Statistically-motivated Second-order Pooling*

Kaicheng Yu$^{[0000-0002-0186-3399]}$ and Mathieu Salzmann$^{[0000-0002-8347-8637]}$

CVLab, EPFL, 1015 Lausanne, Switzerland
{kaicheng.yu,mathieu.salzmann}@epfl.ch

Abstract. Second-order pooling, a.k.a. bilinear pooling, has proven effective for deep learning based visual recognition. However, the resulting second-order networks yield a final representation that is orders of magnitude larger than that of standard, first-order ones, making them memory-intensive and cumbersome to deploy. Here, we introduce a general, parametric compression strategy that can produce more compact representations than existing compression techniques, yet outperform both compressed and uncompressed second-order models. Our approach is motivated by a statistical analysis of the network's activations, relying on operations that lead to a Gaussian-distributed final representation, as inherently used by first-order deep networks. As evidenced by our experiments, this lets us outperform the state-of-the-art first-order and second-order models on several benchmark recognition datasets.

Keywords: Second-order descriptors, convolutional neural networks, image classification

1 Introduction

Visual recognition is one of the fundamental goals of computer vision. Over the years, second-order representations, i.e., region covariance descriptors, have proven more effective than their first-order counterparts [2, 11, 31, 38] for many tasks, such as pedestrian detection [46], material recognition [9] and semantic segmentation [6]. More recently, convolutional neural networks (CNNs) have achieved unprecedented performance in a wide range of image classification problems [30, 20, 22]. Inspired by the past developments in handcrafted features, several works have proposed to replace the fully-connected layers with second-order pooling strategies, essentially utilizing covariance descriptors within CNNs [36, 24, 32, 34, 35, 29]. This has led to second-order or bilinear CNNs whose representation power surpasses that of standard, first-order ones.

One drawback of these second-order CNNs is that vectorizing the covariance descriptor to pass it to the classification layer, as done in [36, 24, 32, 34], yields a vector representation that is orders of magnitude larger than that of first-order CNNs, thus making these networks memory-intensive and subject to overfitting. While compression strategies have been proposed [13, 27], they are either nonparametric [13], thus limiting the representation power of the network,

* This work was supported in part by the Swiss National Science Foundation.

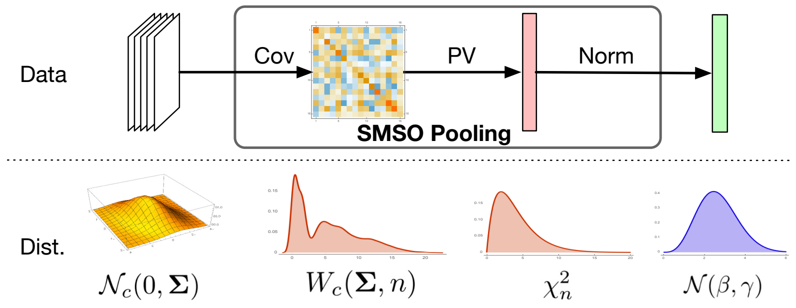
Data
Cov
PV
Norm
SMSO Pooling
Dist.
$\mathcal{N}_c(0, \mathbf{\Sigma})$
$W_c(\mathbf{\Sigma}, n)$
χ^2_n
$\mathcal{N}(\beta, \gamma)$

Fig. 1. Statistically-Motivated Second-Order (SMSO) pooling. Top: Our parametric compression strategy vectorizes a covariance matrix and normalizes the resulting vector. **Bottom:** Each of these operations yields a well-defined distribution of the data, thus resulting in a consistent framework, whose final representation follows a Gaussian distribution, as state-of-the-art first-order deep networks.

or designed for a specific classification formalism [27], thus restricting their applicability.

In this paper, we introduce a general, parametric compression strategy for second-order CNNs. As evidenced by our results, our strategy can produce more compact representations than [13, 27], with as little as 10% of their parameters, yet significantly outperforming these methods, as well as the state-of-the-art first-order [20, 42] and uncompressed second-order pooling strategies [36, 24, 32, 34].

Unlike most deep learning architectures, our approach is motivated by a statistical analysis of the network's activations. In particular, we build upon the observation that first-order networks inherently exploit Gaussian distributions for their feature representations. This is due to the fact that, as discussed in [15, 23] and explained by the Central Limit Theorem, the outputs of *linear* layers, and thus of operations such as global average pooling, follow a multivariate Gaussian distribution. The empirical success of such Gaussian distributions of feature representations in first-order deep networks motivated us to design a compression strategy such that the final representation also satisfies this property.

To this end, as illustrated by Fig. 1, we exploit the fact that the covariance matrices resulting from second-order pooling follow a Wishart distribution [26]. We then introduce a parametric vectorization (PV) layer, which compresses the second-order information while increasing the model capacity by relying on trainable parameters. We show that our PV layer outputs a vector whose elements follow χ^2 distributions, which motivates the use of a square-root normalization that makes the distribution of the resulting representation converge to a Gaussian, as verified empirically in Section 3.4. These operations rely on basic algebraic transformations, and can thus be easily integrated into any deep architecture and optimized with standard backpropagation.

We demonstrate the benefits of our statistically-motivated second-order (SMSO) pooling strategy on standard benchmark datasets for second-order models, including the Describing Texture Dataset (DTD) [9], the Material in Context (MINC) dataset [5] and the scene recognition MIT-Indoor dataset [40]. Our approach consistently outperforms the state-of-the-art second-order pooling strategies, independently of the base network used (i.e., VGG-D [42] or ResNet-50 [20]), as well as these base networks themselves. Our code is publicly available at https://github.com/kcyu2014/smsop.

