.5**

Table 2. Classification errors (%) on the MS-COCO benchmark. The results for all the methods indicated by 'AlexNet-FC' or 'DeepO$_2$P'are duplicated from [14].

illustrated in Figure 2. We achieve the lowest classification error 21.5% at epoch 20. We note that classification error of G^{2}DeNet-FC tends to descend after the final epoch, and so more training epoches may bring further improvement. The training and test time of G^{2}DeNet-FC are about 3 hours and 30 minutes per epoch, respectively. Our G^{2}DeNet-FC shares similar time complexity with DeepO$_2$P-FC.

Comparison results on MS-COCO are listed in Table 2. The AlexNet-FC method indicates the fine-tuned AlexNet where the last layer is replaced by 80-way softmax layer. We also implement the DMMs method [29] by inserting a DMMs layer of 4,096 frequencies into AlexNet with the same settings as G^{2}DeNet-FC, which is called DMMs-FC. According to Table 2, we have the following discourse: (1) our G^{2}DeNet-FC achieves the best results, improving both DeepO$_2$P-FC and DMMs-FC methods by a large margin (3.7% and 3.1%); (2) G^{2}DeNet always outperforms DeepO$_2$P in same settings under different scenarios, which we attribute to the superiority of the proposed global Gaussian embedding layer over the O$_2$P layer; (3) our G^{2}DeNet-FC also is far superior to AlexNet-FC, demonstrating appropriate insertion of a probability distribution as an image representation into deep CNNs is very beneficial.

4.2. Fine-grained Recognition

The second part of experiments is conducted on three fine-grained image benchmarks, on which the recognition task is challenging due to large intra-class variation and small inter-class differences. We mainly compare with BCNN [24], one of the counterparts of our G^{2}DeNet, which is a state-of-the-art fine-grained recognition method. Specifically, we compare with BCNN [D,D] where the two CNN models involved are identical (i.e., VGG-VD16) and most of the best results are achieved. Note that in this case the bilinear pooling method shares the same CNN model, and leads to the second-order, non-central moment of convolutional features. For fair comparison, we adopt exactly the same experimental settings with BCNN wherever possible, e.g., two-stage training manner, hyper-parameters, data processing and SVM training and test [2]. To implement our method, we replace the bilinear layer with the proposed global Gaussian embedding layer.

4.2.1 Birds-200-2011

The Birds-200-2011 [39] is a challenging dataset, including 11,788 images from 200 bird species. The fixed training and test split is provided to evaluate different methods. On this dataset, the part annotations (Parts) and the bounding boxes (BBox) usually are considered to develop recognition methods in training or test. Following the protocols used in BCNN, we evaluate our G^{2}DeNet in two cases, i.e., training and testing G^{2}DeNet with or without bounding boxes.

The results of different methods are listed in Table 3. We first compare our G^{2}DeNet with FC-CNN, FV-CNN and BCNN in the same experimental settings. The FC-CNN extracts the outputs of the penultimate fully connected layer as image representations. The FV-CNN [6] performs encoding and pooling of features from the last convolutional layer with Fisher vector (FV) [32] method, achieving promising results on many image recognition tasks. The BCNN obtains state-of-the-art performance by pooling of outer products of the outputs from the last convolutional layer (with the ReLU operation) of two CNN models [24]. These representations are fed to one-vs-all SVM classifiers for training and test. In the case of no bounding boxes, our G^{2}DeNet outperforms FC-CNN, FV-CNN and BCNN by 16.7%, 12.4% and 3.1%, respectively. When bounding boxes are used, the performance of all methods can be improved and G^{2}DeNet is still better than FC-CNN, FV-CNN and BCNN by 11.2%, 10.1% and 2.5%, respectively. The significant improvements over the three methods show the superiority of our global Gaussian embedding layer.

