ReDro: Efficiently Learning Large-sized SPD Visual Representation

Saimunur Rahman[1,2][0000−0002−5250−5612], Lei Wang[1][0000−0002−0961−0441], Changming Sun[2][0000−0001−5943−1989], and Luping Zhou[3][0000−0003−1065−6604]

[1] VILA, School of Computing and Information Technology, University of Wollongong, NSW 2522, Australia
[2] CSIRO Data61, PO Box 76, Epping, NSW 1710, Australia
[3] School of Electrical and Information Engineering, University of Sydney, NSW 2006, Australia
sr801@uowmail.edu.au; leiw@uow.edu.au; changming.sun@csiro.au; luping.zhou@sydney.edu.au

Abstract. Symmetric positive definite (SPD) matrix has recently been used as an effective visual representation. When learning this representation in deep networks, eigen-decomposition of covariance matrix is usually needed for a key step called matrix normalisation. This could result in significant computational cost, especially when facing the increasing number of channels in recent advanced deep networks.
This work proposes a novel scheme called Relation Dropout (ReDro). It is inspired by the fact that eigen-decomposition of a *block diagonal* matrix can be efficiently obtained by decomposing each of its diagonal square matrices, which are of smaller sizes. Instead of using a full covariance matrix as in the literature, we generate a block diagonal one by randomly grouping the channels and only considering the covariance within the same group. We insert ReDro as an additional layer before the step of matrix normalisation and make its random grouping transparent to all subsequent layers. Additionally, we can view the ReDro scheme as a dropout-like regularisation, which drops the channel relationship across groups. As experimentally demonstrated, for the SPD methods typically involving the matrix normalisation step, ReDro can effectively help them reduce computational cost in learning large-sized SPD visual representation and also help to improve image recognition performance.

Keywords: Block diagonal matrix, Covariance, Eigen-decomposition, SPD representation, Fine-grained image recognition.

1 Introduction

Learning good visual representation remains a central issue in computer vision. Representing images with local descriptors and pooling them to a global representation has been effective. Among the pooling methods, covariance matrix based pooling has gained substantial interest by exploiting the second-order information of features. Since covariance matrix is symmetric positive definite

(SPD), the resulting representation is often called SPD visual representation. It has shown promising performance in various tasks, including fine-grained image classification [21], image segmentation [11], generic image classification [12, 19], image set classification [33], activity and action recognition [39, 14] and few-shot learning [36, 37], to name a few. With the advent of deep learning, several pieces of pioneering work have integrated SPD representation into convolutional neural networks (CNNs) and investigated a range of important issues such as matrix function back-propagation [11], compact matrix estimation [7], matrix normalisation [20, 16, 15] and kernel-based extension [6]. These progresses bring forth effective SPD visual representations and improve image recognition performance.

Despite the successes, the end-to-end learning of SPD representation in CNNs poses a computational challenge. This is because i) the size of covariance matrix increases quadratically with the channel number in a convolutional feature map and ii) eigen-decomposition is often needed to normalise the covariance matrix in back-propagation for each training sample. This results in significant computation, especially considering that many channels are deployed in recent advanced deep networks. Although a dimension reduction layer could always be used to reduce the channel number beforehand, we are curious about if this computational challenge can be mitigated from another orthogonal perspective.

This work is inspired by the following fact: the eigen-decomposition of a *block diagonal* matrix can be obtained by simply assembling the eigenvectors and eigenvalues of its diagonal square matrices [23]. Each diagonal matrix is smaller in size and the eigen-decomposition needs less computation. Motivated by this, we propose to replace a full covariance matrix with a block diagonal one. To achieve this, all channels must be partitioned into mutually exclusive groups and the covariance of the channels in different groups shall be omitted (i.e., set as zero). A question that may arise is how to optimally partition the channels to minimise the loss of covariance information or maximise the final recognition performance. Although this optimum could be pursued by redesigning network architecture or loss function (e.g., considering the idea in [38]), it will alter the original SPD methods that use matrix normalisation and potentially increase the complexity of network training.

To realise a block diagonal covariance matrix with the minimal alteration of the original methods, negligible extra computation and no extra parameters to learn, we resort to a *random* partitioning of channels. This can be trivially implemented, with some housekeeping operation to make the randomness transparent to all the network layers after the matrix normalisation step. To carry out the end-to-end training via back-propagation, we derive the relevant matrix gradients in the presence of this randomisation. The saving on computation and the intensity of changing the random partitioning pattern are also discussed.

In addition, we conceptually link the proposed random omission of relationship (i.e., covariance) of channels to the dropout scheme commonly used in deep learning [27]. We call our scheme "Relation Dropout (ReDro)" for short. It is found that besides serving the goal of mitigating the computational challenge, the proposed scheme could bring forth an additional advantage of improving the

network training and image recognition performance, which is consistent with the spirit of the extensively used dropout techniques.

The main contributions of this paper are summarised as follows.

- To mitigate the computational issue in learning large-sized SPD representation for the methods using matrix normalisation, this paper proposes a scheme called ReDro to take advantage of the eigen-decomposition efficiency of a block diagonal matrix. To the best of our survey, such a random partition based scheme is new for the deep learning of SPD visual representation.
- Via the randomisation mechanism, the ReDro scheme maintains the minimal change to the original network design and negligible computational overhead. This work derives the forward and backward propagations in the presence of the proposed scheme and discusses its properties.
- Conceptually viewing the ReDro scheme as a kind of dropout, we investigate its regularisation effect and find that it could additionally help improving network training efficiency and image recognition performance.

Extensive experiments are conducted on one scene dataset and three fine-grained image datasets to verify the effectiveness of the proposed scheme.

2 Related Work

Learning SPD visual representation. SPD visual representation can be traced back to covariance region descriptor in object detection, classification and tracking [29, 30, 25]. The advent of deep learning provides powerful image features and further exhibits the potential of SPD visual representation. After early attempts which compute covariance matrix on pre-extracted deep features [3], research along this line quickly enters the end-to-end learning paradigm and thrives. Covariance matrix is embedded into CNNs as a special layer and jointly learned with network weights to obtain the best possible SPD visual representation.

