2022

# Learning Deep SPD Visual Representation for Image Classification

Saimunur Rahman

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

# Learning Deep SPD Visual Representation for Image Classification

Saimunur Rahman

*This thesis is presented as part of the requirements for the conferral of the degree:*

Doctor of Philosophy

Supervisor:
Prof. Lei Wang

Co-supervisors:
Prof. Changming Sun & A/Prof. Luping Zhou

The University of Wollongong
School of School of Computing and Information Technology

June, 2022

This research has been conducted with the support of an International Postgraduate Tuition Award (IPTA) Scholarship, a CSIRO Data61 PhD Scholarship and an University Postgraduate Award (UPA) Scholarship.

# Declaration

I, *Saimunur Rahman*, declare that this thesis is submitted in partial fulfilment of the requirements for the conferral of the degree *Doctor of Philosophy*, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

**Saimunur Rahman**

April 12, 2023

# Abstract

Symmetric positive definite (SPD) visual representations are effective due to their ability to capture high-order statistics to describe images. Reliable and efficient calculation of SPD matrix representation from small sized feature maps with a high number of channels in CNN is a challenging issue. This thesis presents three novel methods to address the above challenge. The first method, called Relation Dropout (ReDro), is inspired by the fact that eigen-decomposition of a block diagonal matrix can be efficiently obtained by eigen-decomposition of each block separately. Thus, instead of using a full covariance matrix as in the literature, this thesis randomly group the channels and form a covariance matrix per group. ReDro is inserted as an additional layer preceding the matrix normalisation step and the random grouping is made transparent to all subsequent layers. ReDro can be seen as a dropout-related regularisation which discards some pair-wise channel relationships across each group. The second method, called FastCOV, exploits the intrinsic connection between eigensytems of $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$. Specifically, it computes position-wise covariance matrix upon convolutional feature maps instead of the typical channel-wise covariance matrix. As the spatial size of feature maps is usually much smaller than the channel number, conducting eigen-decomposition of the position-wise covariance matrix avoids rank-deficiency and it is faster than the decomposition of the channel-wise covariance matrix. The eigenvalues and eigenvectors of the normalised channel-wise covariance matrix can be retrieved by the connection of the $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$ eigen-systems. The third method, iSICE, deals with the reliable covariance estimation from small sized and high-dimensional CNN feature maps. It exploits the prior structure of the covariance matrix to estimate sparse inverse covariance which is developed in the literature to deal with the covariance matrix's small sample issue. Given a covariance matrix, this thesis iteratively minimises its log-likelihood penalised by a sparsity with gradient descend. The resultant representation characterises partial correlation instead of indirect correlation characterised in covariance representation. As experimentally demonstrated, all three proposed methods improve the image classification performance, whereas the first two proposed methods reduce the computational cost of learning large SPD visual representations.

# Acknowledgments

A doctoral thesis is often described as a solitary endeavour; however, the long list that follows proves the opposite. There are many people I must thank for contributing to the four wonderful years of my experience as a doctoral candidate.

First and foremost, I would like to thank Prof. Lei Wang for moulding and chiselling me into a competent researcher from an eager and confusing student through more than a thousand emails and many meetings over the last four years. The passion, wisdom, aspiration and determination for the perfection of Prof. Lei is infectious. Every time I came up with an ambitious research goal he evaluated its feasibility and guided me on how to successfully achieve them. Every time I produced a terrible paper draft he patiently spent hours revising and explaining how to write an excellent paper. Every time I pitched an idea related or not related to my research he uncovered its actual fundamental story by distilling its essence from my pitching and putting forth its wider contexts. I lost two of my family members during my doctoral study and he was there to comfort me and inspire me to continue my research. I am very grateful to Prof. Lei for teaching me not just how to do high quality fundamental computer vision research or the art of writing a good paper and preparing a good presentation, but for how to become a good researcher.

Second, I would like to thank Prof. Changming Sun for patiently revising those terrible paper drafts I sent over the years and teaching me how to maintain consistency in writing. Every time I went to him with an issue related to CSIRO he patiently listened and resolved the issue. Prof. Changming taught me how be sincere about commitments and stay on the goals. I had the privilege once to visit the Marshfield site of CSIRO and while he was showing me around an elderly woman asked for help with the water leakage in her house. We went to that house and spent an hour helping the lady to clean her house. The dedication to helping others I saw in him that day gave me a deep impression. I would like to thank him for this kindness during my doctoral study.

Third, I would like to thank A/Prof. Luping Zhou. I thank her for her valuable comments during my paper draft preparations. I had the privilege of meeting A/Prof. Luping a few times during my doctoral study. Every time, I was fascinated by her thoughts, wisdom in computer vision research and motivating ability for her students. She taught me how to think out of the box and conduct high quality medical image analysis research. I am thankful to her for her kindness during my doctoral study.

Fourth, I would like to thank Dr. Piotr Koniusz. I feel privileged for the chance of collaborating with him during my doctoral study. I am thankful to him for the late night meetings, research and career advice and more importantly, for teaching me how to work with optimisation theories. I also want to thank him for the paper revisions with helpful comments. Dr. Piotr taught me how to work hard on good ideas, how to write systematic and comprehensive paper drafts, how to align novel research ideas with trendy topics and how to think like a researcher working on hard problems. I am grateful to him for the stories from his personal life that have made my doctoral experience memorable and the kindness during my difficult times.

Fifth, I would like to thank CSIRO Data61 for providing me with an excellent scholarship for pursuing a PhD at the University of Wollongong, Australia. I also thank the University of Wollongong, Australia for awarding me the International Postgraduate Tuition (IPTA) and University Postgraduate (UPA) Awards. I am also thankful to CSIRO Scientific Computing, National Computing Infrastructure (NCI) and Multi-modal Australian ScienceS Imaging and Visualisation Environment (MASSIVE) for providing the necessary computing resources to make this thesis successful.

Sixth, I would like to remember my deceased father Late Mr. Habibur Rahman who could not be able to see me finishing the adventure of doctoral study. His encouragement for higher education and philosophies have shaped me into who I am today. I wish he was here with us and feel proud to see me completing the highest academic degree. I would like to thank my younger brother Mr. Minhajur Rahman (Fahad) for his encouragement over the years. Thanks for believing in me and making all the sacrifices after our father's death for my sake. I would like to thank my mother Mrs. Shamim Ara Chowdhury for teaching me life values and raising me differently than others. Without a mother like you, I would not have been a strong man today. I also thank my all cousins who helped us after our father's and grandmother's deaths with warm hearts.

Seventh, I would like to thank the following people who have contributed to my daily life and made my doctoral journey pleasurable. From Visual Information Learning and Analysis Research Group: Huan Wang, Peng Wang, Biting Yu, Zhongyan Zhang, Yu Ding, Din Mohammed Sangrasi, Xiwu Zhang, Jiashuang Huang, Yang Li, Melih Engin, Bela Chakraborty and Zhexuan Zhou. From Engineering and Information Science IT team: Jun Hu. From CSIRO Scientific Computing Team: Peter, Romy, Matthew, Ondrej and a few more. From Engineering and Information Science Research team: Donna Wright. From Graduate Research School: Alicia Allen, Erin Sharp, Lara Duggan, Kirsty Greatz and Kate Kang. From AskUOW team: Emily. From National Computing Infrastructure Support Team: Andrey Bliznyuk, Javed Shaikh and Yue Sun.

Eighth, I am grateful to the following people for their friendship and hospitality: Stephen Kennard and his family, Fawad Mangi, Abdul Malak, Rubel Ahmed, Zhubair Ahmed Ratan, Faisal Ahmed Shuvo, Ashib Ahmed, Minarul Islam, Abdul Aziz Ghazi and more.

# Publications

This thesis is mainly developed based on the following papers which are listed in the order they appear in the thesis.

- **Saimunur Rahman**, Lei Wang, Changming Sun, and Luping Zhou. "ReDro: Efficiently Learning Large-sized SPD Visual Representation." In European Conference on Computer Vision (ECCV), pp. 1-17. Springer, Cham, 2020.
- **Saimunur Rahman**, Lei Wang, Luping Zhou, Piotr Koniusz and Changming Sun. "Efficient Eigen-based Matrix Normalisation for Learning SPD Visual Representation." Under review for a new submission to IEEE Transactions on Pattern Analysis and Machine Intelligence.
- **Saimunur Rahman**, Piotr Koniusz, Lei Wang, Luping Zhou, Peyman Moghadam and Changming Sun. "Learning Partial Correlation based Deep Visual Representation for Image Classification." Accepted in IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

Besides the papers above, the following co-authored paper is also published during my doctoral study. However, the work in this publication is not included in this thesis.

- **Saimunur Rahman**, Lei Wang, Changming Sun, and Luping Zhou. "Deep learning based HEp-2 image classification: A comprehensive review." Medical Image Analysis 65 (2020): 101764.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Feature representation is an important task in computer vision. For the past several decades, extracting semantic information from visual data such as images and videos with low-level descriptors and representing them with a global representation method has been popular. The low-level descriptors are based on pixels or image regions and usually contain information about gradients, edges, intensities, locations, etc. By nature, these low-level descriptors are noisy and large in size so they need to be aggregated into a global representation for achieving compactness and robustness against noise. However, how to effectively aggregate low-level descriptors into a global representation has been a great challenge in many vision tasks. The global representation has to capture information from the descriptors and express a robust summary of them. Furthermore, it should produce a fixed-size representation regardless of the number of descriptors so that consistent comparisons between visual data can be made.

Recently, there has been an interest in the computer vision community for exploiting symmetric positive definite (SPD) matrix as a global representation with Convolutional Neural Networks (CNNs) for end-to-end training [1]–[7]. A well-known example of the SPD matrix is the covariance matrix, which has been successfully used in many visual applications such as image classification [8], action understanding [9] and object detection [10]. There are several advantages of using covariance matrix as a global representation of CNN local descriptors: (1) it exhibits statistical correlations between descriptor components. Its diagonal entries exhibit variances of each descriptor components and off-diagonal entries exhibit covariances between the components of different descriptors; (2) it computes a fixed size representation regardless of the number of components (i.e., features) in the descriptors; (3) it is not affected by the order of features in the descriptors or the information about the number of descriptors. This characteristic is useful to achieve some form of invariance against scaling or rotations in the image data even when the descriptors are itself do not have any such property [8].

Another example of an SPD matrix is the kernel matrix, which has been proposed to characterise statistical correlations between the components of descriptors with a non-linear kernel. It has several advantages over the covariance matrix: (1) it is able to model the non-linear correlation between the components of descriptors. Given a set of CNN descriptors, it computes a kernel matrix with a pre-selected kernel. Each entry of the matrix contains a kernel value realised between the two feature values of all descriptors; (2) its type of non-linear corrections between descriptor components can be flexibly chosen with different kernel functions. The covariance matrix is a special type of kernel matrix which uses linear kernel; (3) it is more robust towards the matrix singularity issue caused by the small sample of CNN descriptors; (4) it has the same size as the covariance matrix. A recent work [6] has demonstrated that it is significantly superior to its covariance matrix counterpart on fine-grained image recognition and scene recognition tasks.

Motivated by the effectiveness of the SPD matrix in characterising second-order information in the visual descriptors, several works integrated it with CNN and conducted end-to-end training [1], [3]–[5]. These works investigated several important issues associated with SPD matrix estimation from CNN feature maps, such as matrix function backpropagation [11] and matrix normalisation [4]. These advancements improved the effectiveness of SPD representation and lead to better visual recognition performance.

Despite the success of visual recognition tasks with the SPD matrix representation, its effective and efficient end-to-end learning with CNN is still a challenging task. The challenges come from the following issues: (1) covariance matrix estimated from a large number of small spatially small-sized convolutional feature maps becomes biased, (2) the size of covariance matrix increases quadratically with the number of channels in a convolutional feature map and (3) eigen-decomposition is often needed to normalise the covariance matrix for each training sample during end-to-end learning. Considering that recent advanced deep neural networks such as ResNet [12] use many spatially small-sized channels in their final convolutional layer, the first factor leads to the unreliability of estimated covariance matrix and the other two factors lead to significant computational overheads.

## 1.1 Research Questions

This thesis focuses on the following two key questions related to the SPD visual representation:

1. Efficient matrix normalisation of large-sized SPD matrix. Matrix normalisation has received considerable attention in recent years for its effectiveness in reducing the swelling effect in SPD matrices computed from CNN feature maps. However, it is computationally challenging to perform matrix normalisation of large SPD matrices on GPU based on eigen-decomposition. This is because the existing eigen-decomposition algorithm has limited GPU acceleration support. This thesis

explores the eigen structure of SPD matrices to reduce the computational burden of performing normalisation using eigen-decomposition in CNN.