2 Related Work

Visual recognition has a long history in computer vision. Here, we focus on the methods that, like us, make use of representations based on second-order information to tackle this task. In this context, the region covariance descriptors (RCDs) of [46] constitute the first attempt at leveraging second-order information. Similarly, Fisher Vectors [2] also effectively exploit second-order statistics. Following this success, several metrics have been proposed to compare RCDs [3, 37, 39, 43], and they have been used in various classification frameworks, such as boosting [12], kernel Support Vector Machines [47], sparse coding [7, 16, 49] and dictionary learning [44, 19, 33, 18]. In all these works, however, while the classifier was trained, no learning was involved in the computation of the RCDs.

To the best of our knowledge, [17], and its extension to the log-Euclidean metric [21], can be thought of as the first attempts to learn RCDs. This, however, was achieved by reducing the dimensionality of input RCDs via a single transformation, which has limited learning capacity. In [22], the framework of [17] was extended to learning multiple transformations of input RCDs. Nevertheless, this approach still relied on RCDs as input. The idea of incorporating second-order descriptors in a deep, end-to-end learning paradigm was introduced concurrently in [24] and [36]. The former introduced the DeepO$_2$P operation, consisting of computing the covariance matrix of convolutional features. The latter proposed the slightly more general idea of bilinear pooling, which, in principle, can exploit inner products between the features of corresponding spatial locations from different layers in the network. In practice, however, the use of cross-layer bilinear features does not bring a significant boost in representation power [13, 34], and bilinear pooling is therefore typically achieved by computing the inner products of the features within a single layer, thus becoming essentially equivalent to second-order pooling.

A key to the success of second-order pooling is the normalization, or transformation, of the second-order representation. In [24], the matrix logarithm was employed, motivated by the fact that covariance matrices lie on a Riemannian manifold, and that this operation maps a matrix to its tangent space, thus producing a Euclidean representation. By contrast, [36] was rather inspired by previous normalization strategies for handcrafted features [2, 38], and proposed to perform an element-wise square-root and ℓ_2 normalization after vectorization of the matrix representation. More recently, [32, 34] introduced a matrix square-root normalization strategy that was shown to outperform the other transformation techniques.

All the above-mentioned methods simply vectorize the second-order representation, i.e., covariance matrix. As such, they produce a final representation whose size scales quadratically with the number of channels in the last convolutional feature map, thus being typically orders of magnitude larger than the final representation of first-order CNNs. To reduce the resulting memory cost and parameter explosion, several approaches have been proposed to compress second-order representations while preserving their discriminative power. The first attempt at compression was achieved by [13], which introduced two strategies, based on the idea of random projection, to map the covariance matrices to vectors. These projections, however, were not learned, thus not increasing the capacity of the network and producing at best the same accuracy as the bilinear CNN of [36]. In [27], a parametric strategy was employed to reduce the dimensionality of the bilinear features. While effective, this strategy was specifically designed to be incorporated in a bilinear Support Vector Machine.

By contrast, here, we introduce a parametric compression approach that can be incorporated into any standard deep learning framework. Furthermore, our strategy is statistically motivated so as to yield a final representation whose distribution is of the same type as that inherently used by first-order deep networks. As evidenced by our experiments, our method can produce more compact representations than existing compression techniques, yet outperforms the state-of-the-art first-order and second-order models.

Note that higher-order information has also been exploited in the past [10, 28]. While promising, we believe that developing statistically-motivated pooling strategies for such higher-order information goes beyond the scope of this paper.

3 Methodology

In this section, we first introduce our second-order pooling strategy while explaining the statistical motivation behind it. We then provide an alternative interpretation of our approach yielding a lower complexity, study and display the empirical distributions of our network's representations, and finally discuss the relation of our model to the recent second-order pooling techniques.

3.1 SMSO Pooling

Our goal is to design a general, parametric compression strategy for second-order deep networks. Furthermore, inspired by the fact that first-order deep networks inherently make use of Gaussian distributions for their feature representations, we reason about the statistical distribution of the network's intermediate representations so as to produce a final representation that is also Gaussian. Note that, while we introduce our SMSO pooling strategy within a CNN formalism, it applies to any method relying on second-order representations.

Formally, let $\mathbf{X} \in \mathbb{R}^{n \times c}$ be a data matrix, consisting of n sample vectors of dimension c. For example, in the context of CNNs, $\mathbf{X}$ contains the activations of the last convolutional layer, with $n = w \times h$ corresponding to the spatial resolution of the corresponding feature map. Here, we assume $\mathbf{x}_i \in \mathbb{R}^c$ to follow a multivariate Gaussian distribution $\mathcal{N}_c(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In practice, as discussed in [15, 23] and explained by the Central Limit Theorem, this can be achieved by using a *linear* activation after the last convolutional layer, potentially followed by batch normalization [23].

Covariance Computation. Given the data matrix $\mathbf{X}$, traditional second-order pooling consists of computing a covariance matrix $\mathbf{Y} \in \mathbb{R}^{c \times c}$ as

$$\mathbf{Y} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T = \frac{1}{n-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \ , \tag{1}$$

where $\tilde{\mathbf{X}}$ denotes the mean-subtracted data matrix.

The following definition, see, e.g., [26], determines the distribution of $\mathbf{Y}$.