DeepO$_2$P [11] and Bilinear CNN (BCNN) [21] are two pieces of pioneering work that learn SPD visual representation in an end-to-end manner. The framework of DeepO$_2$P is largely followed and continuously improved by subsequent works. It generally consists of three parts. The first part feeds an image into a CNN backbone and processes it till the last layer of 3D convolutional feature maps, with width w, height h and channel number d. Viewing this map as a set of $w \times h$ local descriptors of d dimensions, the second part computes a (normalised, which will be detailed shortly) $d \times d$ covariance matrix to characterise the channel correlation. The last part is routine, usually consisting of fully connected layers and the softmax layer for prediction.

Matrix normalisation. The step of matrix normalisation in the second part above plays a crucial role. It is widely seen in the recent work to learn SPD visual representation due to three motivations: i) Covariance matrix resides on a Riemannian manifold, whose geometric structure needs to be considered; ii) Normalisation

is required to battle the "burstiness" phenomenon—a visual pattern usually occurs more times once it appears in an image; iii) Normalisation helps to achieve robust covariance estimation against small sample.

After element-wise normalisation, the recent work turns to matrix-logarithm or matrix-power normalisation because they usually produce better SPD representation.[4] Nevertheless, both of them involve the eigen-decomposition of covariance matrix, whose computational complexity can be up to $\mathcal{O}(d^3)$. This operation has to be applied for each training sample in every forward and backward propagations. The step of matrix normalisation becomes a computational bottleneck in the end-to-end learning of SPD visual representation.

The recent literature has made an effort to reduce the computation of matrix normalisation. They consider a special case of matrix power normalisation, that is, matrix square-root normalisation. In the work of [20], this is approximately calculated by applying Newton-Schulz iteration for root finding. It makes forward propagation computationally more efficient since only matrix multiplications are involved. The backward propagation still needs to solve a Lyapunov equation, which has the complexity at the same level of eigen-decomposition. After that, the work in [18] solves matrix square-rooting more efficiently. It proposes a sandwiched Newton-Schulz iteration and implements it via a set of layers with loop-embedded directed graph structure to obtain an approximate matrix square-root. It can be used for both forward and backward propagations.

Although the work in [20, 18] achieves computational advantage and promising performance by using matrix square-root, their methods do not generalise to matrix normalisation with other power value p or other normalisation function f. In addition, the applicability of iterative Newton-Schulz equation to large-sized covariance matrix is unclear since only smaller-sized covariance matrices (i.e., of size 256×256) are used in that two works. These motivate us to mitigate the computational issue of matrix normalisation from other perspectives.

Finally, there are also research works to address the complexity of learning large-sized SPD representations by focusing on the parts other than matrix normalisation. Compact, low-rank and group bilinear pooling methods [7, 13, 38] address the high dimensionality of the feature representation after vectorising covariance matrix, and group convolution is used in [32] for the similar purpose. Linear transformation is designed in [1, 10, 24, 35, 32] to project large covariance matrices to more discriminative, smaller ones. To efficiently capture higher-order feature interactions, kernel pooling is developed [4]. Furthermore, the work in [2] develops a new learning framework to directly process manifold-valued data including covariance matrix. For our work, instead of competing with these works, it complements them and could be jointly used to mitigate the computational issue of eigen-decomposition of large SPD matrices when needed. In this work,

[4] Let $\mathbf{C}$ be a SPD matrix and its eigen-decomposition is $\mathbf{C} = \mathbf{U}\mathbf{D}\mathbf{U}^\top$. The columns of $\mathbf{U}$ are eigenvectors while the diagonal of the diagonal matrix $\mathbf{D}$ consists of eigenvalues. Matrix normalisation with a function f is defined as $f(\mathbf{C}) = \mathbf{U}f(\mathbf{D})\mathbf{U}^\top$, where $f(\mathbf{D})$ means f is applied to each diagonal entry of $\mathbf{D}$. Matrix-logarithm and matrix-power based normalisations correspond to $f(x) = \log(x)$ and $f(x) = x^p$.

we focus on the frameworks in [20, 19, 18, 6] which typically employ matrix normalisation as an important step in learning SPD representation.

Dropout schemes. Dropout [27] is a common regularisation technique that randomly drops neuron units from fully connected layers to improve generalisation. Several new schemes have extended this idea to convolutional layers. SpatialDropout [28] randomly drops feature channels. DropBlock [8] drops a block of pixels from convolutional feature maps. Weighted channel dropout [9] randomly drops feature channels with lower activations. Conceptually, the proposed "Relation Dropout (ReDro)" can be viewed as another scheme. Unlike the above ones, it randomly drops the covariance relationship of the channels across groups. It yields block-diagonal variants of a covariance matrix, and could produce dropout-like regularisation effect in training, as will be experimentally demonstrated.

3 The Proposed Relation Dropout (ReDro)

An overview of ReDro is in Figure 1. From the left end, an image is fed into a CNN backbone, and the last convolutional feature map of d channels is routinely obtained. ReDro firstly conducts a random permutation of these channels and then evenly partitions them into k groups by following the channel number. Restricted to group i ($i = 1, 2, \cdots, k$), a smaller-sized covariance matrix $\mathbf{C}_i$ is computed and its eigen-decomposition is conducted as $\mathbf{C}_i = \mathbf{U}_i\mathbf{D}_i\mathbf{U}_i^\top$. The eigenvectors in $\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k$ are then arranged to form a larger, block-diagonal matrix $\mathbf{U}_b$. The similar procedure applies to the eigenvalues in $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k$ to form $\mathbf{D}_b$. Note that $\mathbf{U}_b$ and $\mathbf{D}_b$ are just the eigen-decomposition of the $d \times d$ block-diagonal covariance matrix $\mathbf{C}_b = \mathrm{diag}(\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_k)$. At the last step of ReDro, $\mathbf{U}_b$ and $\mathbf{D}_b$ is *permuted back* to the original order of the channels in the last convolutional feature map. This is important because it makes the random permutation *transparent* to subsequent network layers. This completes the proposed ReDro scheme.

In doing so, the eigen-decomposition of a covariance matrix, with part of the entries dropped, can be more efficiently obtained by taking advantage of the block-diagonal structure of $\mathbf{C}_b$. Then matrix normalisation can be readily conducted with any valid normalisation function.