2. Integration of sparse inverse covariance matrix with CNN. The covariance and kernel SPD matrices effectively characterise pairwise correlations of CNN descriptors. Recently, it has been shown that sparse inverse covariance is more effective than the covariance and kernel SPD matrices in dealing with small feature samples and higher dimensionality [13]. Sparse inverse covariance is estimated by solving an optimisation problem involving the minimisation of a log-likelihood penalised by sparsity. How to integrate the sparse inverse covariance into the CNN is still an unexplored issue. This thesis explores a way to integrate sparse inverse covariance into the CNN and conduct end-to-end training.

## 1.2 Contributions

The key contributions of this thesis are as follows.

1. This thesis proposes a scheme called Relation Dropout (ReDro) to efficiently carry out matrix normalisation for end-to-end learning of large SPD visual representation. It is inspired by the fact that eigen-decomposition of a block diagonal matrix can be efficiently obtained by eigen-decomposition of each block separately. Thus, instead of using a full covariance matrix as in the literature, this scheme randomly groups the channels and forms a covariance matrix per group. ReDro is inserted into the CNN as an additional layer preceding the matrix normalisation step and makes its random grouping transparent to all subsequent layers. It can be seen as a dropout-related regularisation which discards some pair-wise channel relationships in each group. Experimental studies on several image datasets reveal that ReDro can be beneficial to significantly reduce the computational cost of performing matrix normalisation of large SPD matrices using eigendecomposition without sacrificing performance.

2. This thesis proposes another scheme called FastCOV to efficiently carry out matrix normalisation for end-to-end learning of large covariance matrix representation. It exploits the intrinsic connection between eigensystems of $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$. Specifically, it computes position-wise covariance matrix upon convolutional feature maps instead of the typical channel-wise covariance matrix. As the spatial size of feature maps is usually much smaller than the channel size, conducting eigen-decomposition of the position-wise covariance matrix avoids rank deficiency, while being faster than the decomposition of the channel-wise covariance matrix. Finally, the eigenvalues and eigenvectors of the normalized channel-wise covariance matrix are retrieved by the connection of the $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$ eigen-systems. Experimental studies reveal

that FastCOV is significantly faster than the ReDro scheme in computing covariance representation when it is used with CNNs that has small spatial sized feature maps with the higher number of channels. Furthermore, it achieves competitive performance with respect to the existing methods.

3. This thesis integrates sparse inverse covariance estimation as a novel structured layer into CNN. To realise end-to-end training of the resultant CNN, this thesis develops an iterative method based on Newton-Schulz iteration to solve the sparse inverse covariance estimation during backpropagation. By doing so, this thesis mitigates the small sample problem for the covariance estimation in CNN. On top of that, the developed method is fully compatible with GPU and with the presence of a large number of CNN feature channels. Extensive experiments on various hyperparameters of the proposed method are performed using one scene and three fine-grained image datasets to assess its robustness. Experimental results confirm that the proposed method significantly outperforms its covariance matrix based counterparts.

## 1.3 Outline of this Thesis

Chapter 1 of this thesis begins with a general introduction of SPD visual representation, which enables to discuss its several key issues when integrated with deep neural networks, specially CNNs. In Chapter 2, a detailed discussion on how SPD visual representation is computed from local CNN descriptors in the recent literature is given for a better understanding of the problems targeted in this thesis. Since SPD visual representation has already been used for many computer vision applications, a brief overview of current application areas of SPD representation is also provided in this chapter.

Chapter 3 and 4 present two methods from different perspectives to mitigate the high computation time of large-sized SPD matrix normalisation and facilitate efficient end-to-end CNN training. Specifically, Chapter 3 presents the Relation Dropout method which performs matrix normalisation on small block-diagonal SPD matrices instead of the full SPD matrix to improve the training efficiency of large-sized SPD matrices. Chapter 4 presents the FastCOV method which performs matrix normalisation on the position-wise matrix instead of the channel-wise matrix to improve the training efficiency of large-sized SPD matrices. In deep networks such as ResNets [12] where the spatial resolution of feature maps is significantly smaller than the number of feature channels, the FastCOV method can greatly improve the SPD matrix computation efficiency. The Relation Dropout method does not take the spatial resolution of feature maps into the account but still reduces the SPD matrix computation time by resorting to the block-diagonal matrix trick.

While the above two chapters make an effort to efficiently deal with large-sized SPD matrices, there are tools in the literature such as Sparse Inverse Covariance Estimate (SICE) to deal with the reliable SPD matrix estimate from the small number of samples which is commonly encountered in CNNs. Chapter 5 presents Iterative Sparse Inverse Covariance Estimation (iSICE) method which targets estimating reliable covariance matrix based SPD visual representation. SICE can not be directly integrated into deep networks due to the unavailability of efficient and differentiable solvers in the literature. iSICE presents a scalable GPU-compatible solver and a framework for computing sparse inverse covariance based SPD visual representation. Extensive experiments demonstrate that iSICE estimates better SPD representation than the existing methods.

In the final chapter (Chapter 6), a conclusion is given outlining a summary of each chapter and the key contributions of this thesis to the existing literature. In addition to that, future extensions of the proposed methods in Chapters 3, 4 & 5 and research directions are discussed. For a better understanding of the contents of each chapter, a logical connection between chapters is given in Figure 1.1 and a chapter-wise summary is given below.

- Chapter 2 gives an overview of literature related to the SPD matrix based representation (or SPD representation, in short). It introduces global feature representation in CNNs and presents SPD representation as a second-order global feature representation. Then it gives a discussion on how the existing methods are approaching the key issues of SPD representation learning with CNN. Finally, it discusses some application areas of SPD representation. The chapter also gives a visual illustration of recent research progress of CNN based end-to-end SPD representation.

- Chapter 3 proposes the ReDro scheme. It introduces the key motivations for ReDro and discusses the related literature. Following that, the working principle of ReDro and how it helps to reduce the computational cost of performing matrix normalisation on large SPD matrices are discussed. Then, its forward and backward propagations are derived, and the key parameters are discussed. Three major sets of experiments are presented. The first set of experiments shows the computational advantages of ReDro when integrated with two SPD matrix based representations, i.e., covariance and kernel matrix based representations. Then it shows experiments using and not using ReDro with two CNN backbones and four popular datasets. Finally, it shows experiments where ReDro is integrated with a few existing SPD representation methods. In addition to the above experiments, it also presents an ablation study on key parameters of ReDro. The experimental study shows that ReDro can effectively improve the efficiency of eigen-decomposition based matrix normalisation of large sized SPD matrices.

- Chapter 4 proposes the FastCOV scheme. Similar to the previous chapter, it starts by introducing the motivation and the related works. Then it introduces the FastCOV scheme and discusses the forward and backward propagations with CNN. After that, it gives a discussion on the computations savings with FastCOV from a theoretical viewpoint, i.e., computational complexity. Finally, it presents the experiments with two popular CNN backbones. The experiments are divided into three parts. The first part contains the experiments related to the computational advantage of FastCOV over ReDro and existing SPD representation methods. The second part compares the performance of FastCOV with regular covariance matrix based representation. The third part compares FastCOV with ReDro and state-of-the-art SPD representation methods. The experimental studies demonstrate that FastCOV can efficiently perform matrix normalisation of covariance matrices than the existing methods.

- Chapter 5 proposes a method for learning sparse covariance matrix representation with CNN. Starting with motivation, it gives an overview of the current efforts in estimating robust covariance matrix representation. Then it discusses the key idea of sparse covariance matrix representation and the challenges of integrating it onto CNN with the existing optimisation problem solvers. After that, it presents the proposed idea of learning sparse covariance matrix estimation with CNN and discusses its details. Finally, it shows the experiments where the following items are comprehensively discussed: (1) the robustness of proposed methods with respect to key hyper-parameters; (2) the performance of medium and large sized sparse covariance matrix representations; (3) comparison with the methods from previous chapters and state-of-the-art SPD methods. The experimental study shows that the proposed idea can effectively improve the existing covariance representation with the sparse inverse covariance estimation and lead to better image recognition performance.

- Chapter 6 summarises the key contributions in this thesis and discusses future works.

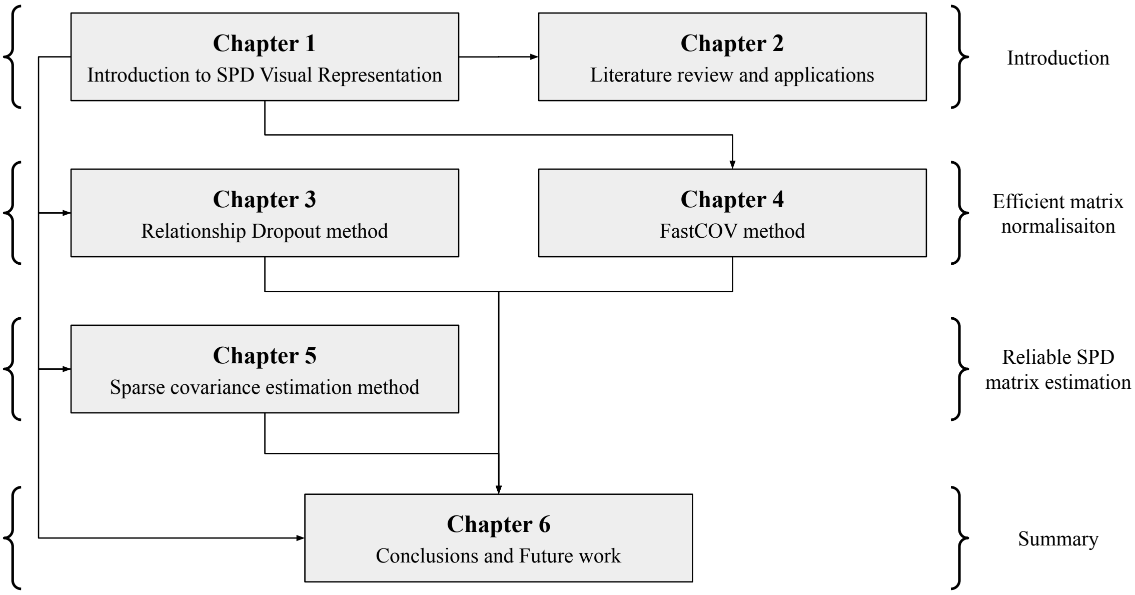
Chapter 1
Introduction to SPD Visual Representation
Chapter 2
Literature review and applications
Introduction
Chapter 3
Relationship Dropout method
Chapter 4
FastCOV method
Efficient matrix normalisaiton
Chapter 5
Sparse covariance estimation method
Reliable SPD matrix estimation
Chapter 6
Conclusions and Future work
Summary

**Figure 1.1:** Logical connections between chapters.

# Chapter 2

# Literature Review

Learning good visual representation remains a challenging task in image classification due to variations in illumination, geometry, and background in image data. Describing images with local feature descriptors and aggregating them into a global representation with an appropriate pooling method remains an effective approach. The rise of deep learning based methods, in particular, the ones based on convolutional neural networks (CNNs) further enhanced this approach with automatic feature learning. This thesis focuses on obtaining effective global representations based on the symmetric positive definite (SPD) matrix with CNN. This chapter begins with an introduction to the common CNN based global representation methods. Then it will discuss the major developments in SPD matrix based representation learning with CNN. Finally, it will discuss some application areas of SPD matrix based representation.

## 2.1 Global Image Representation Learning with CNN

The idea of a global image representation from a set of local visual descriptors has been in the literature for more than two decades. Some of the popular methods for obtaining global image representation from a set of local image features include Fisher Vectors (FV) [14], Vector of Locally Aggregated Descriptors (VLAD) [15] and Sparse Coding (SC) [16].

Suppose, an image $I$ is represented using a set of local feature descriptors $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \subset \mathbf{R}^d$. These methods aggregate these descriptors into a global image representation $\psi(I)$ and ensure that $\psi(I)$ is compact and discriminative. Generally, $\psi(I)$ uses two steps: embedding and aggregation. In the embedding step, the local descriptors $\mathbf{x}_n$ are mapped to higher dimensional vectors with a mapping function $\phi(\cdot)$ as $\phi(\mathbf{x}) \in \mathbf{R}^D$. Then the aggregation step aggregates the mapped vectors $\{\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), ..., \phi(\mathbf{x}_n)\} \subset \mathbf{R}^D$. One simple operation for the aggregation step could be a summation operation $\psi(I) = \sum_{i=1}^{N} \phi(\mathbf{x}_i)$, however, more complex operations such as Fisher kernel [14] can be used.