We also compare with six recently proposed methods, which, to our best knowledge, reported the previous best results without exploiting extra training data [3]. RAID-G [40] presented a robust infinite dimensional Gaussian descriptor based on pretrained VGG-VD19 model (no finetuning), getting 82.1% in accuracy without parts and BBoxes. PG-Alignment [18] generated parts for bird images by using co-segmentation and alignment in an unsupervised manner.

[2]We use the source code released by the authors of [24], available at https://bitbucket.org/tsungyu/bcnn-package.

[3]One very recent work reported an accuracy of 92.3% by using large scale additional annotation bird images from the web [19].

Methods	Train		Test		Pre-trained CNN models	Accuracy (%)
	BBox	Parts	BBox	Parts		
PG-Alignment [18]	✓		✓		VGG-VD19	82.8
RAID-G [40]					VGG-VD19	82.1
ST-CNN [16]					Inception+BN	84.1
PD+FC+SWFV-CNN [42]					VGG-VD16	84.5
SPDA-CNN+ensemble [41]	✓	✓	✓		VGG-VD16 + AlexNet	85.1
PN-CNN [3]	✓	✓	✓	✓	AlexNet	85.4
FC-CNN [D] (w/ ft)					VGG-VD16	70.4
FC-CNN [D] (w/ ft)	✓		✓		VGG-VD16	76.4
FV-CNN [D] (w/ ft) [6]					VGG-VD16	74.7
FV-CNN [D] (w/ ft) [6]	✓		✓		VGG-VD16	77.5
BCNN [D,D] (w/ ft) [24]					VGG-VD16	84.0
BCNN [D,D] (w/ ft) [24]	✓		✓		VGG-VD16	84.8
BCNN [D,M] (w/ ft) [24]	✓		✓		VGG-VD16 + VGG-M	85.1
G^2DeNet (Ours)					VGG-VD16	**87.1**
G^2DeNet (Ours)	✓		✓		VGG-VD16	**87.6**

Table 3. Classification accuracies of different methods with various experimental protocols on Birds-200-2011 dataset. 'BBox' and 'Parts' indicate bounding boxes and parts, respectively. The results of FC-CNN, FV-CNN and BCNN are duplicated from [24]. The results of other methods are respectively from original papers.

Combining bounding boxes and fine-tuned VGG-VD19 model, PG-Alignment achieved 82.8% in accuracy. ST-CNN [16] introduced a trainable Spatial Transformer (ST) module for overcoming lack of spatial invariance of existing CNN architectures. The fine-tuned ST-CNN based on the Inception architecture with batch normalization [13] obtained 84.1% in accuracy. Zhang *et al*. [42] proposed a part detector (PD) while considering filter responses, and represented the bag of parts using spatially weighted (SW) FV-CNN and FC-CNN. They reported an accuracy of 84.5%. The semantic part detection and abstraction CNN (SPDA-CNN) [41] developed an end-to-end architecture containing two sub-networks which performed semantic parts detection and recognition in a unified framework. SPDA-CNN achieved an accuracy of 85.1% with an ensemble of VGG-VD16 model and AlexNet. Branson *et al*. [3] proposed a pose normalized deep convolutional neural network (PN-CNN) to locate and normalize image patches, while employing a deep CNN to extract features for patch representation. By using both part annotations and bounding boxes, PN-CNN achieved 85.4% in accuracy.

Our G^2DeNet achieves the best results among all reported methods. Compared with ST-CNN, our G^2DeNet produces orderless representations which does not explicitly consider the spatial invariance, but outperforming ST-CNN which performs the spatial transformations of features. The methods [18, 42, 41, 3] all exploit part detectors or ground truth part annotations, which often can significantly improve recognition accuracies for fine-grained recognition task. Even without bounding boxes and part detector, our G^2DeNet achieves $1.7\% \sim 5.0\%$ gains over them. The competitive results show that our G^2DeNet is a very discriminative and robust image representation. Integration of part annotation with our G^2DeNet may further improve the performance, which will be our future work.