Definition 1. *If the elements* $\mathbf{x}_i \in \mathbb{R}^c$ *of a data matrix* $\mathbf{X} \in \mathbb{R}^{n \times c}$ *follow a zero mean multivariate Gaussian distribution* $\mathbf{x}_i \sim \mathcal{N}_c(0, \boldsymbol{\Sigma})$*, then the covariance matrix* $\mathbf{Y}$ *of* $\mathbf{X}$ *is said to follow a Wishart distribution, denoted by*

$$\mathbf{Y} = \mathbf{X}^T \mathbf{X} \sim W_c(\boldsymbol{\Sigma}, n) \ . \tag{2}$$

Note that, in the bilinear CNN [36], the mean is typically not subtracted from the data. As such, the corresponding bilinear matrix follows a form of non-central Wishart distribution [25].

Second-order Feature Compression. The standard way to use a second-order representation is to simply vectorize it [36, 24], potentially after some form of normalization [34, 32]. This, however, can yield very high-dimensional vectors that are cumbersome to deal with in practice. To avoid this, motivated by [13, 27], we propose to compress the second-order representation during vectorization. Here, we introduce a simple, yet effective, compression technique that, in contrast with [13], is parametric, and, as opposed to [27], amenable to general classifiers.

Specifically, we develop a parametric vectorization (PV) layer, which relies on trainable weights $\mathbf{W} \in \mathbb{R}^{c \times p}$, with p the dimension of the resulting vector. Each dimension j of the vector $\mathbf{z}$ output by this PV layer can be expressed as

$$z_j = \mathbf{w}_j^T \mathbf{Y} \mathbf{w}_j \ , \tag{3}$$

where $\mathbf{w}_j$ is a column of $\mathbf{W}$.

The distribution of each dimension $\mathbf{z}_j$ is defined by the following theorem.

Theorem 1 (Theorem 5.6 in [26]). *If* $\mathbf{Y} \in \mathbb{R}^{c \times c}$ *follows a Wishart distribution* $W_c(\boldsymbol{\Sigma}, n)$*, and* $\mathbf{w} \in \mathbb{R}^c$ *and* $\mathbf{w} \neq \mathbf{0}$*, then*

$$z = \frac{\mathbf{w}^T \mathbf{Y} \mathbf{w}}{\mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w}} \tag{4}$$

follows a χ^2 *distribution with degree of freedom* n*, i.e.,* $z \sim \chi^2_n$*.*

From this theorem, we can see that each output dimension of our PV layer follows a scaled χ^2 distribution $\gamma\chi^2_n$, where $\gamma = \mathbf{w}_j^T \mathbf{\Sigma} \mathbf{w}_j$, with $\mathbf{\Sigma}$ the covariance matrix of the original multivariate Gaussian distribution.

Transformation and normalization. As shown above, each dimension of our current vector representation follows a χ^2 distribution. However, as discussed above, first-order deep networks inherently exploit Gaussian distributions for their feature representations. To make our final representation also satisfy this property, we rely on the following theorem.

Theorem 2 ([50]). *If $z \sim \chi^2_n$ with degree freedom n, then*

$$z' = \sqrt{2z} \tag{5}$$

converges to a Gaussian distribution with mean $\sqrt{2n-1}$ and standard deviation $\sigma = 1$ when n is large, i.e., $z' \sim \mathcal{N}(\sqrt{2n-1}, 1)$.

Following this theorem, we therefore define our normalization as the transformation

$$\mathbf{z}'_j = \sqrt{\alpha \mathbf{z}_j} - \sqrt{2n-1}\ , \tag{6}$$

for each dimension j, where we set $\alpha = 2/(\mathbf{w}_j^T \mathbf{\Sigma} \mathbf{w}_j)$ to correspond to Theorem 2, while accounting for the factor γ arising from our parametric vectorization above. Note that other transformations, such as $\log(z)$ and $(z/n)^{1/3}$, are known to also converge to Gaussian distributions as n increases [4, 50]. We show that these operations yield similar results to the one above in Section 4.4.

Note that, according to Theorem 2, the mean and variance of the resulting Gaussian distribution are determined by the degree of freedom n, which, in our case, corresponds to the number of samples used to compute the covariance matrix in Eq. 1. Such pre-determined values, however, might limit the discriminative power of the resulting representation. To tackle this, we further rely on trainable scale and bias parameters, yielding a final representation

$$\mathbf{z}''_j = \beta_j + \gamma_j \mathbf{z}'_j\ , \tag{7}$$

where $\gamma_j > 0, \beta_j \in \mathbb{R}$. Note that this transformation is also exploited by batch normalization. However, here, we do not need to compute the batch statistics during training, since Theorem 2 tells us that the batches follow a consistent distribution.

Altogether, our SMSO pooling strategy, defined by the operations discussed above, yields a p-dimensional vector. This representation can then be passed to a classifier. It can easily be verified that the above-mentioned operations are differentiable, and the resulting deep network can thus be trained end-to-end.

3.2 Alternative Computation

Here, we derive an equivalent way to perform our SMSO pooling, with a lower complexity when p is small, as shown in the supplementary material. Note, however, that our statistical reasoning is much clearer for the derivation of Section 3.1 and was what motivated our approach.

To derive the alternative, we note that

$$\frac{1}{\sqrt{\alpha}}\mathbf{z}'_j = \sqrt{\mathbf{w}_j^T \mathbf{Y} \mathbf{w}_j} \tag{8}$$

$$= \sqrt{\mathbf{w}_j^T \left(\sum_{i=1}^{n} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T \right) \mathbf{w}_j} \tag{9}$$

$$= \sqrt{\sum_{i=1}^{n} \left(\mathbf{w}_j^T (\mathbf{x}_i - \mu)\right) \left((\mathbf{x}_i - \mu)^T \mathbf{w}_j\right)} \tag{10}$$

$$= \sqrt{\sum_{i=1}^{n} (\mathbf{w}_j^T \tilde{\mathbf{x}}_i)^2} \ , \tag{11}$$

where $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mu$.