3.1 Forward propagation in the presence of ReDro

Let $\mathbf{X}_{d\times(wh)}$ be a data matrix consisting of the d-dimensional local descriptors in the last convolutional feature map. Recall that d is the channel number while w and h are the width and height of the feature map. A random partitioning of the d channels can be represented by k index sets, $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_k$, which contain the IDs of the channels in each group, respectively.

Let a roster of all the IDs in these k index sets be $\{r_1, r_2, \cdots, r_d\}$. It is a permutation of the original channel IDs $\{1, 2, \cdots, d\}$, and therefore induces

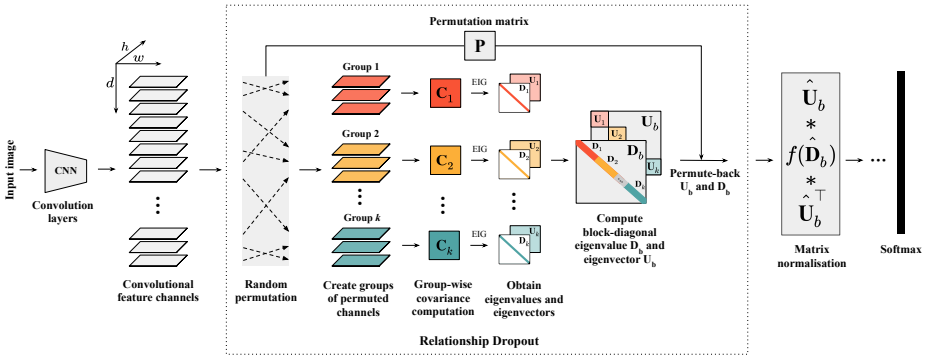

Fig. 1. Proposed relation dropout (ReDro) scheme for mitigating the computational issue of matrix normalisation in learning large-sized SPD visual representation.

a permutation matrix $\mathbf{P}_{d\times d}$.[5] The ith row of $\mathbf{P}$ is $\mathbf{e}_{r_i}^\top = (0,\cdots,0,1,0,\cdots,0)$, which is a standard basis vector with "1" at its r_ith entry and zeros elsewhere. The effect of $\mathbf{P}$ can be intuitively interpreted. By left-multiplying $\mathbf{P}$ to $\mathbf{X}$, the rows of $\mathbf{X}$ will be permuted. A more intuitive interpretation, which will be used later, is that it permutes the d axes of an original coordinate frame $\mathcal{F}$ to form another new frame $\mathcal{F}'$. For a quantity (e.g., eigenvector) represented in $\mathcal{F}'$, we can inverse the permutation by left-multiplying $\mathbf{P}^{-1}$ to it. It is known in matrix analysis that for any permutation matrix $\mathbf{P}$, it satisfies $\mathbf{P}\mathbf{P}^\top = \mathbf{I}$. Therefore, $\mathbf{P}^{-1}$ can be trivially obtained as $\mathbf{P}^\top$. This result will be used shortly.

Now, for the channels within each group $\mathcal{G}_i$ $(i = 1,\cdots,k)$, we compute a covariance matrix $\mathbf{C}_i$. Collectively, they form a $d \times d$ block-diagonal matrix

$$\mathbf{C}_b = \text{diag}(\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_k). \tag{1}$$

Let the eigen-decomposition of $\mathbf{C}_i$ be $\mathbf{C}_i = \mathbf{U}_i\mathbf{D}_i\mathbf{U}_i^\top$. It is well-known by matrix analysis that the eigen-decomposition of $\mathbf{C}_b$ can be expressed as [23]

$$\mathbf{C}_b = \mathbf{U}_b\mathbf{D}_b\mathbf{U}_b^\top, \tag{2}$$

$$\mathbf{U}_b = \text{diag}(\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k) \;\text{ and }\; \mathbf{D}_b = \text{diag}(\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k). \tag{3}$$

Note that the eigenvectors in $\mathbf{U}_b$ are obtained in the new coordinate frame $\mathcal{F}'$. To retrieve their counterparts, $\hat{\mathbf{U}}_b$, in the original frame $\mathcal{F}$ (i.e., corresponding to the original order of the d channels), we apply the inverse permutation as

$$\hat{\mathbf{U}}_b = \mathbf{P}^{-1}\mathbf{U}_b = \mathbf{P}^\top\mathbf{U}_b. \tag{4}$$

The eigenvalue matrix $\mathbf{D}_b$ does not need to be permuted back because an eigenvalue represents the data variance along the corresponding eigenvector. It is not affected by the permutation of the coordinate axes.

[5] In matrix analysis, a permutation matrix $\mathbf{P}$ is a square binary matrix. It has one and only one "1" entry in each row and each column, with all the remainder being "0". It is easy to verify that $\mathbf{P}\mathbf{P}^\top = \mathbf{P}^\top\mathbf{P} = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix.

In this way, we obtain the eigen-decomposition of $\mathbf{C}_b$ *corresponding to the original order of the channels (i.e,* $1, 2, \cdots, d$*)* as

$$\hat{\mathbf{U}}_b = \mathbf{P}^\top \cdot \mathrm{diag}(\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k); \quad \hat{\mathbf{D}}_b = \mathrm{diag}(\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k). \tag{5}$$

With this result, matrix normalisation with any valid function f can now be applied. Algorithm 1 summarises the steps of the proposed ReDro scheme.

Algorithm 1: Relation Dropout (ReDro)

Input : Convolutional feature map $\mathbf{X}_{d\times(wh)}$; The number of groups k.

1. Randomly partition the d channels to groups $\mathcal{G}_1$, $\mathcal{G}_2$, $\cdots$, $\mathcal{G}_k$;
2. Create the permutation matrix $\mathbf{P}$ accordingly;
3. **foreach** *group* $\mathcal{G}_i$ **do**
   1. Compute the covariance matrix $\mathbf{C}_i$;
   2. Calculate its eigen-decomposition $\mathbf{C}_i = \mathbf{U}_i\mathbf{D}_i\mathbf{U}_i^\top$;

   end
4. Form the eigenvectors and eigenvalues of the block-diagonal matrix $\mathbf{C}_b$: $\mathbf{U}_b = \mathrm{diag}(\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k)$, $\mathbf{D}_b = \mathrm{diag}(\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k)$;
5. By permuting back, the eigen-decomposition of $\mathbf{C}_b$ *corresponding to the original order of the* d *channels* are $\hat{\mathbf{U}}_b = \mathbf{P}^\top\mathbf{U}_b$ and $\hat{\mathbf{D}}_b = \mathbf{D}_b$.