The aggregation step plays an important role in obtaining an effective global image representation. To obtain better global image representations from CNN feature descriptors, a large amount of research has been conducted on feature aggregation in recent years. Some of the popular feature aggregation methods (also known as pooling methods) used in existing CNN architectures such as AlexNet [17], VGGNet [18], GoogleNet [19], ResNet [12] and DenseNet [20] include max pooling, average pooling and global average pooling. The latter received much attention recently due to its effectiveness with a large number of feature channels commonly seen in ResNet-inspired architectures such as ResNet-101. Given a convolutional feature channel $\mathbf{x} \in R^{n \times n}$ of the same width and height, the global average pooling computes an output $\mathbf{y}$ as follows.

$$\mathbf{y} = \frac{1}{n \times n} \sum_{i=1,j=1}^{n} x_{ij} \tag{2.1}$$

The above common aggregation methods characterise first-order information in the descriptor components. In this thesis, we refer to them as first-order feature aggregation methods or simply first-order representation methods. In traditional image classification problems such as ImageNet classification [17], first-order representation methods have shown impressive performance. However, in more difficult problems such as fine-grained image classification where the availability of labelled training images is limited, higher-order representation methods are found to be more effective [4]. They characterise higher-order, i.e., pairwise, triple-wise, or even higher, correlations between descriptor components which help to improve the classification performance.

As a higher-order representation method, the SPD matrix based representation methods has shown promising performance in recent years. They characterise pairwise, i.e., second-order, correlations in the descriptor components. The most common SPD matrix based representation method in the literature is covariance matrix based representation. Given a set of feature descriptors $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \in \mathbf{R}^d$, covariance matrix $\mathbf{\Sigma}$ represents them as a $d \times d$ matrix

$$\mathbf{\Sigma} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^\top, \tag{2.2}$$

where $\mu = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$. The matrix $\mathbf{\Sigma}$ contains the second-order statistical information of feature descriptors. The off-diagonal and diagonal entries of $\mathbf{\Sigma}$ represent the covariance and variance between the descriptor components, respectively. Another key feature of $\mathbf{\Sigma}$ is it is symmetric, therefore, only its upper or lower triangular entries can be used as a representation. The other type of SPD matrix representation used in the literature is kernel matrix-based representation which depicts nonlinear correlations between descriptor components. A covariance matrix can be regarded as a special case of a kernel matrix, where a linear kernel is used for characterising feature relationships. This thesis focuses on

SPD matrix based representation (or 'SPD representation', in short) learning with CNN. Below we discuss its developments in recent years.

## 2.2 SPD Representation Learning with CNN

After the promising success of covariance representation with pre-extracted CNN descriptors on texture classification [21] and material recognition [22], several pioneering works developed end-to-end trainable CNN based covariance representation frameworks. Among them, Bilinear CNN [1] and DeepO$_2$P [11] have inspired many researchers to make the SPD representation more effective under an end-to-end learning framework.

One of the key focuses of these frameworks was reliable covariance matrix estimation from low resolution feature maps and a larger number of feature channels. Modern CNNs have less number of features and large feature dimensions which makes the covariance matrix biased. When the covariance matrix becomes biased, its large eigenvalues become larger and its smaller eigenvalues become smaller, resulting in unreliability. This issue is also known as the swelling effect [23]. Bilinear CNN has applied element-wise square root normalisation to each entry of the covariance matrix to combat this issue by considering the Riemannian geometry of the covariance matrix, but it did not help them to achieve a very good performance. The covariance matrix resides on a Riemannian manifold, therefore, its geometrical structure must be considered before applying normalisation. DeepO$_2$P proposed matrix logarithm normalisation of eigenvalues to combat the matrix swelling issue. It also shows gradient rules for performing matrix normalisation on the eigenvalues with backpropagation. Though their work has achieved better performance than the Bilinear CNN, the matrix logarithm has an issue of changing the magnitudes of eigenvalues which affects their significance (a comprehensive analysis on this is available in [4]).

Motivated by the drawbacks of matrix logarithm, the idea of matrix power normalisation is proposed in [3], [4]. Given the covariance matrix $\mathbf{\Sigma}$, the matrix power normalisation is performed by taking the square root of the eigenvalues

$$\mathbf{\Sigma}^{\alpha} = \mathbf{U}\mathbf{D}^{\alpha}\mathbf{U}^{\top}, \tag{2.3}$$

where $\mathbf{U}$ and $\mathbf{D}$ are the eigenvectors and eigenvalues, respectively, and $\alpha$ is set to 0.5 for performing square rooting. Matrix power normalisation does not change the magnitudes or the significance of the eigenvalues, therefore, it ensures the reliability of the estimated covariance matrix from small samples. One of the key issues with matrix power normalisation is the computation of eigenvalues and eigenvectors with eigen-decomposition. Existing eigen-decomposition algorithms are not well paralleisable which limits their fast computation with both the central processing unit (CPU) and graphics processing unit

| Method | Normalisation | Complexity | Support GPU | Compactness |
|---|---|---|---|---|
| DeepO$_2$P [11] | Matrix Logarithm | $\mathcal{O}(d^3)$ | Unsatisfactory | Unsatisfactory |
| MPN [4] | Matrix Sqrt. Root | $\mathcal{O}(d^3)$ | Unsatisfactory | Unsatisfactory |
| I-BCNN [3] | Matrix Sqrt. Root | $\mathcal{O}(d^3)$ | Unsatisfactory | Unsatisfactory |
| iSQRT [5] | Matrix Sqrt. Root | $\mathcal{O}(kd^3)$ | Satisfactory | Unsatisfactory |
| DeepKSPD [5] | Matrix $\alpha$ Root | $\mathcal{O}(d^3)$ | Unsatisfactory | Unsatisfactory |
| RUN [24] | Matrix Sqrt. Root | $\mathcal{O}(kd^3)$ | Satisfactory | Satisfactory |

**Table 2.1:** Summary of existing matrix normalisation methods. Satisfactory and unsatisfactory mean limited and good, respectively. $k$ is the number of iterations and $d$ is the number of feature channels.

(GPU). In a survey by Li et al. [5], it has been demonstrated that eigen-decomposition with CPU is much faster than the GPU. This computational burden increases with the size of the covariance matrix which will be shown in the following chapters.

To improve the above condition with eigen-decomposition, Lin et al. [3] proposed to perform matrix power normalisation with approximate square root based on Newton-Schulz iterations instead of the exact root. They show that performing matrix power normalisation on CPU with approximate square root using a few Newton-Schulz iterations is faster than eigen-decomposition. Given $\mathbf{A}_0 = \mathbf{\Sigma}$ and $\mathbf{B}_0 = \mathbf{I}$, Newton-Schulz iterations normalise $\mathbf{\Sigma}$ with its approximate root as follows

$$\mathbf{A}_{i+1} = \frac{1}{2}\mathbf{A}_i(3\mathbf{I} - \mathbf{B}_i\mathbf{A}_i), \mathbf{B}_{i+1} = \frac{1}{2}(3\mathbf{I} - \mathbf{B}_i\mathbf{A}_i)\mathbf{B}_i, \tag{2.4}$$

where $\mathbf{I}$ is an identity matrix, and $\mathbf{A}_i$ and $\mathbf{B}_i$ quadratically converge to $\mathbf{\Sigma}^{\frac{1}{2}}$ and $\mathbf{\Sigma}^{-\frac{1}{2}}$, respectively. They also show that matrix power normalisation with approximate square root can achieve similar performance as with the exact root in significantly less time. However, the method proposed in [3] computes square root with Newton-Schulz iterations only in the forward propagation of CNN. During the backward propagation, they still need to perform eigen-decomposition or solve a Lyapunov equation which takes almost equal time as eigen-decomposition.

Motivated by the limitation of work in [3], Li et al. [5] proposed a method to perform Newton-Schulz iterations in both forward and backward propagations. They showed that Eq. (2.4) and its gradients can be efficiently computed with GPU. Furthermore, they solved the numerical instability issue encountered in the work of [3]. Their method inspired other researchers to develop more efficient iterative matrix normalisation methods. For example, normalisation with the rank-1 update [24] considers multiplication between matrix and vector for more efficient computation.

Another focus in existing frameworks is learning kernel matrix representation. A key work in this direction is kernel pooling [25]. They use a Taylor expansion of the RBF kernel on convolutional features and show promising performance with a fourth-order kernel. Another key work is DeepKSPD [6] which computes an RBF kernel based second-order matrix representation and proposes to normalise it with a $\alpha$-square root normalisation. $\alpha$-square root normalisation is different from the square root normalisation used in the previous works in two ways, 1) the value of $\alpha$ is learned with backpropagation, therefore, the matrix power can be adjusted according to the data instead of fixing it to $\frac{1}{2}$ and 2) it is free from the numerical instability issues faced in [3], [4]. We summarise the above progress on matrix normalisation in Table 2.1.

While the above works focus on computing effective SPD representation, a few researchers worked on the problem of higher dimensionality incurred with SPD representation since it outputs $d(d+1)/2$ dimensional vectors, where $d$ is the number of feature channels. In recent CNNs, $d$ can be as high as 2048. These researchers mainly concentrate on reducing the dimension of SPD representation, specifically, covariance matrix representation. The first key work in this direction is compact bilinear pooling [2]. It uses tensor sketch [26] and random Maclaurin [27] projections for reducing the dimension of Bilinear CNN without compromising its performance. Another key work in this direction is low-rank bilinear pooling [28]. It uses a factorised low-rank bilinear classifier for performing classification with bilinear representation. These two works do not consider matrix normalisation in their proposed methods. Motivated by the performance improvements brought by the matrix square normalisation in the works of [3]–[5], the following methods have been developed to perform matrix normalisation methods on compact SPD representations. The existing matrix square normalisation methods discussed above can not be applied to compact SPD representations since they do not have a strict SPD or square matrix structure.

The first method is based on rank-1 matrix update [24]. It performs the normalisation of convolutional feature maps prior to the computation of compact covariance representations. Since the normalisation method involves only matrix and vector multiplication, it takes less time than the Newton-Schulz iterations [5]. The second method is based on Newton-Schulz iterations and shifted random Maclaurin [7]. It applies shifted random Maclaurin projection to the normalised covariance matrix for computing compact covariance representation. The performance and efficiency of work in [24] is better than the work in [7]. More relevant work in this direction is by Li et al. in [29]. After applying matrix square root normalisation to the covariance matrix and then vectoring, they used group convolution [17] to reduce the final dimension. Since the idea is straightforward and relies on linear projection, the selection of the number of convolutional kernel groups must be decided carefully for better performance.