So, in essence, given $\mathbf{X}$, $\mathbf{z}'$ can be computed by performing a 1×1 convolution, with weights shaped as $(1, 1, c, p)$ and without bias, followed by a global ℓ_2 pooling operation, and a scaling by the constant $\sqrt{\alpha}$. Note that ℓ_2 pooling was introduced several years ago [41], but has been mostly ignored in the recent deep learning advances. By contrast, feature reduction with 1×1 convolutions is widely utilized in first-order network designs [45, 20]. In essence, this mathematically equivalent formulation shows that our second-order compression strategy can be achieved without explicitly computing covariance matrices. Yet, our statistical analysis based on these covariance matrices remains valid.

3.3 Relation to Other Methods

In this section, we discuss the connections between our method and the other recent second-order pooling strategies in CNNs. In the supplementary material, we compare the computational complexity of different second-order methods with that of ours.

Normalization. Bilinear pooling (BP) [36] also proposed to make use of a square-root as normalization operation. An important difference with our approach, however, is that BP directly vectorizes the matrix representation $\mathbf{Y}$. It is easy to see that the diagonal elements of $\mathbf{Y}$ follow a χ^2 distribution, e.g., by taking $\mathbf{w}$ in Theorem 1 to be a vector with a single value 1 and the other values 0. Therefore, after normalization, some of the dimensions of the BP representation also follow a Gaussian distribution. However, the off-diagonal elements follow a variance-gamma distribution, and, after square-root normalization, will not be Gaussian, thus making the different dimensions of the final representation follow inconsistent distributions.

In [24] and [32], normalization was performed on the matrix $\mathbf{Y}$ directly, via a matrix logarithm and a matrix power normalization, respectively. As such, it is difficult to understand what distribution the elements of the final representation, obtained by standard vectorization, follow.

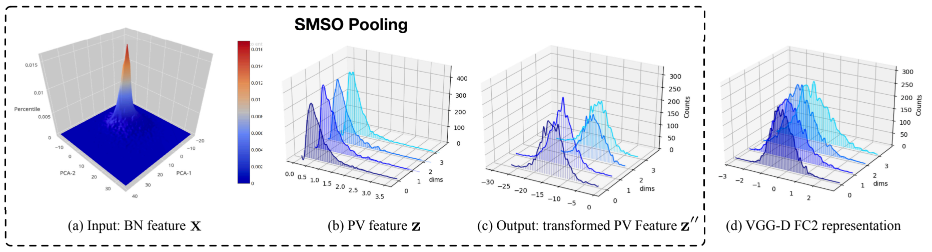

Fig. 2. Histograms of SMSO intermediate feature vectors. We plot the distribution of (a) the initial features $\mathbf{X}$, (b) the features after our PV layer $\mathbf{z}$, (c) the final representation $\mathbf{z}''$ and, for comparison, (d) first-order features after the last fully-connected layer in VGG-D [42]. Note that, as discussed in the text, these empirical distributions match the theoretical ones derived in Section 3.1, and our final representation does exploit the same type of distribution as first-order networks.

Compression. The compact bilinear pooling (CBP) of [13] exploits a compression scheme that has a form similar to ours in Eq. 3. However, in [13], the projection vectors $\mathbf{w}_j$ are random but fixed. Making them trainable, as in our PV layer, increases the capacity of our model, and, as shown in Section 4, allows us to significantly outperform CBP.

In [27], a model is developed specifically for a max-margin bilinear classifier. The parameter matrix of this classifier is approximated by a low-rank factorization, which translates to projecting the initial features to a lower-dimensional representation. As with our alternative formulation of Section 3.2, the resulting bilinear classifier can be obtained without having to explicitly compute $\mathbf{Y}$. This classifier is formulated in terms of quantities of the form $\|\mathbf{U}^T\mathbf{X}_i\|_F^2$, where $\mathbf{U}$ is a trainable low-rank weight matrix. In essence, this corresponds to removing the square-root operation from Eq. 11 and summing over all dimensions j. By contrast, our representation, ignoring the scale and bias of Eq. 7, is passed to a separate classification layer that computes a linear combination of the different dimensions with trainable weights, thus increasing the capacity of our model.

3.4 Empirical distributions of SMSO pooling

Our SMSO pooling strategy was motivated by considering the distribution of the representation at various stages in the network. Here, we study the empirical distributions of these features using the MINC dataset, discussed in Section 4, and with a model based on VGG-D. To this end, in Fig. 2, we provide a visualization of the distributions after the initial batch normalization (i.e., before computing the covariance matrix, see Section 4.1 for details), after our PV layer, and after square-root transformation with trainable scaling and bias. Specifically, for the initial features, because visualizing a Gaussian distribution in hundreds of dimensions is not feasible, we plot the distribution along the first 2 principal components. For our representations, where each dimension follows an independent Gaussian, we randomly select four dimensions and plot stacked histograms.

As expected from the theory, the initial features are close to Gaussian, and the features after our PV layer therefore follow a long-tailed χ^2 distribution. The final features, after square-root normalization, scaling and bias, are much less skewed, and thus much closer to a Gaussian distribution, thus matching the type of distribution that the final representations of state-of-the-art deep networks follow, as shown in Fig. 2(d). To further verify this, we conducted a Shapiro-Wilk test on the final representation. This resulted in a p-value of $0.19 > 0.05$, which means that the Gaussian assumption is not rejected, sustaining our claim.