3.2 Backward propagation in the presence of ReDro

To derive the gradients for back-propagation, the composition of functions from the feature map $\mathbf{X}$ to the objective function $J(\mathbf{X})$ is illustrated as follows.

$$\mathbf{X} \to \underbrace{\mathbf{P}\mathbf{X}}_{\mathbf{Y}} \to \underbrace{(\mathbf{Y}\mathbf{Y}^\top)\circ\mathbf{S}}_{\mathbf{C}_b} \to \underbrace{(\mathbf{P}^\top\mathbf{C}_b\mathbf{P})}_{\mathbf{A}(\text{Auxiliary})} \to \underbrace{f(\mathbf{A})}_{\mathbf{Z}} \to \cdots \text{layers} \cdots \to \underbrace{J(\mathbf{X})}_{\text{Objective}} \tag{6}$$

$\mathbf{P}$ is the permutation matrix. $\mathbf{Y}$ is the feature map with its channels permutated. $\mathbf{S} = \mathrm{diag}(\mathbf{1}_1, \mathbf{1}_2, \cdots, \mathbf{1}_k)$ is a block-diagonal binary matrix, where $\mathbf{1}_i$ is a square matrix of all "1"s and its size is the same as that of channel group $\mathcal{G}_i$. Noting that "$\circ$" denotes element-wise multiplication, $\mathbf{S}$ represents the selection of $\mathbf{C}_b$ out of the full covariance matrix $\mathbf{Y}\mathbf{Y}^\top$. The letter under each term is used to assist derivation. $\mathbf{A}$ is an auxiliary variable and not computed in practice. $f(\mathbf{A})$ is the step of matrix normalisation. Its result $\mathbf{Z}$ is used by the subsequent fully connected and softmax layers. Our goal is to work out $\frac{\partial J}{\partial \mathbf{X}}$. Once obtained, all gradients before the convolutional feature map $\mathbf{X}$ can be routinely obtained.

In the literature, the seminal work in [11] and other work such as [6] demonstrate the rules of differentiation for matrix-valued functions. By these rules, we derive the following results (details are provided in the supplement).

$$\begin{aligned}
&\frac{\partial J}{\partial \mathbf{A}} = \hat{\mathbf{U}}_b\left(\mathbf{G}\circ\left(\hat{\mathbf{U}}_b^\top\frac{\partial J}{\partial \mathbf{Z}}\hat{\mathbf{U}}_b\right)\right)\hat{\mathbf{U}}_b^\top; \quad \frac{\partial J}{\partial \mathbf{C}_b} = \mathbf{P}\frac{\partial J}{\partial \mathbf{A}}\mathbf{P}^\top; \\
&\frac{\partial J}{\partial \mathbf{Y}} = \left(\mathbf{S}\circ\left(\frac{\partial J}{\partial \mathbf{C}_b} + \left(\frac{\partial J}{\partial \mathbf{C}_b}\right)^\top\right)\right)\mathbf{Y}; \quad \frac{\partial J}{\partial \mathbf{X}} = \mathbf{P}^\top\frac{\partial J}{\partial \mathbf{Y}},
\end{aligned} \tag{7}$$

where $\mathbf{G}$ is a $d \times d$ matrix defined based on $\hat{\mathbf{D}}_b$.[6] As seen, given $\frac{\partial J}{\partial \mathbf{Z}}$, we can work out $\frac{\partial J}{\partial \mathbf{A}}$, $\frac{\partial J}{\partial \mathbf{C}_b}$, $\frac{\partial J}{\partial \mathbf{Y}}$ and then $\frac{\partial J}{\partial \mathbf{X}}$. In the whole course, only $\hat{\mathbf{U}}_b$, $\hat{\mathbf{D}}_b$, $\mathbf{P}$ and $\mathbf{S}$ are needed. The first two have been efficiently obtained via the proposed scheme ReDro in Sec. 3.1, while the latter two are known once the random grouping is done. Now, we have all the essential results for back-propagation. An end-to-end learning with ReDro can be readily implemented.

3.3 Discussion

Computational savings. As aforementioned, a computational bottleneck in SPD representation learning is the eigen-decomposition of a $d \times d$ full covariance matrix. Generally, its complexity is at the order of $\mathcal{O}(d^3)$.[7] Without loss of generality, assuming that d can exactly be divided by the number of groups k, the size of each group will be d/k. The complexity incurred by ReDro, which conducts eigen-decomposition of the k smaller-sized covariance matrices, will be $\mathcal{O}(d^3/k^2)$. Therefore, in the theoretical sense, the proposed ReDro scheme can save the computation up to k^2 times.

The implementation of ReDro only needs a random permutation of the IDs of the d channels. For the gradient computation in Eq.(7), it appears that compared with the case of using a full covariance matrix, ReDro incurs extra computation involving the multiplication of $\mathbf{P}$ or $\mathbf{S}$ in $\frac{\partial J}{\partial \mathbf{C}_b}$, $\frac{\partial J}{\partial \mathbf{Y}}$ and $\frac{\partial J}{\partial \mathbf{X}}$ (Note that $\frac{\partial J}{\partial \mathbf{A}}$ needs to be computed for any eigen-decomposition based matrix normalisation, even if ReDro is not used). Nevertheless, both $\mathbf{P}$ and $\mathbf{S}$ are binary matrices simply induced by the random permutation. Their multiplication with other variables can be trivially implemented, incurring little computational overhead.

Two key parameters. One key parameter is the number of groups, k. Since the improvement on computational efficiency increases quadratically with k, a larger k would be preferred. Meanwhile, it is easy to see that the percentage of the entries dropped by ReDro is $(1 - \frac{1}{k}) \times 100\%$. A larger k will incur more significant loss of information. As a result, a value of k balancing these two aspects shall be used, which will be demonstrated in the experimental study.