Methods	Accuracy (%)	
	Aircraft	Cars
FC-CNN (VGG-VD16)	74.1	79.8
FV-CNN (VGG-VD16) [6]	77.6	85.7
BCNN (VGG-VD16)[14]	84.1	90.6
BCNN (VGG-VD16 + VGG-M) [14]	83.9	91.3
G^2DeNet (Ours, w/o BBox)	**89.0**	**92.5**
Other Methods	75.9 [5]	90.5 [43]
	80.7 [10]	**92.6** [18]

Table 4. Classification accuracies of various methods on FGVC-Aircraft and FGVC-Cars benchmarks.

4.2.2 FGVC-Aircraft

The FGVC-aircraft dataset [27] is a part of the FGComp 2013 challenge, which consists of 10,000 images across 100 aircraft classes. Comparison with birds dataset, the inter-class variation of airplanes is more subtle, and in the images the airplanes fill up larger regions but with more clear background. We exploit the fixed train/test split provided by the dataset developers, and compare with FC-CNN, FV-CNN, BCNN with VGG-VD16 model, and several other methods.

The results of different methods are listed in Table 4 (middle column). We can see that our G^2DeNet is better than its counterpart BCNN by 4.9%, and outperforms FV-CNN and FC-CNN by 11.4% and 14.9%, respectively. As we employ the same CNN model (i.e., VGG-VD16) with FC-CNN, FV-CNN and BCNN, we attribute the improvements to the proposed global Gaussian embedding layer. Finally, we note that our G^2DeNet outperforms the previous methods [5, 10] by a large margin.

4.2.3 FGVC-Cars

The FGVC-Cars dataset [20] is also presented as a part of the FGComp 2013 challenge, containing 16,185 images from 196 car categories. Following the commonly used settings, we adopt the provided roughly 50-50 split by divid-

ing the data into 8,144 training images and 8,041 test images. We also compare with FC-CNN, FV-CNN, BCNN and the other two state-of-the-art methods. The results are reported in Table 4 (right-most column). It can be seen that our G^{2}DeNet outperforms BCNN by 1.9% when VGG-VD16 is used. Combining VGG-VD16 and VGG-M, BCNN improves but G^{2}DeNet still has 1.2% gains. Meanwhile, G^{2}DeNet performs better than recently reported result [43], and is comparable to the previous best result [18] where the bounding boxes are employed.

4.3. Ablation Experiments and Analysis

Finally, we employ Birds-200-2011 dataset without BBox to analyze the effects of different training methods and Gaussian embedding strategies on G^{2}DeNet. Here, the experimental settings are the same as those in Section 4.2.

4.3.1 Training Methods

Firstly, we conduct experiments using three kinds of training methods based on VGG-VD16 model for our proposed network. The first one (*VD16-NoTr*) combines global Gaussian embedding layer with the VGG-VD16 model pre-trained on ImageNet dataset in a non-end-to-end manner, which can be seen as G^{2}DeNet without any training. For the second, we fine-tune VGG-VD16 model on birds dataset, then combine global Gaussian embedding layer with the fine-tuned VGG-VD16. This method is called *VD16-FT*, which can be seen as training G^{2}DeNet in a non-end-to-end manner. The last one is our *G^{2}DeNet*. We initialize it with VGG-VD16 model pre-trained on ImageNet dataset, then train our G^{2}DeNet in an end-to-end manner. The results of different training methods are illustrated in Figure 3. Our G^{2}DeNet outperforms VD16-NoTr and VD16-FT by 5.9% and 3.6%, respectively. It shows that plugging the global Gaussian embedding layer into the deep CNN trained end-to-end is much better than those with no training and training separately, and it also demonstrates the effectiveness of our structural backpropagation method.