4 Experiments

Here, we first provide implementation details and introduce the baseline models. We then compare our approach to these baselines on four standard benchmark datasets, and provide an ablation study of our framework.

4.1 Implementation Details

We evaluate our method on two popular network architectures: the VGG-D network of [42] (a.k.a. VGG-16) and the ResNet-50 of [20]. For all second-order models discussed below, i.e., ours and the baselines, we remove all the fully-connected layers and the last max pooling layer from VGG-D, that is, we truncate the model after the ReLU activation following *conv5-3*. For ResNet-50, we remove the last global average pooling layer and take our initial features from the last residual block. As in [32], we add a 1×1 convolutional layer to project the initial features to $c = 256$ for all the experiments. Note that this is a linear layer, and thus makes the resulting features satisfy our Gaussian assumption.

Following common practice [13, 27, 36, 32], we rely on weights pre-trained on ImageNet and use stochastic gradient descent with an initial learning rate 10 times smaller than the one used to learn from scratch, i.e., 0.001 for VGG-D and 0.01 for ResNet-50. We then divide this weight by 10 when the validation loss has stopped decreasing for 8 epochs. We initialize the weights of the new layers, i.e., the 1×1 convolution, the PV layer and the classifier, with the strategy of [14], i.e., random values drawn from a Gaussian distribution. We implemented our approach using Keras [8] with TensorFlow [1] as backend.

4.2 Baseline Models

We now describe the different baseline models that we compare our approach with. Note that the classifier is defined as a k-way softmax layer for all these models, as for ours, except for low-rank bilinear pooling, which was specifically designed to make use of a low-rank hinge loss.

Original model: This refers to the original, first-order, models, i.e., either VGG-D or ResNet-50, pre-trained on ImageNet and fine-tuned on the new data. Other than replacing the 1000-way ImageNet classification layer with a k-way one, we keep the original model settings described in [42] and [20], respectively.

Bilinear Pooling (BP) [36]: This corresponds to the original, uncompressed bilinear pooling strategy, with signed square-root and ℓ_2 normalization after vanilla vectorization. In this case, we set $c = 512$, as in the original paper, as the feature dimension before computing the second-order representation. If the original feature dimension does not match this value, i.e., with ResNet-50, we make use of an additional 1×1 convolutional layer. Note that we observed that using either 512 or 256 as feature dimension made virtually no difference on the results. We therefore used $c = 512$, which matches the original paper.

DeepO$_2$P [24]: This refers to the original, uncompressed covariance-based model, with matrix logarithm and vanilla vectorization. Again, as in the original paper, we set $c = 512$ as the feature dimension before computing the covariance matrix, by using an additional 1×1 convolutional layer when necessary.

Matrix Power Normalization (MPN) [32]: This model relies on a matrix square-root operation acting on the second-order representation. Following the original paper, we set $c = 256$ by making use of an additional 1×1 convolutional layer before second-order pooling. Note that the improved bilinear pooling of [34] has the same structure as MPN, and we do not report it as a separate baseline.

Compact bilinear pooling (CBP) [13]: We report the results of both versions of CBP: the Random Maclaurin (RM) one and the Tensor Sketch (TS) one. For both versions, we set the projection dimension to $d = 8,192$, which was shown to achieve the same accuracy as BP, i.e., the best accuracy reported in [13]. As in the original paper, we apply the same normalization as BP [36].

Low rank bilinear pooling (LRBP) [27]: This corresponds to the compression method dedicated to the bilinear SVM classifier. Following [27], we set the projection dimension to $m = 100$ and its rank to $r = 8$, and initialize the dimensionality reduction layer using the SVD of the Gram matrix computed from the entire validation set. Following the authors' implementation, we apply a scaled square-root with factor 2×10^5 after the *conv5-3* ReLU, which seems to prevent the model from diverging. Furthermore, we found that training LRBP from the weights of BP fine-tuned on each dataset also helped convergence.

4.3 Comparison to the Baselines

Let us now compare the results of our model with those of the baselines described in Section 4.2. To this end, we make use of four diverse benchmark image classification datasets, thus showing the versatility of our approach. These datasets are the Describing Texture Dataset (DTD) [9] for texture recognition, the challenging Material In Context (MINC-2500) dataset [5] for large-scale material recognition in the wild, the MIT-Indoor dataset [40] for indoor scene understanding and the Caltech-UCSD Bird (CUB) dataset [48] for fine-grained classification. DTD contains 47 classes for a total of 5,640 images, mostly capturing the texture itself, with limited surrounding background. By contrast, MINC-2500, consisting of 57,500 images of 23 classes, depicts materials in their real-world environment, thus containing strong background information and making it more challenging. MIT-Indoor contains 15,620 images of 67 different indoor scenes, and, with

Fig. 3. Sample images from DTD, MINC-2500, MIT-Indoor and CUB.

Model name	Feature dim.	# param.	DTD [9]	MIT [40]	MINC [5]	CUB [48]
VGG-D [42]	4,096	119.64M	60.11	64.51	73.01	66.12
BP [36]	2.6×10^5	3.015M	67.50	77.55	74.50	81.02
MPN [32]	$32,896$	0.752M	68.01	76.49	76.24	84.10
DeepO$_2$P [24]	2.6×10^5	3.015M	66.07	72.35	69.29	-
CBP-TS [13]	8,192	0.189M	67.71	76.83	73.28	84.00
CBP-RM [13]	8,192	0.189M	63.24	73.89	73.54	83.86
LRBP [27]	100	0.068M	65.80	73.59	69.10	84.21
SMSO (*Ours*)	64	0.013M	68.18	75.37	74.18	82.66
SMSO (*Ours*)	2,048	0.057M	**69.26**	**79.45**	**78.00**	**85.01**

Table 1. Comparison of VGG-D based models. We report the top 1 classification accuracy (in %) of the original VGG-D model, uncompressed second-order models with different normalization strategies (BP, DeepO$_2$P, MPN), second-order compression methods (CBP-TS, CBP-RM, LRBP), and our approach (SMSO) with different PV dimensions. Note that our approach significantly outperforms all the baselines despite a more compact final representation (Feature dim.) and much fewer parameters (# param is the number of trainable parameters after the last convolutional layer).