When k is given, the other key parameter is the "intensity" of conducting the random permutation. Doing it for every training sample leads to the most intensive change of dropout pattern. As will be shown, this could make the objective function fluctuate violently, affecting the convergence. To show the impact of this intensity, we will experiment the random permutation at three levels, namely, epoch-level (EL), batch-level (BL) and sample-level (SL) and hold it for various intervals. For example, batch-level with interval of 2 (i.e., "BL-2") uses the same random permutation for two consecutive batches before refreshed. Similarly, SL-1 conducts the random permutation for every training sample.

[6] For the matrix $\mathbf{G}$, its (i,j)th entry g_{ij} is defined as $\frac{f(\lambda_i)-f(\lambda_j)}{\lambda_i-\lambda_j}$ if $\lambda_i \neq \lambda_j$ and $f'(\lambda_i)$ otherwise, where λ_i is the ith diagonal element of $\hat{\mathbf{D}}_b$.

[7] For a symmetric matrix, the complexity of eigen-decomposition could be improved up to the order of $\mathcal{O}(d^{2.38})$ by more sophisticated algorithms though [5].

In addition, ReDro could bring less biased eigenvalue estimate. As known, when eigen-decomposition is applied to a large, full covariance matrix, eigenvalue estimation will be considerably biased (i.e., larger/smaller eigenvalues are estimated to be over-large/over-small) when samples are not sufficient. When ReDro is used, eigenvalues are estimated from each block sub-matrix on the diagonal. Because they are smaller in size, eigenvalue estimate could become less biased. This property can be regarded as a by-product of the ReDro scheme.

4 Experimental Result

We conduct extensive experiments on scene classification and fine-grained image classification to investigate the proposed ReDro scheme. For scene classification, *MIT Indoor* dataset [26] is used. For fine-grained image classification, the commonly used *Birds* [34], *Airplanes* [22] and *Cars* [17] datasets are tested. For all datasets, the original training and testing protocols are followed, and we do not utilise any bounding box or part annotations. Following the literature [20, 18], we resize all images to 448×448 during training and testing. More details on datasets and implementation of ReDro are provided in the supplement material.

This experiment consists of three main parts, with an additional ablation study. Sec. 4.1 shows the computational advantage brought by ReDro. Sec. 4.2 investigates the efficiency of ReDro versus the intensity level at which it is applied. Sec. 4.3 validates the performance of ReDro via multiple typical SPD representation learning methods that explicitly use matrix normalisation.

4.1 On the computational advantage of ReDro

This part compares the computational cost between the case without ReDro and the case using ReDro. The former means a full covariance matrix is used.

Specifically, we compare with four typical SPD representation learning methods, namely, MPN-COV [19], DeepCOV [6], Improved BCNN (IBCNN) [20] and

Table 1. Comparison of computational time (in second) for covariance estimation and matrix normalisation by using or not using the proposed ReDro scheme. The reported time is the sum of forward and backward propagation (individual propagation time is given in the supplement). The four methods to the left represent the case not using ReDro. The case using ReDro is implemented upon DeepCOV [6] with various k. The boldface shows that ReDro saves computational time

Matrix Dimension	No ReDro used				DeepCOV [6] using ReDro			
	MPN-COV [19]	Deep-COV [6]	IBCNN [20]	iSQRT-COV [18]	with $k=2$	with $k=4$	with $k=8$	with $k=16$
128×128	0.004	0.004	0.006	0.001	0.007	0.009	0.011	0.015
256×256	0.013	0.013	0.014	0.006	0.011	0.011	0.013	0.020
512×512	0.031	0.031	0.032	0.030	0.030	**0.022**	**0.023**	0.030
1024×1024	0.097	0.097	0.121	0.097	**0.090**	**0.076**	**0.056**	**0.062**

iSQRT-COV [18]. The first two methods conduct matrix normalisation via eigen-decomposition, while the last two realise normalisation more efficiently by matrix square-rooting. These four methods represent the case without using ReDro. To compare with them, we implement ReDro within the DeepCOV method with ResNet-50 network as backbone. When doing this, we only modify its COV layers with the proposed ReDro scheme, leaving all the other settings unchanged.

Since the layers of covariance computation and matrix normalisation in these methods are often tightly coupled, it is difficult to exclude the normalisation step to exactly compare with ReDro for computational cost. To be fair, the total time taken by both the steps of ReDro and matrix normalisation is used for comparison. To test the computational cost over covariance matrix of various sizes, we follow the literature to apply an additional 1×1 convolutional layer to reduce the number of channels when necessary. The comparison is conducted on a computer with a Tesla P100 GPU, 12-core CPU and 12 GB RAM. ReDro is implemented with MatConvNet library [31] on Matlab 2019a.

Table 1 shows the timing result. Firstly, along the size of covariance matrix, we can observe that i) when the size is relatively smaller (i.e., 256×256), ReDro is unnecessary. This is because the eigen-decomposition does not incur significant computation. Rigidly using ReDro in this case complicates the procedure, adding computational overhead; ii) however, when the matrix size increases to 512×512, the advantage of ReDro emerges, and becomes more pronounced for 1024×1024. In these cases, ReDro is computationally more efficient by $20\% \sim 50\%$ than the no-ReDro counterparts (see the results in bold). In addition, in terms of the total training time, ReDro can significantly shorten the time from 52.4 hours down to 24.3 hours on Birds dataset, with 1.3 percentage point improvement on classification accuracy, as will be detailed in Figure 2.

Secondly, we test different group number k in ReDro. As seen, ReDro shows higher computational efficiency when $k = 4$ or 8. For $k = 2$, we can expect from the previous complexity analysis that its efficiency shall be lower. For $k = 16$, the overhead for processing more matrices, although in smaller sizes, becomes non-trivial. Since $k = 4$ gives rise to a high computational efficiency (and overall good classification, as will be shown later), the following two experiments will focus on this setting, with more discussion on k left to the ablation study.

4.2 On the efficiency of ReDro versus its intensity level

Based on the findings in the last experiment, we focus on testing with 512×512- and 1024×1024-sized matrices in this experiment.