In Figure 2.1, we summarise the current research progress on SPD representation discussed above. We show how the end-to-end SPD representation frameworks have

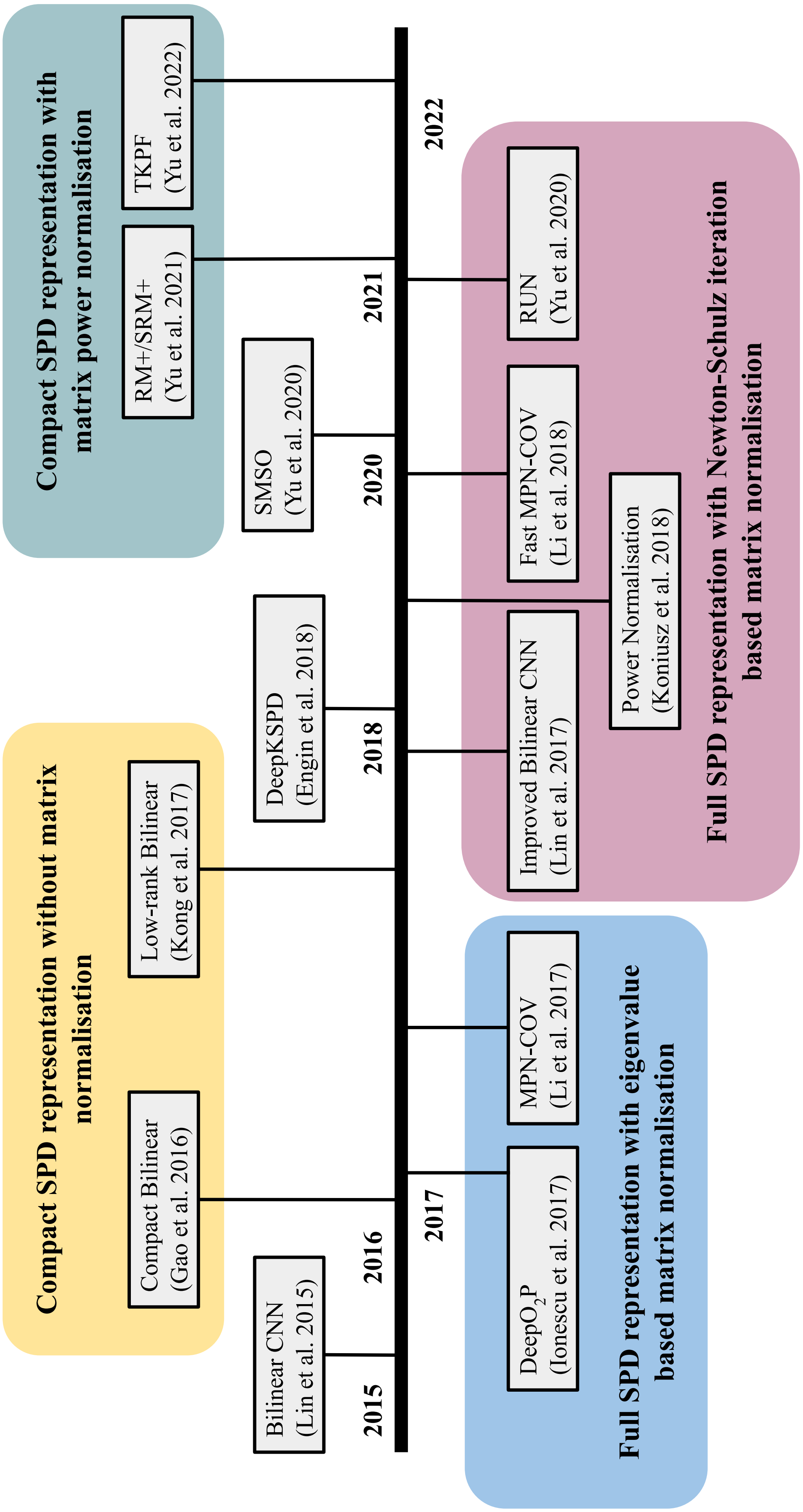
Compact SPD representation with matrix power normalisation
TKPF (Yu et al. 2022)
RM+/SRM+ (Yu et al. 2021)
SMSO (Yu et al. 2020)
Compact SPD representation without matrix normalisation
Low-rank Bilinear (Kong et al. 2017)
Compact Bilinear (Gao et al. 2016)
DeepKSPD (Engin et al. 2018)
Bilinear CNN (Lin et al. 2015)
2015
2016
2017
2018
2020
2021
2022
Full SPD representation with Newton-Schulz iteration based matrix normalisation
RUN (Yu et al. 2020)
Fast MPN-COV (Li et al. 2018)
Power Normalisation (Koniusz et al. 2018)
Improved Bilinear CNN (Lin et al. 2017)
Full SPD representation with eigenvalue based matrix normalisation
MPN-COV (Li et al. 2017)
DeepO$_2$P (Ionescu et al. 2017)

**Figure 2.1:** An overview of end-to-end SPD representation learning framework research progress over the years.

evolved over the years.

## 2.3 Application of SPD Representation

As a generic representation, SPD Representation can be used in any computer vision application. However, in application areas such as fine-grained image classification where the number of labelled training data is limited, SPD representation is used to exploit the relationship between feature descriptors so that the performance can be improved. Some of the application areas of end-to-end learned SPD representation include fine-grained image classification, generic image classification, few-shot learning, image retrieval, action understanding, image segmentation, video classification, image set classification, person re-identification, domain adaptation, visual question answering and so on. Below we give a brief discussion on a few of them.

**Fine-grained image classification.** It is one of the most active areas to use SPD representation. Classification of fine-grained image datasets remains a challenging task due to their lower intra-class variations and higher inter-class variations. In addition, the number of images in fine-grained datasets is also limited. In fine-grained image classification, SPD representation is used to characterise second-order information of visual descriptors. The second-order information acts as prior knowledge of objects in the images which helps to improve classification performance. Some popular and recent SPD representation methods for fine-grained image classification are Bilinear CNN [3], Compact BCNN [2], DeepKSPD [6], iSQRT-COV [5], G2DeNet [30], HBP [31] GSoP-Net [32], MOMN [33], SVD by Padé Approximants [34], squeezed bilinear pooling [35], semantic bilinear pooling [36], MOFS [37], Kernelized random projection based compact bilinear pooling [38], FBC [39], Hierarchical Biquadratic Pooling [40], adaptive bilinear pooling [41], WAM [42] and HM-CNN [43].

**Large-scale image classification.** Covariance representation has been successfully used for large-scale ImageNet classification [17]. Similar to fine-grained image classification, it exploits second-order information in the visual descriptors to improve the image classification performance. CNN models that have been trained with covariance representation are found to be more effective during the transfer learning tasks, e.g., [4]. Some notable examples of SPD representation based large-scale image classification methods are MPN-COV [4] and iSQRT-COV [5], MPA-Lya [44], SVD-Taylor [45] and SVD by Padé Approximants [34].

**Single and Few-shot learning.** Single and few-shot learning are two challenging areas of computer vision. The challenges are due to the availability of limited training samples.

SPD representation has been successfully used for single and few-shot learning tasks. It helps one and few-shot learning frameworks to capture rich second-order information from limited visual data in order to achieve better performance. Some key examples of SPD representation based single and few-shot learning methods are SoSN [46], SalNet [47], MsSoSN [48], LR-PABN [49], MlSo [50], DSN [51] and SmCT [52].

**Action recognition.** It is one of the popular areas in computer vision. One of the primary challenges in this area is spatio-temporal encoding of visual information. SPD representation has been used to characterise second-order information from the action descriptors. The works in [53], [54] and [55] have confirmed that incorporation of second-order kernel based representation improves the classification performance of actions in well-known datasets.

**Person Re-identification.** Similar to action recognition, person re-identification in video has become one of the popular research areas in computer vision due to its application in crime prevention. SPD representation has been found to be effective in improving the person re-identification performance with the characterisation of second-order information in cases where persons are generally hard to distinguish. An example of a person re-identification method with SPD representation is SCCT [56]. It uses second-order colour transformation information to improve classification performance. Another example is HOReID [57].

**Domain adaptation.** Generalisation of knowledge learned for a task in a specific environment and applying it to perform the same task in a slightly different environment is of great interest in machine learning. SPD representation has been found to be effective in performing the above goal [58]. In the work of [58], it has been shown that higher-order information encoding can have a positive impact on performing better domain adaptation.

**Visual attention learning.** Recently, visual attention mechanism learning has become popular due to its wide application with CNNs. It helps the network to learn better representation. Second-order information based visual attention mechanisms has been proposed in the literature to exploit the rich higher-order statistics in the data. Several works have shown that second-order attention maps help the network learn better features. Examples of some notable second-order notable methods include mixed attention [59], SAN [60], GSoP [32], SONA [61], SOLAR [62] and bilinear attention [63].

**Student-teacher learning.** It is an efficient mechanism for knowledge transfer from large to small machine learning models and has a wide application. Recently, Li et al. have successfully used SPD representation to improve the performance of CNN using an intuition from the student-teacher learning mechanism. Their method shows how second-order information can be only used at the training stage to learn better features. The method achieves competitive performance on a large-scale dataset.

**Visual-question answering.** It is one of the challenging research areas in computer vision. One of the key problems in this area is how to combine visual and text information and represent them in a shared space. An early work by Akira et al. [64] has shown that fusing multi-modal information to a common feature space using covariance representation leads to better performance. Their work demonstrated good performance using SPD representation of multi-modal data on visual-question answering. Following this work, many researchers such as FBC [39] employed SPD representation.

## 2.4 Conclusion

This chapter summarises recent progress on end-to-end learned SPD visual representation. It discusses the key challenges associated with SPD matrix estimation from CNN feature maps and how existing methods are solving those challenges. It gives a detailed discussion of research directions for SPD matrix normalisation, a step to reduce the impact of the swelling effect. In the end, it discusses some application areas of SPD representation.

# Chapter 3

# ReDro for Efficiently Learning Large-sized SPD Visual Representation

> *The preliminary work of this chapter has been published at the 2020 European Conference on Computer Vision (ECCV 2020) and an extended version is currently under the review of IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI).*

Symmetric positive definite (SPD) matrix has recently been used as an effective visual representation. When learning this representation in deep networks, eigen-decomposition of the covariance matrix is usually needed for a key step called matrix normalisation. This could result in significant computational costs, especially when facing the increasing number of channels in recent advanced deep networks. This chapter 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.

## 3.1 Introduction

Learning good visual representation remains a central issue in computer vision. Representing images with local descriptors and pooling them into 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 the 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 [1], image segmentation [11], generic image classification [4], [65], image set classification [66], activity and action recognition [67], [68] and few-shot learning [47], [69], 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 [2], matrix normalisation [3], [70], [71] 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 the covariance matrix increases quadratically with the channel number in a convolutional feature map and ii) eigendecomposition 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 [72]. Each diagonal matrix is smaller in size and the eigendecomposition 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 [73]), 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 [74]. 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 chapter are summarised as follows.

- To mitigate the computational issue in learning large-sized SPD representation for the methods using matrix normalisation, this chapter 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.

## 3.2 Related Work

In this section, we review the methods related to SPD visual representation learning, SPD matrix normalisation and dropout regularisation.

**Learning SPD visual representation.** SPD visual representation can be traced back to covariance region descriptor in object detection, classification and tracking [8], [10], [75]. 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 [21], 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.

Deep$O_2$P [11] and Bilinear CNN (BCNN) [1] are two pieces of pioneering work that learn SPD visual representation in an end-to-end manner. The framework of Deep$O_2$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 the small sample.

After element-wise normalisation, the recent work turns to matrix-logarithm or matrix-power normalisation because they usually produce better SPD representation.[a] 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 [3], 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 [5] 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 [3], [5] achieves computational advantage and promising performance by using matrix square-root, their methods do not generalise to matrix normalisation

[a] 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 $f(x) = x^p$.

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 $256 \times 256$) 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 [2], [28], [73] address the high dimensionality of the feature representation after vectorising covariance matrix, and group convolution is used in [29] for the similar purpose. Linear transformation is designed in [29], [76]–[79] to project large covariance matrices to more discriminative, smaller ones. To efficiently capture higher-order feature interactions, kernel pooling is developed [25]. Furthermore, the work in [80] 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, we focus on the frameworks in [3]–[6] which typically employ matrix normalisation as an important step in learning SPD representation.

**Dropout schemes.** Dropout [74] 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 [81] randomly drops feature channels. DropBlock [82] drops a block of pixels from convolutional feature maps. Weighted channel dropout [83] 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.

## 3.3 The proposed method

In this section, we describe our relation dropout (ReDro) method. We also provide their forward and backward propagation steps. Furthermore, we discuss their computational advantages and key parameters.

### 3.3.1 Relation Dropout (ReDro)

We start by providing an overview of our first proposed scheme, called ReDro, for matrix normalisation, which is shown in Figure 3.1. From the left end, an image is fed into a CNN backbone, and the last convolutional feature map of $d$ channels is 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 $i$ ($i = 1, 2, \cdots, k$), smaller-sized covariance matrices $\mathbf{C}_i$ are computed and their eigen-decompositions evaluated 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 same 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 $\mathbf{D}_b$ are *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 functions.

#### Forward propagation in the presence of ReDro

Let $\mathbf{X}_{d\times n}$, where $n = wh$, 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.

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 a permutation matrix $\mathbf{P}_{d\times d}$.[b] The $i$th 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 $r_i$th 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 a 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.

[b] 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.