DTD, has often been used to demonstrate the discriminative power of second-order representations. The CUB dataset contains 11,788 images of 200 different bird species. In Fig. 3, we provide a few samples from each dataset. For our experiments, we make use of the standard train-test splits released with these datasets. For DTD, MIT-Indoor and CUB, we define the input size as 448×448 for all the experiments. For the large-scale MINC-2500 dataset, we use 224×224 images for all models to speed up training. Note that a larger input size could potentially result in higher accuracies [5]. For all datasets and all models, we use the same data augmentation strategy as in [36, 34].

Experiments with VGG-D. We first discuss the results obtained with the VGG-D architecture as base model. These results are reported in Table 1 for all models and all datasets. In short, our SMSO framework with PV dimension $p = 2,048$ outperforms all the baselines by a significant margin on all three datasets. In particular, our accuracy is 7% to 19% higher than the original VGG-D, with *much* fewer parameters, thus showing the benefits of exploiting second-order features. MPN is the best-performing baseline, but, besides the fact that

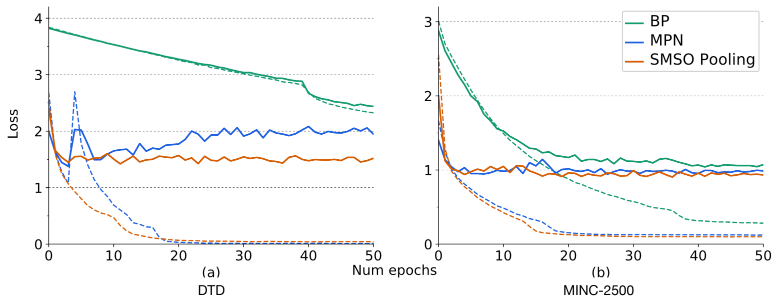

Fig. 4. Training and validation loss curves. We plot the training (dashed) and validation (solid) loss values as a function of the number of training epochs for our SMSO pooling strategy (orange), BP (green) and MPN (blue) on DTD (a) and MINC-2500 (b). Our models clearly converge faster than BP, and tend to be more stable than MPN, particularly on the smaller-scale DTD dataset.

we consistently outperform it, has a much higher computational complexity and run time, as shown in the supplementary material. The second-order compression methods (CBP and LRBP) underperform the uncompressed models on average. By contrast, even with $p = 64$, we outperform most baselines, with a model that corresponds to 10% of the parameters of the most compact baseline.

In Fig. 4, we compare the training and validation loss curves of our approach with those of the best-performing baselines, BP and MPN, on DTD and MINC. Note that our model converges much faster than BP and tends to be more stable than MPN, particularly on DTD. This, we believe, is due to the fact that we rely on basic algebraic operations, instead of the eigenvalue decomposition involved in MPN whose gradient can be difficult to compute, particularly in the presence of small or equal eigenvalues [24].

During these VGG-D based experiments, we have observed that, in practice, LRBP was difficult to train, being very sensitive to the learning rate, which we had to manually adapt throughout training. Because of this, and the fact that LRBP yields lower accuracy than uncompressed models, we do not include this baseline in the remaining experiments. We also exclude DeepO$_2$P from the next experiments, because of its consistently lower accuracy.

Experiments with ResNet-50. To further show the generality of our approach, we make use of the more recent, very deep ResNet-50 [20] architecture as base network. Table 2 provides the results of our SMSO framework with $p = 64$ and $p = 2,048$, and of the baselines. In essence, the conclusions remain unchanged; we outperform all the baselines for $p = 2,048$. Note that, here, however, the second-order baselines typically do not even outperform the original ResNet-50, whose results are significantly higher than the VGG-D ones. By contrast, our model is able to leverage this improvement of the base model and to further increase its accuracy by appropriately exploiting second-order features.

Model name	Feature dim.	# param.	DTD [9]	MIT [40]	MINC [5]	CUB [48]
ResNet-50 [20]	2,048	4K	71.45	76.45	79.12	74.51
BP [36]	32,896	752K	69.37	68.35	79.05	82.70
MPN [32]	32,896	752K	71.10	72.12	79.83	85.43
CBP-TS [13]	8,192	189K	65.30	72.60	75.91	77.35
CBP-RM [13]	8,192	189K	62.35	67.81	74.15	-
SMSO (*Ours*)	64	13K	71.03	76.31	79.17	81.98
SMSO (*Ours*)	2,048	57K	**72.51**	**79.68**	**81.33**	**85.77**

Table 2. Comparison of ResNet-50 based models. We report the top 1 classification accuracy (in %) of the original ResNet-50 model, uncompressed second-order models with different normalization strategies (BP, MPN), second-order compression methods (CBP-TS, CBP-RM), and our approach (SMSO). Note that, as in the VGG-D case, our model outperforms all the baselines, including the original ResNet-50, which is not the case of most second-order baselines. It also yields much more compact models than the second-order baselines. (# params. refers to the same quantity as in Table 1.)