This experiment investigates the impact of intensity level, at which ReDro is applied, to classification improvement. Again, we implement ReDro with DeepCOV [6]. Specifically, we train DeepCOV by applying ReDro with 15 intensity levels, i.e., EL-$\{100, 50, 20, 10, 5, 2, 1\}$, BL-$\{20, 10, 5, 2, 1\}$ and SL-$\{6, 3, 1\}$, whose meaning is explained in Sec. 3.3. The obtained classification accuracy is compared with those of the original DeepCOV reported in [6].

The left part of Table 2 shows the results when the covariance matrix size is 512×512. Firstly, along the intensity level, we can observe that, i) the efficiency

Table 2. Results using ReDro with $k = 4$ at different intensity levels on DeepCOV [6] (left) and DeepCOV-ResNet (right). The results higher than the baseline are shown in bold. The highest results for each dataset are marked by asterisks

		DeepCOV [6] (512×512)				DeepCOV-ResNet (1024×1024)			
		MIT	Airplane	Cars	Birds	MIT	Airplane	Cars	Birds
No ReDro		79.2	88.7	91.7	85.4	83.4	83.9	85.0	86.0
ReDro with $k = 4$	EL-1	**80.5***	**89.1**	**92.2***	**86.5**	**84.0***	**85.4**	**88.9**	**86.2***
	EL-2	**80.5***	**88.9**	**92.1**	**86.5**	–	–	–	–
	EL-5	**80.1**	**89.0**	91.7	**86.0**	**83.6**	**85.8***	**89.8***	86.0
	EL-10	**80.3**	88.6	88.9	**85.5**	–	–	–	–
	EL-20	**79.9**	88.6	91.3	85.0	81.5	**85.3**	**87.9**	85.9
	EL-50	**79.4**	**89.0**	91.0	**85.8**	–	–	–	–
	EL-100	**80.2**	**89.2***	90.8	**86.1**	–	–	–	–
	BL-1	76.8	**88.8**	87.5	**86.6**	**84.0***	**85.2**	**87.9**	**86.2***
	BL-2	76.9	**89.1**	88.0	**86.4**	–	–	–	–
	BL-5	76.6	**89.1**	87.8	**86.6**	**83.8**	**85.2**	**88.0**	**86.1**
	BL-10	76.8	**89.0**	88.0	**86.5**	–	–	–	–
	BL-20	78.6	88.7	87.5	**86.5**	**84.0***	**85.5**	**87.9**	**86.1**
	SL-1	77.7	**88.9**	90.0	**86.6**	**83.8**	**85.1**	84.8	86.0
	SL-3	78.1	**89.0**	89.8	**86.5**	–	–	–	–
	SL-6	79.1	**89.2***	91.1	**86.7***	**83.8**	**85.2**	**86.2**	**86.1**

of ReDro indeed varies with the intensity level; ii) multiple intensity levels lead to improved classification over the baseline "No ReDro." In particular, EL-1 and EL-2 work overall better than others across all datasets; iii) also, EL-1 works well with MIT and Cars, and SL-6 works well with Airplane and Birds though. This suggests that for some datasets, less intensive change of random permutation, i.e., at epoch level (EL), is preferred. We observe that in this case, applying more intensive change, i.e., at sample level (SL), could cause objective function to fluctuate violently and affect convergence.

Secondly, along the datasets, we can observe that DeepCOV trained with ReDro improves classification over its baseline on all datasets, with the magnitude of $0.5 \sim 1.3\%$. For example, on Birds with the intensity level of SL-6, ReDro improves the accuracy from the baseline 85.4% to 86.7%.

Besides improving classification, ReDro at various intensity levels also shortens the total network training time. At each epoch of training DeepCOV, ReDro saves about 40% of the GPU time. We find that it also helps network to converge faster. Figure 2 shows that DeepCOV with ReDro achieves lower classification error with a smaller number of epochs than the baseline "No ReDro" on Birds. As seen, i) when ReDro is used, training the network, including the one achieving the highest accuracy, generally takes about half of the time of the baseline (indicated by the vertical dotted line), regardless of the intensity level; ii) with ReDro, to train a network to achieve comparable performance with the baseline, it only takes about 36% (i.e, down from 52.4 hours to 19.0 hours) of the baseline training time. The similar observation is seen from other datasets besides Birds.

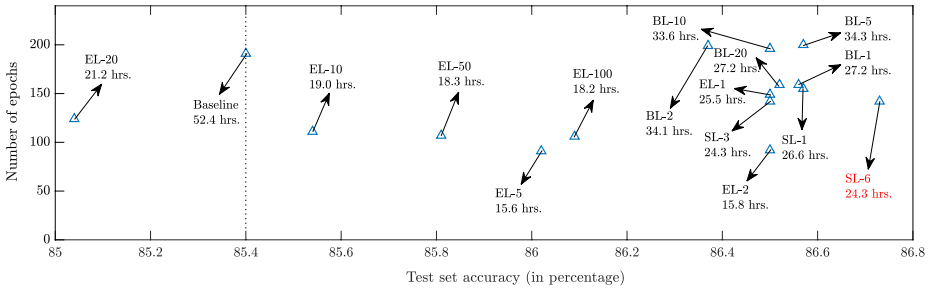

Fig. 2. Comparison of the classification error, network training time, and the total number of epochs obtained by using ReDro at various intensity levels. Each small triangle in this figure represents a ReDro case. The arrow points to its intensity level and the network training time taken to achieve its highest test accuracy, as reflected by the x-axis. The result in red indicates the best performer. Birds dataset is used.

The right part of Table 2 shows the result when the covariance matrix size is 1024×1024. Note that no existing networks have ever tried such a large matrix. We use DeepCOV network with ResNet-50 as the backbone, and apply 1×1 convolution to reduce the number of feature channels from 2048 to 1024. We call this network "DeepCOV-ResNet." Due to the longer training period caused by the larger covariance matrix, we sample eight out of the 15 intensity levels in the table and test them. From the results, we can observe that, i) except for a few intensity levels, DeepCOV-ResNet with ReDro outperforms the baseline consistently across all datasets; ii) the improvement varies between $0.7 \sim 4.8\%$ (This largest improvement is achieved by ReDro with EL-5 on Cars). The improvement could be attributed to two factors: a) The dropout-like regularisation effect in ReDro helps the network learn better features, and b) ReDro estimates smaller-sized covariance matrices, instead of a full $d \times d$ one, from the local descriptors in the convolutional feature map. This helps to mitigate the bias issue in the estimation of the eigenvalues of covariance matrix.