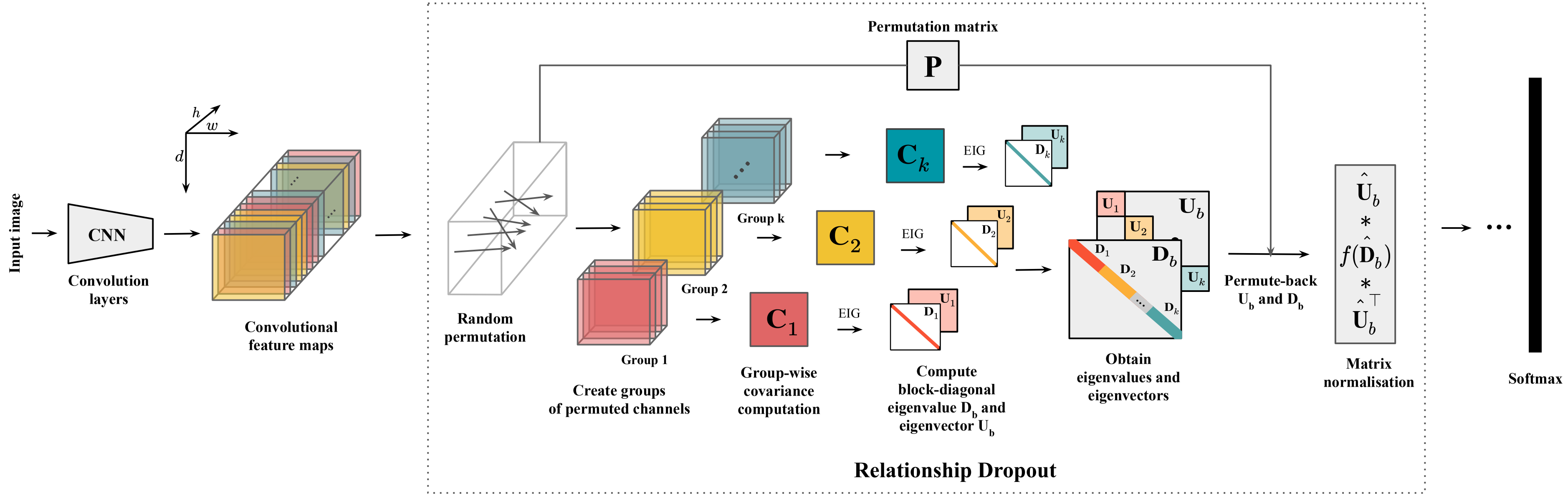
Input image
CNN
Convolution layers
$h$
$w$
$d$
Convolutional feature maps
Permutation matrix
P
Random permutation
Group 1
Group 2
Group k
Create groups of permuted channels
$\mathbf{C}_1$
$\mathbf{C}_2$
$\mathbf{C}_k$
Group-wise covariance computation
EIG
$\mathrm{U}_1$ $\mathrm{D}_1$
$\mathrm{U}_2$ $\mathrm{D}_2$
$\mathrm{U}_k$ $\mathrm{D}_k$
Compute block-diagonal eigenvalue $\mathrm{D}_\mathrm{b}$ and eigenvector $\mathrm{U}_\mathrm{b}$
$\mathbf{U}_b$
$\mathbf{D}_b$
Obtain eigenvalues and eigenvectors
Permute-back $\mathrm{U}_\mathrm{b}$ and $\mathrm{D}_\mathrm{b}$
$\hat{\mathbf{U}}_b * f(\hat{\mathbf{D}}_b) * \hat{\mathbf{U}}_b^\top$
Matrix normalisation
Relationship Dropout
...
Softmax

**Figure 3.1:** Proposed relation dropout (ReDro) scheme for mitigating the computational issue of matrix normalisation in the end-to-end learning of large-sized SPD visual representation. EIG denotes eigen-decomposition.

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

$$\mathbf{C}_b = \text{diag}(\mathbf{C}_1, \mathbf{C}_2, \cdots, \mathbf{C}_k). \tag{3.1}$$

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 from matrix analysis [72] that the eigen-decomposition of $\mathbf{C}_b$ can be expressed as

$$\mathbf{C}_b = \mathbf{U}_b \mathbf{D}_b \mathbf{U}_b^\top, \tag{3.2}$$

$$\begin{aligned} \mathbf{U}_b &= \text{diag}(\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k) \\ \text{and } \mathbf{D}_b &= \text{diag}(\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k). \end{aligned} \tag{3.3}$$

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

$$\hat{\mathbf{U}}_b = \mathbf{P}^{-1} \mathbf{U}_b = \mathbf{P}^\top \mathbf{U}_b. \tag{3.4}$$

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.

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

$$\begin{aligned} \hat{\mathbf{U}}_b &= \mathbf{P}^\top \cdot \text{diag}(\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k); \\ \hat{\mathbf{D}}_b &= \text{diag}(\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k). \end{aligned} \tag{3.5}$$

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

### Backward propagation in the presence of ReDro

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

$$\mathbf{X} \rightarrow \underbrace{\mathbf{PX}}_{\mathbf{Y}} \rightarrow \underbrace{(\mathbf{YY}^\top) \circ \mathbf{S}}_{\mathbf{C}_b} \rightarrow \underbrace{(\mathbf{P}^\top \mathbf{C}_b \mathbf{P})}_{\mathbf{A}\text{(Auxiliary)}} \rightarrow \underbrace{f(\mathbf{A})}_{\mathbf{Z}} \rightarrow \cdots \text{layers} \cdots \rightarrow \underbrace{J(\mathbf{X})}_{\text{Objective}} \tag{3.6}$$

$\mathbf{P}$ is the permutation matrix. $\mathbf{Y}$ is the feature map with its channels permuted. $\mathbf{S} = \text{diag}(\mathbf{1}_1, \mathbf{1}_2, \cdots, \mathbf{1}_k)$ is a block-diagonal binary matrix, where $\mathbf{1}_i$ is a square matrix of all

```
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 $\mathbf{C}_b$:
   $\mathbf{U}_b = \text{diag}(\mathbf{U}_1, \mathbf{U}_2, \cdots, \mathbf{U}_k), \mathbf{D}_b = \text{diag}(\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k)$;
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 $\hat{\mathbf{D}}_b = \mathbf{D}_b$.
```

"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 derive $\frac{\partial J}{\partial \mathbf{X}}$. Once obtained, all the gradients before the convolutional feature map $\mathbf{X}$ can be obtained.

According to the process shown in Equation (3.6), $J$ is a composite function of $\mathbf{X}$ and it can be equally expressed as a function of each of the intermediate variables as follows.

$$J(\mathbf{X}) = J_1(\mathbf{Y}) = J_2(\mathbf{C}_b) = J_3(\mathbf{A}) = J_4(\mathbf{Z}) \tag{3.7}$$

By the rules of differentiation, the following results can be obtained

$$\delta\mathbf{Y} = \mathbf{P}\delta\mathbf{X}, \tag{3.8}$$

$$\delta\mathbf{C}_b = [(\delta\mathbf{Y})\mathbf{Y}^\top + \mathbf{Y}(\delta\mathbf{Y})^\top] \circ \mathbf{S}, \tag{3.9}$$

$$\delta\mathbf{A} = \mathbf{P}^\top\delta\mathbf{C}_b\mathbf{P} \tag{3.10}$$

By the differentiation rule of a scalar-valued matrix function, we know that

$$\delta J = \left\langle \text{vec}\left(\frac{\partial J_3}{\partial \mathbf{A}}\right), \text{vec}(\delta\mathbf{A}) \right\rangle = \text{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{A}}\right)^\top \delta\mathbf{A}\right) \tag{3.11}$$

where vec$(\cdot)$ denotes the vectorization of a matrix and $\langle\cdot,\cdot\rangle$ denotes the inner product. Combining the result with $\delta\mathbf{A} = \mathbf{P}^\top\delta\mathbf{C}_b\mathbf{P}$ in Equation (3.10), we can obtain

$$\begin{aligned}
\delta J &= \operatorname{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{A}}\right)^\top \delta\mathbf{A}\right) \\
&= \operatorname{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{A}}\right)^\top \mathbf{P}^\top \delta\mathbf{C}_b \mathbf{P}\right) \\
&= \operatorname{trace}\left(\left(\mathbf{P}\frac{\partial J_3}{\partial \mathbf{A}}\mathbf{P}^\top\right)^\top \delta\mathbf{C}_b\right) \\
&= \operatorname{trace}\left(\left(\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top \delta\mathbf{C}_b\right)
\end{aligned} \tag{3.12}$$

The last equality holds because from Equations (3.7) and (3.11) we know that $\delta J$ can also be equally written as $\operatorname{trace}\big((\frac{\partial J_2}{\partial \mathbf{C}_b})^\top \delta\mathbf{C}_b\big)$. Noting that Equation (3.12) is true for any $\delta\mathbf{C}_b$, we can therefore derive that

$$\frac{\partial J_2}{\partial \mathbf{C}_b} = \mathbf{P}\frac{\partial J_3}{\partial \mathbf{A}}\mathbf{P}^\top \tag{3.13}$$

Note that $\frac{\partial J_3}{\partial \mathbf{A}}$ can be computed as $\frac{\partial J_3}{\partial \mathbf{A}} = \hat{\mathbf{U}}_b\big(\mathbf{G}\circ(\hat{\mathbf{U}}_b^\top \frac{\partial J_4}{\partial \mathbf{Z}}\hat{\mathbf{U}}_b)\big)\hat{\mathbf{U}}_b^\top$, where $\hat{\mathbf{U}}_b$ and $\hat{\mathbf{D}}_b$ are the inverse permutated eigenvectors and eigenvalues, respectively, and $\mathbf{G}$ is a matrix whose $(i,j)$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 $\lambda_i$ is the $i$th diagonal element of $\hat{\mathbf{D}}_b$. Readers are referred to [6] for the proof of $\frac{\partial J_3}{\partial \mathbf{A}}$.

Again, combining $\delta J = \operatorname{trace}\big((\frac{\partial J_2}{\partial \mathbf{C}_b})^\top \delta\mathbf{C}_b\big)$ with $\delta\mathbf{C}_b = [(\delta\mathbf{Y})\mathbf{Y}^\top + \mathbf{Y}(\delta\mathbf{Y})^\top]\circ\mathbf{S}$ in Equation (3.9), it can be obtained that

$$\operatorname{trace}\left(\left(\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top \delta\mathbf{C}_b\right) = \operatorname{trace}\left(\left(\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top ([(\delta\mathbf{Y})\mathbf{Y}^\top + \mathbf{Y}(\delta\mathbf{Y})^\top]\circ\mathbf{S})\right) \tag{3.14}$$

By the identity that $\operatorname{trace}(\mathbf{A}^\top(\mathbf{B}\circ\mathbf{C})) = \operatorname{trace}((\mathbf{B}\circ\mathbf{A})^\top\mathbf{C})$, we can obtain

$$\operatorname{trace}\left(\left(\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top \delta\mathbf{C}_b\right) = \operatorname{trace}\left(\left(\mathbf{S}\circ\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top ((\delta\mathbf{Y})\mathbf{Y}^\top + \mathbf{Y}(\delta\mathbf{Y})^\top)\right) \tag{3.15}$$

Denoting $\big(\mathbf{S}\circ\frac{\partial J_2}{\partial \mathbf{C}_b}\big)$ with $\mathbf{Q}$, and applying the identity that $\operatorname{trace}(\mathbf{A}+\mathbf{B}) = \operatorname{trace}(\mathbf{A}) + \operatorname{trace}(\mathbf{B})$, $\operatorname{trace}(\mathbf{ABC}) = \operatorname{trace}(\mathbf{CAB})$ and $\operatorname{trace}(\mathbf{ABC}) = \operatorname{trace}((\mathbf{ABC})^\top)$, we can further simplify Equation (3.15) as

$$\begin{aligned}
\operatorname{trace}\left(\left(\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top \delta\mathbf{C}_b\right) &= \operatorname{trace}\left(\mathbf{Q}^\top((\delta\mathbf{Y})\mathbf{Y}^\top + \mathbf{Y}(\delta\mathbf{Y})^\top)\right) \\
&= \operatorname{trace}\left(\left((\mathbf{Q}+\mathbf{Q}^\top)\mathbf{Y}\right)^\top \delta\mathbf{Y}\right) \\
&= \operatorname{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{Y}}\right)^\top \delta\mathbf{Y}\right)
\end{aligned} \tag{3.16}$$

Again, because we know $\delta J$ can also be equally expressed as $\mathrm{trace}\big((\frac{\partial J_1}{\partial \mathbf{Y}})^\top \delta \mathbf{Y}\big)$ and the last result is valid for any $\delta\mathbf{Y}$, it can be obtained that

$$\frac{\partial J_1}{\partial \mathbf{Y}} = (\mathbf{Q}+\mathbf{Q}^\top)\mathbf{Y} \tag{3.17}$$