4.3.2 Gaussian Embedding

To show the advantage of our Gaussian embedding strategy in G^{2}DeNet, we compare with the three other kinds of Gaussian embedding methods as described in Section 3.4. The results of different Gaussian embedding methods are listed in Table 5. From it we can see that our introduced embedding method achieves the best performance, outperforming the competing methods by $3\% \sim 3.6\%$. The performance gains of our embedding method over [28] and [4, 26] may be ascribed to the fact that ours appropriately uses the Lie group structure of Gaussians, while the latter two only consider the manifold structure. The embedding matrix in [4, 26] is symmetric positive definite, and

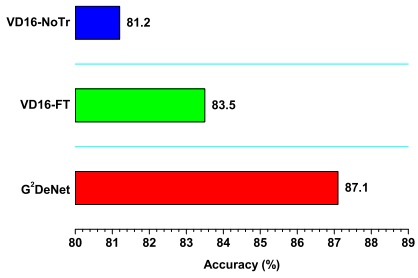

Figure 3. Effects of different training methods on G^{2}DeNet using VGG-VD16 on Birds-200-2011 dataset.

can be further subject to matrix logarithm [2], which, however, produces unsatisfactory results. From the perspective of computing, [4, 26] keep the eigenvalues of $\begin{bmatrix} \boldsymbol{\Sigma}+\boldsymbol{\mu}\boldsymbol{\mu}^T & \boldsymbol{\mu} \\ \boldsymbol{\mu}^T & 1 \end{bmatrix}$ as they are, while [4, 26] + Log-Euclidean [2] and ours perform nonlinear scaling of the eigenvalues by logarithm and square root, respectively. We conjuncture that the nonlinear scaling can be seen as a kind of eigenvalues normalization, and the square root may be more favorable than the logarithm in such scenarios. The above analysis may account for why different embedding strategies perform differently but this issue needs further study in the future.

Method	Acc. (%)
Nakayama *et al.* [28]	83.5
Calvo *et al.* [4] or Lovrić *et al.* [26]	84.1
Calvo *et al.* [4] or Lovrić *et al.* + Log-Euclidean [2]	83.8
Ours	**87.1**

Table 5. Comparison of different Gaussian embedding methods for the G^{2}DeNet methodology on Birds-200-2011 dataset.

5. Conclusion

This paper proposed to plug a trainable layer of a global Gaussian distribution as an image representation into deep CNN architectures in an end-to-end learning fashion. It can capture discriminative first- and second-order image characteristics while appropriately utilize the structures of geometry and smooth group of Gaussians. The competitive performance on large-scale region classification and challenging fine-grained recognition tasks demonstrate the effectiveness of our proposed method. As far as we know, we are among the first who explicitly combined parametric statistical modeling with deep CNNs in an end-to-end manner. This may motivate interests and efforts in plugging other parametric distributions into CNNs, e.g., generalized Gaussian distribution [30]. The proposed global Gaussian embedding layer is modular and is of no parameter to learn, readily applicable to AlexNet or VGG-Net, and combining this layer with other CNN models (e.g., Inception [36] and ResNet [12]) is our future research. We will also study other applications of the proposed method, e.g., image retrieval.

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, 3

[2] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Fast and simple calculus on tensors in the Log-Euclidean framework. In *MICCAI*, 2005. 2, 5, 8

[3] S. Branson, G. V. Horn, P. Perona, and S. J. Belongie. Improved bird species recognition using pose normalized deep convolutional nets. In *BMVC*, 2014. 7

[4] M. Calvo and J. M. Oller. A distance between multivariate normal distributions based on an embedding into the Siegel group. *JMVA*, 35(2):223–242, 1990. 3, 5, 8

[5] Y. Chai, V. Lempitsky, and A. Zisserman. Symbiotic segmentation and part localization for fine-grained categorization. In *ICCV*, 2013. 7

[6] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and segmentation. In *CVPR*, 2015. 1, 6, 7