4.3 On the performance of ReDro with typical methods

The above experiments verify the efficiency of ReDro by integrating it into the DeepCOV method. Now, we further integrate ReDro into other typical SPD representation methods that use matrix normalisation as an important step, namely, Improved BCNN [20], MPN-COV [19] and iSQRT-COV [18]. Since iSQRT-COV uses matrix square-root normalisation without involving eigen-decomposition, we investigate the regularisation effect of ReDro for it. This also applies to BCNN [21] which does not have the matrix normalisation step. Since most of these methods are originally proposed for fine-grained image classification, we focus on the datasets of Airplane, Cars and Birds.

Table 3 shows the results. In total, six scenarios are implemented with these methods, using differently sized covariance matrix. For each scenario, ReDro at

the same intensity level is used. We compare the baseline "No ReDro" to the cases in which ReDro is integrated. As seen, i) except iSQRT-COV, networks trained with ReDro generally outperform their baseline counterparts. ReDro-based iSQRT-COV is comparable with the baseline; ii) overall, the improvement is consistent across all datasets. The improvement higher than 1% can be commonly seen, with the maximum one being 4.8%; iii) For the results from ReDro with $k = 2$ and 4, they are overall comparable or the latter is slightly better (e.g., higher in 10 out of the total 18 results). Taking the computational saving into account, $k = 4$ is a better option.

Additionally, we provide some explanation to the performance of iSQRT-COV. All the methods except iSQRT-COV use a backbone model that is pretrained *without* incorporating the SPD representation layers. These layers are only incorporated later and then fine-tuned with a fine-grained image dataset. Differently, the pretrained backbone model in iSQRT-COV has incorporated the SPD representation layers that are further fine-tuned with the fine-grained image dataset. This helps iSQRT-COV achieve more promising performance. Meanwhile, as noted previously, iSQRT-COV [18] utilises a special matrix normalisation without involving eigen-decomposition. Our ReDro could provide iSQRT-COV with extra potential in efficiently utilising the eigen-decomposition based matrix normalisation to access more normalisation functions.

Table 3. Results using ReDro in typical SPD visual representation methods. The results higher than the baseline (indicated with "No ReDro") are shown in bold. The IBCNN [20], BCNN [21] and iSQRT-COV [18] networks are trained (including the baseline) with the settings in their original papers. As for MPN-COV [19], it is trained with the same pretrained network as IBCNN, BCNN and DeepCOV for consistency. Note that to ensure a fair of comparison, we report the classification result obtained by softmax predictions and do not utilise the additional step of training a separate SVM classifier. The highest results on each method are marked by asterisks.

Dataset	Training mode	DeepCOV-ResNet (1024×1024)	DeepCOV (512×512)	IBCNN (512×512)	BCNN (512×512)	MPN-COV (256×256)	iSQRT-COV (256×256)
Airp.	No ReDro	83.9	88.7	87.0	85.3	86.1	91.1
	ReDro ($k = 2$)	**85.3**	**89.3***	**88.8***	**86.6***	**87.4**	90.6
	ReDro ($k = 4$)	**85.8***	**89.2**	**88.6**	**86.6***	**88.2***	91.1
Cars	No ReDro	85.0	91.7	90.6	89.1	89.8	92.6
	ReDro ($k = 2$)	**88.2**	90.8	**92.6***	**90.9***	**91.4**	92.3
	ReDro ($k = 4$)	**89.8***	**92.2***	**91.2**	**90.5**	**91.7***	92.6
Birds	No ReDro	86.0	85.4	85.4	84.1	82.9	88.5
	ReDro ($k = 2$)	85.9	**86.5**	**85.5***	**84.6***	**83.5***	88.0
	ReDro ($k = 4$)	**86.2***	**86.7***	84.6	83.9	**83.2**	**88.6***

Table 4. Impact of group number k in ReDro. The Birds dataset is used

Matrix Size	No ReDro	Value of k for ReDro				
		2	3	4	8	16
1024×1024	86.0	85.9	**86.2***	**86.2***	85.3	82.9
512×512	85.6	**85.8**	**85.8**	**85.9***	85.5	83.2
256×256	85.4	**85.6**	**85.7***	**85.5**	83.9	81.5

4.4 Ablation study on the group number k

To gain more understanding on the group number k in ReDro, the following experiment is conducted. With DeepCOV-ResNet, ReDro using different values of k is implemented. Table 4 shows the results. As seen, i) for all matrix sizes, $k = \{3, 4\}$ yields overall better performance than the baseline; ii) larger values of k, i.e., 8 and 16, produce inferior results because they drop a significant amount of information from the covariance matrix.

5 Conclusion

This paper proposes a novel scheme called Relation Dropout (ReDro) for reducing the computational cost in learning large-sized SPD visual representation by deep neural networks. Focusing on the step of matrix normalisation, it utilises the nice property of a block-diagonal matrix to facilitate the eigen-decomposition often required in this process. A detailed description of the proposed scheme including its forward and backward propagations is provided. Extensive experiments are conducted on multiple benchmark datasets with various settings. The result demonstrates the improvement brought by the proposed scheme on both computational efficiency and classification accuracy, when working with the SPD visual representation learning methods typically involving matrix normalisation. This ReDro scheme can be readily inserted into a network to function, without the need to alter the network architecture or the loss function.

In the future work, we will investigate the effectiveness of the ReDro scheme for more methods designed for learning SPD visual representation and gain more insight into its regularisation effect.

Acknowledgement. This work was supported by the CSIRO Data61 Scholarship; the University of Wollongong Australia IPTA scholarship; the Australian Research Council (grant number DP200101289); and the Multi-modal Australian ScienceS Imaging and Visualisation Environment (MASSIVE).