By substituting the value of $\mathbf{Q}$ in Equation (3.17) and using the identitiy $(\mathbf{A}\circ\mathbf{B})^\top = \mathbf{A}^\top \circ \mathbf{B}^\top$, we then obtain

$$\begin{aligned}\frac{\partial J_1}{\partial \mathbf{Y}} &= \left(\left(\mathbf{S}\circ\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top + \left(\left(\mathbf{S}\circ\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top\right)^\top\right)\mathbf{Y}\\ &= \left(\mathbf{S}\circ\left(\frac{\partial J_2}{\partial \mathbf{C}_b} + \left(\frac{\partial J_2}{\partial \mathbf{C}_b}\right)^\top\right)\right)\mathbf{Y}\end{aligned} \tag{3.18}$$

Again, combining $\delta J = \mathrm{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{Y}}\right)^\top \delta\mathbf{Y}\right)$ with $\delta\mathbf{Y} = \mathbf{P}\delta\mathbf{Y}$ in Equation (3.8), it can be obtained that

$$\begin{aligned}\mathrm{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{Y}}\right)^\top \delta\mathbf{Y}\right) &= \mathrm{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{Y}}\right)^\top \mathbf{P}\delta\mathbf{X}\right)\\ &= \mathrm{trace}\left(\left(\mathbf{P}^\top\frac{\partial J_1}{\partial \mathbf{Y}}\right)^\top \delta\mathbf{X}\right)\\ &= \mathrm{trace}\left(\left(\frac{\partial J}{\partial \mathbf{X}}\right)^\top \delta\mathbf{X}\right)\end{aligned} \tag{3.19}$$

Again, because we know $\delta J$ can also be equally expressed as $\mathrm{trace}\big((\frac{\partial J}{\partial \mathbf{X}})^\top \delta \mathbf{X}\big)$ and the last result is valid for any $\delta\mathbf{X}$, it can be obtained that

$$\frac{\partial J}{\partial \mathbf{X}} = \mathbf{P}^\top\frac{\partial J_1}{\partial \mathbf{Y}} \tag{3.20}$$

This completes the proof. As can be seen, given $\frac{\partial J}{\partial \mathbf{Z}}$, we can derive $\frac{\partial J}{\partial \mathbf{A}}$, $\frac{\partial J}{\partial \mathbf{C}_b}$, $\frac{\partial J}{\partial \mathbf{Y}}$ 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. 3.3.1, while the latter two will become known once the random grouping is performed. Now, we have all the essential elements for back-propagation. An end-to-end learning with ReDro can be readily implemented.

#### Discussion on the proposed ReDro scheme

**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 in the order of $\mathcal{O}(d^3)$.[c] Without loss of generality, assuming that $d$ can be exactly divided by the number of groups $k$, the size of each group will be $d/k$. The complexity incurred by using ReDro, which performs eigen-decomposition of the $k$ smaller-sized covariance matrices, will be $\mathcal{O}(d^3/k^2)$. Therefore, in the theoretical sense, the ReDro scheme can reduce the computation cost by up to $k^2$ times.

The implementation of ReDro only requires a random permutation of the IDs of the $d$ channels. For the gradient computation, 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 "frequency" of conducting the random permutation. Conducting it for every training sample leads to the most frequent change of the relation dropout pattern in the ReDro scheme. As will be shown, this could make the value of the objective function fluctuate violently, which will affect the convergence of network training. To show the impact of this frequency, we will carry out experiments on the random permutation at three levels, namely, epoch-level (EL), batch-level (BL), and sample-level (SL) and hold this frequency for various intervals. For example, batch-level with interval of 2 ("BL-2") uses the same random permutation pattern for two consecutive batches before refreshing. Similarly, SL-1 conducts random permutation for every training sample during the end-to-end training process.

In addition, ReDro could bring less biased eigenvalue estimates. As is known, when eigen-decomposition is applied to a large full covariance matrix, eigenvalue estimation will become 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 (with respect to the corresponding subspace though) will be estimated from each block sub-matrix on the diagonal. Because the matrices are smaller in size, eigenvalue estimate could become less biased. This property can be regarded as a byproduct of the ReDro scheme.

[c] 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 [84].

## 3.4 Experimental Results

We conduct extensive experiments on scene classification and fine-grained image classification datasets to investigate the proposed ReDro scheme. For scene classification, the *MIT Indoor* dataset [85] is used. For fine-grained image classification, the commonly used *Birds* [86], *Airplanes* [87], and *Cars* [88] 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 [3], [5], we resize all images to $448 \times 448$ during training and testing. More details on datasets, and implementation of ReDro are provided in the Appendix A and C.

Our experiments consist of two main parts. The first part in Sec. 3.4.1 describes the experiments with ReDro, including its computational advantage, the impact of key parameters, and its performance on several typical deep architectures for learning SPD representation. The second part in Sec. 3.4.2 compares the classification performance achieved by the two proposed methods with that of the state-of-the-art SPD representation based methods.

### 3.4.1 Experiments with ReDro scheme

In this section, we present experiments with ReDro scheme. We divide the experiments in three main parts. The first part shows the computational advantage brought by the proposed ReDro scheme. The second parts investigates the efficiency of ReDro versus the intensity level at which it is applied. Finally, the third part validates the performance of ReDro via multiple typical SPD representation learning methods that explicitly use matrix normalisation. Furthermore, we show an additional ablation study on the group number $k$.

**On the computational advantage of ReDro**

This part compares the computational cost between the case without ReDro and the case with ReDro. The former means that a full SPD matrix (i.e., covariance or kernel matrix) is used.

Specifically, we experiment with (i) four typical SPD representation learning methods based on covariance matrix, namely, MPN-COV [4], DeepCOV [6], Improved BCNN (IBCNN) [3], and iSQRT-COV [5] and (ii) one SPD representation learning method based on kernel matrix, namely, DeepKSPD [6]. The MPN-COV, DeepCOV, and DeepKSPD methods conduct matrix normalisation via eigen-decomposition, while IBCNN and iSQRT-COV realise matrix normalisation by the matrix square-rooting operation. These five methods represent the case without using ReDro. To compare with them, we implement ReDro within the DeepCOV and DeepKPSD methods with the ResNet-50 network as

| SPD Matrix Dimensions | No ReDro used | | | | |
|---|---|---|---|---|---|
| | Deep-COV [6] | Deep-KSPD [6] | MPN-COV [4] | IBCNN [3] | iSQRT-COV [5] |
| 128×128 | 0.004 | 0.008 | 0.004 | 0.006 | 0.001 |
| 256×256 | 0.013 | 0.016 | 0.013 | 0.014 | 0.006 |
| 512×512 | 0.031 | 0.045 | 0.031 | 0.032 | 0.030 |
| 1024×1024 | 0.097 | 0.189 | 0.097 | 0.121 | 0.097 |

| SPD Matrix Dimensions | DeepCOV [6] using ReDro | | | | DeepKSPD [6] using ReDro | | | |
|---|---|---|---|---|---|---|---|---|
| | with $k = 2$ | with $k = 4$ | with $k = 8$ | with $k = 16$ | with $k = 2$ | with $k = 4$ | with $k = 8$ | with $k = 16$ |
| 128×128 | 0.007 | 0.009 | 0.011 | 0.015 | 0.010 | 0.012 | 0.018 | 0.031 |
| 256×256 | **0.011** | **0.011** | 0.013 | 0.020 | **0.015** | 0.016 | 0.022 | 0.037 |
| 512×512 | **0.030** | **0.022** | **0.023** | **0.030** | **0.031** | **0.029** | **0.035** | 0.054 |
| 1024×1024 | **0.090** | **0.076** | **0.056** | **0.062** | **0.087** | **0.084** | **0.082** | **0.094** |

**Table 3.1:** Comparison of computational time (in second) for SPD matrix estimation and matrix normalisation by using or not using the proposed ReDro scheme. The reported time is the sum of one forward and one backward propagations (individual propagation time is given in the Sec. A.2 of Appendix A). The four methods to the left represent the case not using ReDro. The case using ReDro is implemented upon the DeepCOV [6] and DeepKSPD [6] methods with various $k$. The boldface numbers shows that ReDro saves computational time compared with the cases on the left.

backbone. To this end, we only modify their COV and KSPD layers with the proposed ReDro scheme, leaving all the other settings unchanged.

In the SPD matrix layer of these methods, computation and matrix normalisation operations are often tightly coupled. It is difficult to exclude the normalisation step so as to exactly compare with ReDro for computational cost. To be fair, the total time taken by both steps of ReDro and matrix normalisation is used for comparison. To test the computational cost over the covariance or kernel matrix of various sizes, we follow the literature by applying an additional $1 \times 1$ convolutional layer to vary 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 the MatConvNet library [89] on MATLAB 2019a.

Table 3.1 shows the timing results. Firstly, along the size of the SPD matrix, we can observe that (i) when the size is relatively small (i.e., 256×256), ReDro is unnecessary. This is because the eigen-decomposition does not incur significant computation cost. Using ReDro in this case complicates the procedure and adds computational overhead; (ii) however, when the matrix size increases to 512×512, the advantage of using ReDro emerges, and it becomes more pronounced with a matrix size of 1024×1024. In these cases, ReDro is computationally more efficient by 20% up to 50% than the counterparts without ReDro (see the results in bold). In addition, in terms of the total training time, ReDro can significantly shorten the time. For example, on the Birds dataset, ReDro reduces the training time of the DeepCOV method from 52.4 hours down to 24.3 hours, with 1.3 percentage point improvement on classification accuracy, as will be detailed in Figure 3.2.

Secondly, we test different values for the group number $k$ in ReDro. As can be seen, ReDro shows higher computational efficiency with $k = 4$ or 8 for both DeepCOV and DeepKSPD. Specifically, the computational advantage of ReDro on DeepCOV mainly comes from eigen-decomposition of small-sized covariance matrices, while on DeepKSPD the advantage comes from both smaller-sized kernel matrix computation and eigen-decomposition. The computation of kernel matrix involves more arithmetic operations than that of covariance matrix, hence, it takes more computation time. ReDro significantly reduces the computation time by replacing a full kernel matrix with a block-diagonal one. For $k = 2$, we can expect from the earlier complexity analysis that its efficiency shall be lower. For $k = 16$, the overhead for processing many small matrices becomes non-trivial, not to mention the loss of more information. Since $k = 4$ gives rise to a high computational efficiency (and overall good classification results, as will be shown later), the following experiments will focus on this setting, with more discussion on $k$ left in Sec. 3.4.1.

### On the efficiency of ReDro *vs.* the frequency of redrawing the channel groups

This experiment investigates the impact of the frequency of redrawing the channel-wise grouping for ReDro on the classification performance. Based on the findings in the previous experiment, we focus on 512×512 and 1024×1024-sized SPD matrices in this experiment. Again, we implement ReDro with DeepCOV and DeepKSPD [6]. We begin our investigation with DeepCOV by applying ReDro at 15 different redrawing frequency levels, i.e., epoch-level (EL)-$\{100, 50, 20, 10, 5, 2, 1\}$, batch-level (BL)-$\{20, 10, 5, 2, 1\}$, and sample-level (SL)-$\{6, 3, 1\}$, whose meaning is explained in Sec. 3.3.1. Their classification accuracy is compared with those of the original DeepCOV [6].

The left part of Table 3.2 shows the results when the covariance matrix size is 512×512. Firstly, along the frequency level, we can observe that, (i) the efficiency of ReDro indeed varies with the frequency of redrawing channel-wise grouping; (ii) multiple frequency levels lead to improved classification over the baseline “No ReDro.” In particular, EL-1 and EL-2 work better than others across all datasets; (iii) also, EL-1 works well with the MIT and Cars datasets, and SL-6 works well with the Airplane and Bird datasets though. This suggests that for some datasets, a less intensive change of random permutation, i.e., at epoch level (EL), is preferred when applying ReDro. We observe that in this case, using more intensive changes, i.e., at the sample level (SL), could cause the objective function to fluctuate violently and affect the convergence.

Secondly, we observe that DeepCOV trained with ReDro improves classification over its baseline on all datasets, with the improvement of 0.5 ~ 1.3%. For example, on the Birds dataset with the frequency level of SL-6, ReDro improves the accuracy from the baseline 85.4% to 86.7%. The improvement could be attributed to two factors: (i) The dropout-like regularisation effect in ReDro helps the network learn better features, and (ii) ReDro estimates smaller covariance matrices, instead of a full $d \times d$ one, from the local descriptors in the convolutional feature map. This helps mitigate the bias issue in the estimation of the eigenvalues of the covariance matrix.