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[8] L. Gong, T. Wang, and F. Liu. Shape of Gaussians as feature descriptors. In *CVPR*, 2009. 3

[9] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*, 2014. 1

[10] P. H. Gosselin, N. Murray, H. Jégou, and F. Perronnin. Revisiting the Fisher vector for fine-grained classification. *Pattern Recogn. Lett.*, 49:92–98, 2014. 7

[11] A. Gretton, K. Borgwardt, M. Rasch, B. Schlkopf, and A. Smola. A kernel method for the two sample problem. In *NIPS*, 2007. 1

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

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

[14] C. Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *ICCV*, 2015. 1, 2, 4, 5, 6, 7

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

[16] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015. 7

[17] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE TPAMI*, 34(9):1704–1716, 2012. 3

[18] J. Krause, H. Jin, J. Yang, and L. Fei-Fei. Fine-grained recognition without part annotations. In *CVPR*, 2015. 6, 7, 8

[19] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei. The unreasonable effectiveness of noisy data for fine-grained recognition. In *ECCV*, 2016. 6

[20] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D object representations for fine-grained categorization. In *Workshop on 3D Representation and Recognition, ICCV*, 2013. 2, 5, 7

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 5

[22] P. Li, Q. Wang, H. Zeng, and L. Zhang. Local Log-Euclidean multivariate Gaussian descriptor and its application to image classification. *IEEE TPAMI*, 39(4):803–817, 2017. 3

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 2, 5

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

[25] L. Liu, C. Shen, L. Wang, A. van den Hengel, and C. Wang. Encoding high dimensional local features by sparse coding based Fisher vectors. In *NIPS*, 2014. 1

[26] M. Lovric, M. Min-Oo, and E. A. Ruh. Multivariate normal distributions parametrized as a Riemannian symmetric space. *JMVA*, 74(1):36–48, 2000. 3, 5, 8

[27] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013. 2, 5, 7

[28] H. Nakayama, T. Harada, and Y. Kuniyoshi. Global Gaussian approach for scene categorization using information geometry. In *CVPR*, 2010. 1, 3, 5, 8

[29] J. B. Oliva, D. J. Sutherland, B. Póczos, and J. G. Schneider. Deep mean maps. *arXiv*, abs/1511.04150, 2015. 1, 6

[30] F. Pascal, L. Bombrun, J.-Y. Tourneret, and Y. Berthoumieu. Parameter estimation for multivariate generalized Gaussian distributions. *IEEE TSP*, 61(23):5960–5971, 2013. 8

[31] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*. 2008. 1

[32] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher vector: Theory and practice. *IJCV*, 105(3):222–245, 2013. 1, 3, 6

[33] G. Serra, C. Grana, M. Manfredi, and R. Cucchiara. GOLD: Gaussians of local descriptors for image representation. *CVIU*, 134:22–32, 2015. 1

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 5

[35] L. T. Skovgaard. A Riemannian geometry of the multivariate normal model. *Scand. J. Stat.*, 11(4):211–223, 1984. 3

[36] 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. 8

[37] P. Tang, X. Wang, B. Shi, X. Bai, W. Liu, and Z. Tu. Deep FisherNet for object classification. *arXiv*, abs/1608.00182, 2016. 1, 3

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

[39] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical report, 2011. 2, 5, 6

[40] 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, 6, 7

[41] H. Zhang, T. Xu, M. Elhoseiny, X. Huang, S. Zhang, A. Elgammal, and D. Metaxas. SPDA-CNN: Unifying semantic part detection and abstraction for fine-grained recognition. In *CVPR*, 2016. 7

[42] X. Zhang, H. Xiong, W. Zhou, W. Lin, and Q. Tian. Picking deep filter responses for fine-grained image recognition. In *CVPR*, 2016. 7

[43] F. Zhou and Y. Lin. Fine-grained image classification by exploring bipartite-graph labels. In *CVPR*, 2016. 7, 8