References

1. Brooks, D., Schwander, O., Barbaresco, F., Schneider, J.Y., Cord, M.: Riemannian batch normalization for SPD neural networks. In: Advances in Neural Information Processing Systems. pp. 15489–15500 (2019)

2. Chakraborty, R., Bouza, J., Manton, J., Vemuri, B.C.: A deep neural network for manifold-valued data with applications to neuroimaging. In: International Conference on Information Processing in Medical Imaging. pp. 112–124. Springer (2019)
3. Cimpoi, M., Maji, S., Vedaldi, A.: Deep filter banks for texture recognition and segmentation. In: Proceedings of the International Conference on Computer Vision and Pattern Recognition. pp. 3828–3836. IEEE (2015)
4. Cui, Y., Zhou, F., Wang, J., Liu, X., Lin, Y., Belongie, S.: Kernel pooling for convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2921–2930 (2017)
5. Demmel, J., Dumitriu, I., Holtz, O.: Fast linear algebra is stable. Numerische Mathematik **108**(1), 59–91 (2007)
6. Engin, M., Wang, L., Zhou, L., Liu, X.: DeepKSPD: Learning kernel-matrix-based spd representation for fine-grained image recognition. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 612–627. Springer (2018)
7. Gao, Y., Beijbom, O., Zhang, N., Darrell, T.: Compact Bilinear Pooling. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 317–326. IEEE (2016)
8. Ghiasi, G., Lin, T.Y., Le, Q.V.: Dropblock: A regularization method for convolutional networks. In: Proceedings of the Advances in Neural Information Processing Systems. pp. 10727–10737 (2018)
9. Hou, S., Wang, Z.: Weighted channel dropout for regularization of deep convolutional neural network. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 8425–8432 (2019)
10. Huang, Z., Van Gool, L.: A riemannian network for SPD matrix learning. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
11. Ionescu, C., Vantzos, O., Sminchisescu, C.: Matrix backpropagation for deep networks with structured layers. In: Proceedings of the International Conference on Computer Vision. pp. 2965–2973. IEEE (2015)
12. Jayasumana, S., Hartley, R., Salzmann, M., Li, H., Harandi, M.: Kernel methods on Riemannian manifolds with Gaussian RBF kernels. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(12), 2464–2477 (2015)
13. Kong, S., Fowlkes, C.: Low-rank bilinear pooling for fine-grained classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 365–374 (2017)
14. Koniusz, P., Wang, L., Cherian, A.: Tensor representations for action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (2020)
15. Koniusz, P., Zhang, H.: Power normalizations in fine-grained image, few-shot image and graph classification. IEEE Transactions on Pattern Analysis and Machine Intelligence (2020)
16. Koniusz, P., Zhang, H., Porikli, F.: A deeper look at power normalizations. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 5774–5783. IEEE Computer Society (2018). https://doi.org/10.1109/CVPR.2018.00605
17. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3D object representations for fine-grained categorization. In: Proceedings of the International Conference on Computer Vision Workshops. pp. 554–561. IEEE (2013)
18. Li, P., Xie, J., Wang, Q., Gao, Z.: Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 947–955. IEEE (2018)

19. Li, P., Xie, J., Wang, Q., Zuo, W.: Is second-order information helpful for large-scale visual recognition? In: Proceedings of the International Conference on Computer Vision. pp. 2070–2078. IEEE (2017)
20. Lin, T.Y., Maji, S.: Improved Bilinear Pooling with CNNs. arXiv preprint arXiv:1707.06772 (2017)
21. Lin, T.Y., RoyChowdhury, A., Maji, S.: Bilinear CNN models for fine-grained visual recognition. In: Proceedings of the International Conference on Computer Vision. pp. 1449–1457. IEEE (2015)
22. Maji, S., Rahtu, E., Kannala, J., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. arXiv preprint arXiv:1306.5151 (2013)
23. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Proceedings of the Advances in Neural Information Processing Systems. pp. 849–856 (2002)
24. Nguyen, X.S., Brun, L., Lézoray, O., Bougleux, S.: A neural network based on SPD manifold learning for skeleton-based hand gesture recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 12036–12045 (2019)
25. Porikli, F., Tuzel, O., Meer, P.: Covariance tracking using model update based on lie algebra. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. vol. 1, pp. 728–735. IEEE (2006)
26. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 413–420. IEEE (2009)
27. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
28. Tompson, J., Goroshin, R., Jain, A., LeCun, Y., Bregler, C.: Efficient object localization using convolutional networks. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 648–656. IEEE (2015)
29. Tuzel, O., Porikli, F., Meer, P.: Region covariance: A fast descriptor for detection and classification. In: Proceedings of the European Conference on Computer Vision. pp. 589–600. Springer (2006)
30. Tuzel, O., Porikli, F., Meer, P.: Human detection via classification on riemannian manifolds. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 1–8. IEEE (2007)
31. Vedaldi, A., Lenc, K.: MatConvNet: Convolutional Neural Networks for Matlab. In: Proceedings of the 23rd ACM International Conference on Multimedia. pp. 689–692 (2015)
32. Wang, Q., Xie, J., Zuo, W., Zhang, L., Li, P.: Deep CNNs Meet Global Covariance Pooling: Better Representation and Generalization. IEEE Transactions on Pattern Analysis and Machine Intelligence (2020)
33. Wang, R., Guo, H., Davis, L.S., Dai, Q.: Covariance discriminative learning: A natural and efficient approach to image set classification. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 2496–2503. IEEE (2012)
34. Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., Perona, P.: Caltech-UCSD Birds 200. Tech. Rep. CNS-TR-2010-001, California Institute of Technology (2010)
35. Yu, K., Salzmann, M.: Second-order convolutional neural networks. arXiv preprint arXiv:1703.06817 (2017)

36. Zhang, H., Zhang, J., Koniusz, P.: Few-shot learning via saliency-guided hallucination of samples. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. pp. 2770–2779. IEEE (2019)
37. Zhang, H., Zhang, L., Qui, X., Li, H., Torr, P.H.S., Koniusz, P.: Few-shot action recognition with permutation-invariant attention. In: Proceedings of the European Conference on Computer Vision (ECCV) (2020)
38. Zheng, H., Fu, J., Zha, Z.J., Luo, J.: Learning deep bilinear transformation for fine-grained image representation. In: Proceedings of the Advances in Neural Information Processing Systems. pp. 4279–4288 (2019)
39. Zhu, X., Xu, C., Hui, L., Lu, C., Tao, D.: Approximated Bilinear Modules for Temporal Modeling. In: Proceedings of the International Conference on Computer Vision. pp. 3494–3503. IEEE (2019)