Beside of improving classification, ReDro at various frequency 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 the network to converge faster. Figure 3.2 shows that DeepCOV with ReDro achieves lower classification errors with a smaller number of epochs than the baseline “No ReDro” on the Birds dataset. 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 frequency 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 observations are also made on other datasets.

The middle part of Table 3.2 shows the result when the covariance matrix size is increased to 1024×1024. Note that few existing networks have ever tried such a large matrix. We use the 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”. We would like to highlight that the DeepCOV-ResNet used in this experiment is an improved variant of the one reported in our ECCV paper [90]. The improvements are as follows: (i) the 1×1 convolution operation now has a bias term, (ii) batch normalisation and ReLU activation are applied after the 1×1 convolution operation, and (iii) the network is trained with the Adam optimiser instead of SGD. The improved DeepCOV-ResNet performs better across all datasets as can be seen from the baseline results. Due to the longer training time caused by the larger covariance matrix, we sample eight out of the 15 frequency levels in Table 3.2 and test them. From the results, we can observe that, (i) DeepCOV-ResNet with ReDro achieves comparable results with the baseline consistently across all datasets; (ii) The difference between the results of DeepCOV-ResNet with ReDro and DeepCOV-ResNet without ReDro varies in the range of 0.1 ~ 1% across different frequency levels. Though there is a slight degradation in performance, ReDro helps reduce the training time of DeepCOV-ResNet by up to 50%. This performance degradation can be partially reduced by switching to a smaller $k$ value. The results shown in Table 3.2 is with $k = 4$, i.e., four blocks in the block-diagonal covariance matrix. A higher $k$ value may remove some important feature relationships from the covariance matrix and cause the degradation of classification performance. We

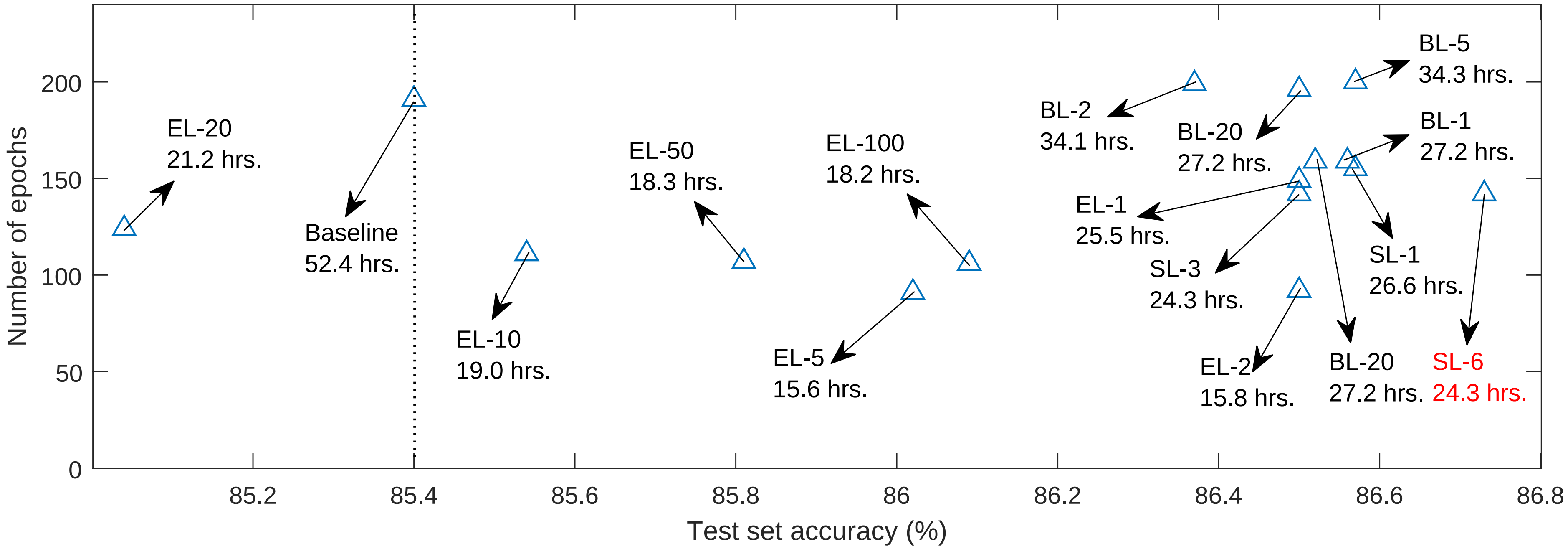

**Figure 3.2:** Comparison of the classification error, network training time, and the total number of epochs obtained by using ReDro at various frequency levels. Each small triangle in this figure represents a ReDro case. The arrow points to its frequency level and the network training time taken to achieve its highest test accuracy, as reflected by the x-axis. The result in red indicates the best performer. The Birds dataset is used.

| Methods → | | DeepCOV [6] (512×512) | | | | DeepCOV-ResNet (1024×1024) | | | | | DeepKSPD [6] (512×512) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Datasets → | | MIT | Airplane | Cars | Birds | MIT | Airplane | Cars | Birds | | MIT | Airplane | Cars | Birds |
| No ReDro | | 79.2 | 88.7 | 91.7 | 85.4 | 84.6* | 87.7* | 91.6* | 86.7* | | 82.5* | 90.0 | 91.6 | 84.8* |
| ReDro with $k = 4$ | EL-1 | **80.5*** | **89.1** | **92.2*** | **86.5** | 83.7 | 87.0 | 91.4 | 86.2 | ReDro with $k = 2$ | 79.5 | 89.8 | **92.3** | 84.5 |
| | EL-2 | **80.5*** | **88.9** | **92.1** | **86.5** | – | – | – | – | | – | – | – | – |
| | EL-5 | **80.1** | **89.0** | 91.7 | **86.0** | 83.6 | 87.0 | 91.2 | 86.0 | | 79.3 | **90.1*** | **92.5** | 84.8* |
| | EL-10 | **80.3** | 88.6 | 88.9 | **85.5** | – | – | – | – | | – | – | – | – |
| | EL-20 | **79.9** | 88.6 | 91.3 | 85.0 | 84.5 | 87.1 | 91.4 | 86.1 | | 80.7 | 90.0 | **92.6*** | 84.6 |
| | 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** | 83.7 | 87.3 | 91.1 | 86.0 | | 78.9 | 89.7 | **92.5** | 84.5 |
| | BL-2 | 76.9 | **89.1** | 88.0 | **86.4** | – | – | – | – | | – | – | – | – |
| | BL-5 | 76.6 | **89.1** | 87.8 | **86.6** | 83.8 | 87.0 | 91.0 | 86.2 | | 78.7 | 89.9 | **92.4** | 84.4 |
| | BL-10 | 76.8 | **89.0** | 88.0 | **86.5** | – | – | – | – | | – | – | – | – |
| | BL-20 | 78.6 | 88.7 | 87.5 | **86.5** | 83.8 | 86.9 | 91.1 | 86.2 | | 78.7 | 89.8 | **92.6*** | 84.4 |
| | SL-1 | 77.7 | **88.9** | 90.0 | **86.6** | 83.7 | 86.5 | 90.1 | 84.9 | | 79.0 | 90.0 | **92.4** | 84.4 |
| | SL-3 | 78.1 | **89.0** | 89.8 | **86.5** | – | – | – | – | | – | – | – | – |
| | SL-6 | 79.1 | **89.2*** | 91.1 | **86.7*** | 84.3 | 86.5 | 90.8 | 85.7 | | 78.7 | 89.8 | **92.4** | 84.5 |

**Table 3.2:** Results using ReDro with $k$ = 2/4 at different frequency levels on DeepCOV [6] (left), DeepCOV-ResNet (middle), and DeepKSPD [6] (right). The results higher than the baseline are shown in bold. The highest results for each dataset are marked by asterisks.

perform further experiments to investigate the impact of $k$ values on DeepCOV-ResNet in Sec. 3.4.1. Nevertheless, using a smaller value of $k$ in ReDro does not save much training time as compared to a larger value. A thorough investigation on a smaller value, i.e., $k = 2$, is presented in the right part of Table 3.2 and described below.

The right part of Table 3.2 shows the result of ReDro applied to a kernel matrix via the DeepKSPD network. This network produces a 512×512-dimensional kernel matrix. Again, considering the longer training time in this case, we focus on testing the frequency levels used in the DeepCOV-ResNet experiments but use the setting of $k = 2$ instead. The obtained classification accuracy is compared with the reproduced results using the source code supplied by the original work [6]. From the results, we can observe that, (i) for all frequency levels, DeepKSPD with ReDro outperforms the baseline results on the Cars dataset (the best performance is achieved by both EL-20 and BL-20); (ii) on the Airplane dataset, one frequency level (i.e., EL-5) outperforms the baseline and the rest achieve equal or comparable results with respect to the baseline; (iii) on the Birds dataset, only one frequency level (i.e., EL-5) achieves equal result as the baseline and the rest achieve comparable results to baseline; iv) on the MIT dataset, all frequency levels do not perform as well as the baseline. This degradation could be due to the removal of pairwise relationships between visual descriptors from the kernel matrix by ReDro. We also perform experiments with $k = 4$ on some frequency levels across all datasets. We found that a larger value of $k$ could further deteriorate the ReDro performance.

### On the performance of ReDro with typical methods

The above experiments investigates the performance of ReDro by integrating it into DeepCOV and DeepKSPD. Below, we further integrate ReDro into other typical SPD representation methods that use matrix normalisation as an important step, namely, Improved BCNN [3], MPN-COV [4], and iSQRT-COV [5]. Since iSQRT-COV uses the matrix square-rooting normalisation without involving eigen-decomposition, we investigate the regularisation effect of ReDro for it. This also applies to BCNN [1] which does not have a 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 3.3 shows the results using ReDro in typical SPD visual representation methods. In total, six scenarios are implemented with these methods, using differently sized covariance matrices. For each scenario, ReDro is used at the same frequency level. We compare the baseline "No ReDro" with the cases in which ReDro is integrated. As can be seen, (i) except iSQRT-COV, networks trained with ReDro generally outperform their baseline counterparts. ReDro-based iSQRT-COV is comparable with the baseline; (ii) except for a few cases, the improvement is consistent across all datasets. Overall, improvements higher than 1% can be commonly seen, with the maximum one being 2% (i.e., IBCNN with ReDro ($k = 2$) on the Cars dataset); (iii) for the results from ReDro with $k = 2$ and

| Dataset | Training mode | DeepCOV-ResNet (1024×1024) | DeepKSPD (512×512) | DeepCOV (512×512) |
|---|---|---|---|---|
| Airplane | No ReDro | 87.7* | 90.0* | 88.7 |
| | ReDro ($k = 2$) | 86.8 | 89.2 | **89.3*** |
| | ReDro ($k = 4$) | 86.5 | 89.1 | **89.2** |
| Cars | No ReDro | 91.6 | 91.6 | 91.7 |
| | ReDro ($k = 2$) | **92.0*** | **92.6*** | 90.8 |
| | ReDro ($k = 4$) | 91.1 | **91.7** | **92.2*** |
| Birds | No ReDro | 86.7 | 84.8 | 85.4 |
| | ReDro ($k = 2$) | **86.9*** | 84.8 | **86.5** |
| | ReDro ($k = 4$) | 86.2 | 83.0 | **86.7*** |

| Dataset | Training mode | IBCNN (512×512) | BCNN (512×512) | MPN-COV (256×256) | iSQRT-COV (256×256) |
|---|---|---|---|---|---|
| Airplane | No ReDro | 87.0 | 85.3 | 86.1 | 91.1 |
| | ReDro ($k = 2$) | **88.8*** | **86.6*** | **87.4** | 90.6 |
| | ReDro ($k = 4$) | **88.6** | **86.6*** | **88.2*** | 91.1 |
| Cars | No ReDro | 90.6 | 89.1 | 89.8 | 92.6 |
| | ReDro ($k = 2$) | **92.6*** | **90.9*** | **91.4** | 92.3 |
| | ReDro ($k = 4$) | **91.2** | **90.5** | **91.7*** | 92.6 |
| Birds | No ReDro | 85.4 | 84.1 | 82.9 | 88.5 |
| | ReDro ($k = 2$) | **85.5*** | **84.6*** | **83.5*** | 88.0 |
| | ReDro ($k = 4$) | 84.6 | 83.9 | **83.2** | **88.6*** |

**Table 3.3:** 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 [3], BCNN [1], and iSQRT-COV [5] networks are trained (including the baseline) with the settings in their original papers. As for MPN-COV [4], it is trained with the same pretrained network as IBCNN, BCNN, DeepCOV, and DeepKSPD for consistency. Note that to ensure a fair of comparison, we report the classification result obtained by softmax predictions as usual, and do not utilise the additional step of training a separate SVM classifier. The highest results on each method are marked by asterisks.