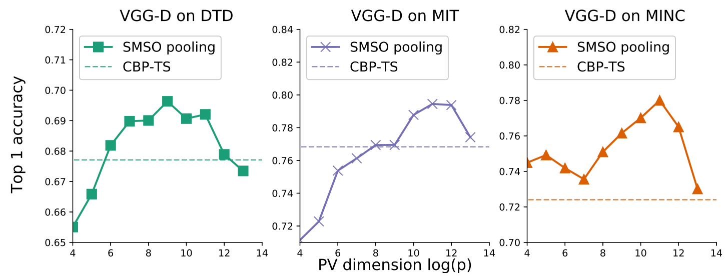

Fig. 5. Influence of the PV dimension p**.** We plot the top 1 accuracy as a function of the value p in logarithmic scale on MIT (left), MINC (middle) and DTD (right). Note that accuracy is quite stable over large ranges of p values, yielding as good results as the best-performing compression baseline (CBP-TS) with as few as $p = 64$ dimensions, corresponding to only 10% of the parameters of CBP-TS.

4.4 Ablation Study

We evaluate the influence of different components of our model on our results.

Influence of the PV dimension. In our experiments, we proposed to set $p = 2,048$ or $p = 64$. We now investigate the influence of this parameter on our results. To this end, we vary p in the range $[2^4, 2^{13}]$ by steps corresponding to a factor 2. The curves for this experiment on the validation data of the three datasets with VGG-D based models are provided in Fig. 5. Note that our model is quite robust to the exact value of this parameter, with stable accuracies outperforming the best compression baseline for each dataset over large ranges. More importantly, even with $p = 64$, our model yields as good results as the best compression method, CBP-TS, with only 10% of its parameters.

Vec.	Flatten	PV					PV		
Trans.	-	-	Sqrt	Sqrt	Sqrt	-	Sqrt	Log	$\sqrt[3]{}$
Norm.	-	-	-	β	γ	β, γ	β, γ		
Dist.	$W_n(\Sigma)$	χ_n^2	$\mathcal{N}(\mu, 1)$	$\mathcal{N}(\beta, 1)$	$\mathcal{N}(\mu, \gamma^2)$	$\gamma\chi_n^2 + \beta$	$\mathcal{N}(\beta, \gamma^2)$		
Acc.	75.97	75.32	76.32	77.12	76.47	76.14	78.00	77.86	77.17

Table 3. Comparison of different final feature distributions. We report the results of different combinations of vectorization (vec.), transformation (trans.) and normalization (norm.) strategies, yielding different final feature distributions. Here, $\mu = \sqrt{2n-1}$ from Theorem 2. Ultimately, these results show that bringing the data back to a Gaussian distribution with a trainable scale and bias yields higher accuracies.

Comparison of different distributions and transformations. We conduct experiments to compare different final feature distributions on MINC-2500 with a VGG-D based model. The results are provided in Table 3. Without our PV compression and without transformation or normalization, the resulting features follow a Wishart distribution, yielding an accuracy of 75.97%, which is comparable to BP [36]. Adding our PV layer $p = 2,048$, but not using any transformation or normalization, yields χ^2-distributed features and an accuracy similar to the previous one. This suggests that our parametric compression is effective, since we obtain similar accuracy with much fewer parameters. Including the square-root transformation, but without the additional scale and bias of Eq. 7, increases the accuracy to 76.32%. Additionally learning the scale and bias boosts the accuracy to 78.00%, thus showing empirically the benefits of Gaussian-distributed features over other distributions.

In the last two columns of Table 3, we report the results of different transformations that bring the χ^2-distributed features to a Gaussian distribution, i.e., the cubic-root and the element-wise logarithm. Note that these two transformations yield accuracies similar to those obtained with the square-root. More importantly, all transformations yield higher accuracies than not using any (76.14%), which further evidences the benefits of Gaussian-distributed features.

5 Conclusion

We have introduced a general and parametric compression strategy for second-order deep networks, motivated by a statistical analysis of the distribution of the network's intermediate representations. Our SMSO pooling strategy outperforms the state-of-the-art first-order and second-order models, with higher accuracies than other compression techniques for up to 90% parameter reduction. With a ResNet-50 base architecture, it is the only second-order model to consistently outperform the original one. While Gaussian distributions have proven effective here and for first-order models, there is no guarantee that they are truly optimal. In the future, we will study if other transformations yielding non-Gaussian distributions can help further improve second-order models.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Arandjelovic, R., Zisserman, A.: All About VLAD. In: CVPR. pp. 1578–1585 (2013)
3. Arsigny, V., Fillard, P., Pennec, X., Ayache, N.: Log-Euclidean metrics for fast and simple calculus on diffusion tensors. Magnetic Resonance in Medicine (2006)
4. Bartlett, M.S., Kendall, D.G.: The statistical analysis of variance-heterogeneity and the logarithmic transformation. Supplement to the Journal of the Royal Statistical Society **8**(1), 128–138 (1946), http://www.jstor.org/stable/2983618
5. Bell, S., Upchurch, P., Snavely, N., Bala, K.: Material Recognition in the Wild with the Materials in Context Database. In: CVPR (2015)
6. Carreira, J., Caseiro, R., Batista, J., Sminchisescu, C.: Semantic Segmentation with Second-Order Pooling. ECCV (2012)
7. Cherian, A., Sra, S.: Riemannian Sparse Coding for Positive Definite Matrices. In: ECCV (2014)
8. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
9. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., Vedaldi, A.: Describing Textures in the Wild. In: CVPR (2014)
10. Cui, Y., Zhou, F., Wang, J., Liu, X., Lin, Y., Belongie, S.: Kernel Pooling for Convolutional Neural Networks. In: CVPR (2017)
11. Dalal, N., Triggs, B.: Histograms of Oriented Gradients for Human Detection. In: CVPR. pp. 886–893 (2005)
12. Freund, Y., Schapire, R.E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences (1997)
13. Gao, Y., Beijbom, O., Zhang, N., Darrell, T.: Compact Bilinear Pooling. In: CVPR. pp. 317–326 (2016)
14. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS (2010)
15. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
16. Guo, K., Ishwar, P., Konrad, J.: Action Recognition Using Sparse Representation on Covariance Manifolds of Optical Flow. In: IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) (2010)
17. Harandi, M.T., Salzmann, M., Hartley, R.: From manifold to manifold: Geometry-aware dimensionality reduction for SPD matrices. In: ECCV (2014)
18. Harandi, M., Salzmann, M.: Riemannian coding and dictionary learning: Kernels to the rescue. In: CVPR (2015)
19. Harandi, M.T., Sanderson, C., Hartley, R.I., Lovell, B.C.: Sparse Coding and Dictionary Learning for Symmetric Positive Definite Matrices - A Kernel Approach. In: ECCV (2012)

20. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: CVPR. pp. 770–778 (2016)
21. Huang, C.H., Boyer, E., Angonese, B.D.C., Navab, N., Ilic, S.: Toward User-Specific Tracking by Detection of Human Shapes in Multi-Cameras. In: CVPR (2015)
22. Huang, G., Liu, Z., Weinberger, K., van der Maaten, L.: Densely Connected Convolutional Networks. In: CVPR (2017)
23. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: ICML (2015)
24. Ionescu, C., Vantzos, O., Sminchisescu, C.: Matrix backpropagation for Deep Networks with Structured Layers (2015)
25. James, A.T.: The non-central wishart distribution. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences **229**(1178), 364–366 (1955), http://www.jstor.org/stable/99771
26. Johnson, R.A., Wichern, D.W., et al.: Applied multivariate statistical analysis, vol. 4. Prentice-Hall New Jersey (2014)
27. Kong, S., Fowlkes, C.: Low-Rank Bilinear Pooling for Fine-Grained Classification. In: CVPR (2017)
28. Koniusz, P., Tas, Y., Porikli, F.: Domain Adaptation by Mixture of Alignments of Second- or Higher-Order Scatter Tensors. In: CVPR (2017)
29. Koniusz, P., Zhang, H., Porikli, F.: A deeper look at power normalizations. In: CVPR. pp. 5774–5783 (2018)
30. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet Classification with Deep Convolutional Neural Networks. In: NIPS. pp. 1106–1114 (2012)
31. Lazebnik, S., Schmid, C., Ponce, J.: Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In: CVPR (2006)
32. Li, P., Xie, J., Wang, Q., Zuo, W.: Is Second-Order Information Helpful for Large-Scale Visual Recognition? In: ICCV (2017)
33. Li, P., Wang, Q., Zuo, W., Zhang, L.: Log-Euclidean Kernels for Sparse Representation and Dictionary Learning. In: ICCV (2013)
34. Lin, T.Y., Maji, S.: Improved Bilinear Pooling with CNNs. In: BMVC (2017)
35. Lin, T.Y., Maji, S., Koniusz, P.: Second-order democratic aggregation. In: ECCV (2018)
36. Lin, T., RoyChowdhury, A., Maji, S.: Bilinear Cnn Models for Fine-Grained Visual Recognition. In: ICCV. pp. 1449–1457 (2015)
37. Pennec, X., Fillard, P., Ayache, N.: A Riemannian Framework for Tensor Computing. IJCV (2006)
38. Perronnin, F., Sánchez, J., Mensink, T.: Improving the Fisher Kernel for Large-Scale Image Classification. ECCV (2010)
39. Quang, M.H., San-Biagio, M., Murino, V.: Log-Hilbert-Schmidt metric between positive definite operators on Hilbert spaces. NIPS (2014)
40. Quattoni, A., Torralba, A.: Recognizing Indoor Scenes. In: CVPR. pp. 413–420 (2009)
41. Sermanet, P., Chintala, S., LeCun, Y.: Convolutional Neural Networks Applied to House Numbers Digit Classification. In: ICPR (2012)
42. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: ICLR (2015)
43. Sra, S.: A new metric on the manifold of kernel matrices with application to matrix geometric means. In: NIPS (2012)
44. Sra, S., Cherian, A.: Generalized Dictionary Learning for Symmetric Positive Definite Matrices with Application to Nearest Neighbor Retrieval. In: Machine Learning and Knowledge Discovery in Databases. Springer, Berlin, Heidelberg (2011)

45. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going Deeper with Convolutions. In: CVPR. pp. 1–9 (June 2015)
46. Tuzel, O., Porikli, F., Meer, P.: Human Detection via Classification on Riemannian Manifolds. In: CVPR. pp. 1–8 (2007)
47. Vapnik, V.: Statistical Learning Theory. Wiley-Interscience (1998)
48. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Tech. rep. (2011)
49. Wang, Q., Li, P., Zuo, W., Zhang, L.: RAID-G - Robust Estimation of Approximate Infinite Dimensional Gaussian with Application to Material Recognition. In: CVPR (2016)
50. Wilson, E.B., Hilferty, M.M.: The distribution of chi-square. Proceedings of the National Academy of Sciences **17**(12), 684–688 (1931)