4, they are 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 an overall better option.

Below, we provide some explanations regarding the performance of iSQRT-COV. All the methods except iSQRT-COV use a backbone model that is pretrained *without* incorporating the SPD representation layers. That is, these layers are only incorporated later and just fine-tuned with a fine-grained image dataset. On the other hand, the backbone model in iSQRT-COV has incorporated the SPD representation layers when it is pretrained, and these layers are also further fine-tuned with the fine-grained image dataset. This helps iSQRT-COV achieve better performance. Meanwhile, as noted previously, iSQRT-COV [5] utilises a special matrix normalisation without eigen-decomposition. Our ReDro could provide a general power normalisation by the eigen-decomposition based matrix

| Datasets → | | MIT | | | Airplane | | |
|---|---|---|---|---|---|---|---|
| Methods → | | DeepCOV-ResNet | | DeepCOV | DeepCOV-ResNet | | DeepCOV |
| | | *ver. 1024* | *ver. 512* | | *ver. 1024* | *ver. 512* | |
| No ReDro | | 84.6 | 84.6 | 79.2 | 87.7 | 84.9 | 88.7 |
| Value of $k$ for ReDro | 2 | 84.0 | 84.3 | **81.2** | 87.0 | **86.2** | **89.3** |
| | 3 | 84.3 | 84.0 | **80.3** | 86.4 | **86.1** | **89.2** |
| | 4 | 84.3 | 83.0 | **80.5** | 87.3 | **86.2** | **89.2** |
| | 8 | 82.7 | 81.4 | 78.6 | 86.3 | **85.6** | 88.1 |
| | 16 | 79.7 | 78.5 | 73.6 | 85.8 | **85.0** | 87.3 |

| Datasets → | | Cars | | | Birds | | |
|---|---|---|---|---|---|---|---|
| Methods → | | DeepCOV-ResNet | | DeepCOV | DeepCOV-ResNet | | DeepCOV |
| | | *ver. 1024* | *ver. 512* | | *ver. 1024* | *ver. 512* | |
| No ReDro | | 91.6 | 89.9 | 91.7 | 86.7 | 86.1 | 85.4 |
| Value of $k$ for ReDro | 2 | 91.3 | **91.3** | 90.8 | **86.8** | **86.3** | **86.5** |
| | 3 | 91.2 | **90.6** | **92.3** | 86.3 | 86.1 | **85.7** |
| | 4 | 91.4 | **90.2** | **92.2** | 86.2 | 85.6 | **86.7** |
| | 8 | 89.8 | 87.9 | 91.0 | 85.4 | 84.2 | 83.4 |
| | 16 | 87.1 | 84.1 | 87.2 | 84.0 | 81.9 | 81.0 |

**Table 3.4:** Impact of the group number $k$ in ReDro with DeepCOV [6] and DeepCOV-ResNet. The results higher than without the ReDro are shown in bold.

normalisation and even access a wider range normalisation functions.

### On the performance of ReDro *vs.* group number $k$

To gain more understanding about the group number $k$ in ReDro, we implement ReDro using different values of $k$ with DeepCOV-ResNet and DeepCOV. Two versions of DeepCOV-ResNet are used for the experiments. The first version learns a 1024×1024 covariance representation which has been used in the previous experiments. The second version learns a 512×512 covariance representation by reducing the number of feature channels of ResNet-50 from 2048 to 512 with the 1×1 convolution. We denote the first and second version as DeepCOV-ResNet *version 1024* and DeepCOV-ResNet *version 512*, respectively. Table 3.4 shows that, (i) for different methods, in most of the cases, $k = \{2, 3, 4\}$ yields overall better performance than the baseline. These settings of $k$ take less computational time during training than the baseline; (ii) for different matrix sizes, overall the setting of $k = 2$ performs better. Though there is less computational benefit with $k = 2$, it is still faster than the baseline; (iii) larger values of $k$, i.e., 8 and 16, produce inferior results because they cause a significant amount of information loss from the original full covariance matrix. Furthermore, if the matrix size is not large enough (i.e., 256×256), the computational benefit with a larger $k$ value may be limited. Overall, our ReDro with a setting of smaller $k$ value is able to achieve comparable performance with the baseline at a reduced computational cost.

| Methods | Backbone | Airplane | Cars | Birds |
|---|---|---|---|---|
| VGG-16 [18] | VGG-16 | 76.6 | 79.8 | 70.4 |
| NetVLAD [91] | | 81.8 | 88.6 | 81.6 |
| NetFV [92] | | 79.0 | 86.2 | 79.9 |
| BCNN [1] | | 83.9 | 90.6 | 84.0 |
| CBP [2] | | 84.1 | 91.2 | 84.3 |
| LRBP [28] | | 87.3 | 90.9 | 84.2 |
| KP [25] | | 86.9 | 92.4 | 86.2 |
| HIHCA [93] | | 88.3 | 91.7 | 85.3 |
| Improved BCNN [3] | | 88.5 | 92.0 | 85.8 |
| SMSO [79] | | – | – | 85.0 |
| MPN-COV [29] | | 89.9 | 92.2 | 86.7 |
| G$^2$DeNet [29] | | 89.0 | **92.5** | **87.1** |
| iSQRT-COV [5] (reproduced)$^\dagger$ | | 88.5 | 86.4 | 78.5 |
| DeepCOV [6] | | 88.7 | 91.7 | 85.4 |
| DeepKSPD [6] | | 90.0 | 91.6 | 84.8 |
| DeepCOV+ ReDro (ours) | | 89.2 | 92.2 | 86.7 |
| CBP [2] | ResNet-50 | 81.6 | 88.6 | 81.6 |
| KP [25] | | 85.7 | 91.1 | 84.7 |
| SMSO [79] | | – | – | 85.8 |
| iSQRT-COV [5] | | 89.5 | 91.7 | **87.3** |
| DeepCOV-ResNet | | 87.7 | 91.6 | 86.7 |
| DeepCOV-ResNet+ ReDro (ours) | | 86.8 | **92.0** | 86.9 |

**Table 3.5:** Comparison between the proposed methods and other SPD representation methods in terms of classification accuracy (%). The performance of existing SPD representation methods are quoted from the original papers. The best methods across datasets and CNN backbones are marked in bold. $^\dagger$With the same hyper-parameter settings, CNN backbone and training strategy as [6], and default iSQRT-COV iterations of 5 times.

### 3.4.2 Comparison with Other Methods

In this section, we compare our proposed methods with the existing SPD representation methods. Specifically, we compare with fifteen state-of-the-art methods, namely, VGG-16 [18], NetVLAD [91], NetFV [92], BCNN [1], CBP [2], LRBP [28], KP [25], HIHCA [93], Improved BCNN [3], SMSO [79], MPN-COV [29], G$^2$DeNet [29], iSQRT-COV [5], DeepCOV [6], and DeepKSPD [6]. For fairness, we divide them into two groups based on which CNN backbone they have used, i.e., VGG-16 or ResNet-50.

The comparison is shown in Table 3.5. For ReDro based DeepCOV methods, we report the best results from Table 3.2. The top part of Table 3.5 shows the results of the methods that uses the VGG-16 network as the backbone. As shown in the table, (i) our ReDro based DeepCOV method (indicated as “DeepCOV+ReDro” in the table) achieves comparable results with the existing methods on all three datasets; (ii) Our DeepCOV+ReDro performs better than many existing methods. In terms efficiency, our DeepCOV+ReDro is significantly faster than the exiting methods involving matrix normalisation of larger SPD matrices, i.e., Improved BCNN, MPN-COV, G$^2$DeNet, iSQRT-COV, DeepCOV, and DeepKSPD. The better result of G$^2$DeNet [29] on the Cars and Birds datasets than our proposed methods could be due to the fact that its backbone network is pre-trained with the second-order representation while the backbone networks pre-trained in our methods are just the commonly used ones which do not involve any second-order representation. Using a backbone network pre-trained in the way of G$^2$DeNet may further improve the performance of our proposed methods.

The bottom part of Table 3.5 shows the results of the methods that use the ResNet-50 network as the backbone. As can be seen, (i) our DeepCOV-ResNet+ReDro (indicated as "DeepCOV-ResNet+ReDro" in the table) method outperforms other methods on the Cars dataset and performs comparably to state-of-the-art methods on the other two datsets. Note that, DeepCOV-ResNet+ReDro is significantly faster than the existing methods; (ii) On the Birds dataset, iSQRT-COV attains better results than the proposed methods. This is again due to the use of a backbone network that has been pre-trained with second-order representation in the method. However, our DeepCOV-ResNet+ReDro is faster than iSQRT-COV and will enjoy the performance improvement if the same the pre-trained backbone model is used.

## 3.5 Conclusion

In this chapter, we propose a method for efficient eigen-based normalisation in SPD visual representations, i.e., covariance and kernel matrix representations. The proposed method, ReDro, exploits the efficiency of computing the eigen-decomposition of the block-diagonal matrix to shorten the end-to-end SPD matrix normalisation time. In our conducted experiments, we demonstrated the benefits of ReDro with SPD matrix sizes up to $1024 \times 1024$ dimensions. In those experiments, we show how ReDro can effectively reduce the matrix normalisation time of large SPD matrices during network training. By principle, ReDro can also be applied to SPD matrix size larger than $1024 \times 1024$ for reducing the matrix normalisation cost during training time.

We also show that besides improving the matrix normalisation time, ReDro also improves the network performance and significantly reduces the network convergence time. The performance improvement and reduced convergence time may be attributed to the following factors: (1) The small matrices it estimates are from a lesser number of channels than a full matrix. Therefore, the estimated eigenvalues in those small matrices become less biased which leads to better SPD matrix estimate. (2) The dropout-like regularisation of ReDro encourages the network to learn better representation.

In the future, we plan to apply ReDro to perform matrix normalisation of much larger SPD matrices. In addition, we would like to apply ReDro to other applications besides image classification such as object detection, semantic segmentation and object tracking.

# Chapter 4

# FastCOV for Efficiently Learning Large-sized SPD Visual Representation

*The work of this chapter is under the review of IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).*

Symmetric positive definite (SPD) visual representations are effective and popular due to their ability of capturing high-order statistics describing images. When computed over features of deep networks, eigen-decomposition of multiple SPD matrices is typically required to perform a key step, called matrix normalisation. This results in a significant computational cost, especially when the number of channels used to compute matrices is large. In this chapter, we propose a novel scheme, called FastCOV, which exploits the intrinsic connection between eigensytems of $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$ to address this situation. Specifically, it computes position-wise covariance matrix upon convolutional feature maps instead of the typical channel-wise covariance matrix. As the spatial size of feature maps is usually much smaller than the channel size, conducting eigen-decomposition of the position-wise covariance matrix avoids rank-deficiency, while being faster than the decomposition of the channel-wise covariance matrix. Finally, the eigenvalues and eigenvectors of the normalised channel-wise covariance matrix are retrieved by the connection of the $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$ eigen-systems. As experimentally demonstrated, the proposed scheme reduce the computational cost of learning large SPD visual representations with advanced CNNs that have higher number of feature channels but low resolution feature maps and improve the recognition performance.

## 4.1 Introduction

In computer vision, learning quality visual representation is challenging. In recent years, representing images with local descriptors and pooling each into a global representation has been proven effective. Among the pooling methods, covariance matrix based pooling, which exploits the second-order information of local descriptors, has gained substantial popularity. Since covariance matrix is symmetric positive definite (SPD), such a representation is often called an SPD visual representation. It has been successfully used in many vision tasks, including generic image classification [4], [65], image set classification [66], image segmentation [11], activity recognition [67], [68], few-shot learning [47], [69], and fine-grained image classification [1]. Inspired by the success of deep learning, several pieces of pioneering work have incorporated SPD visual representation into convolutional neural networks (CNNs) and performed investigation on important issues such as matrix function back-propagation [11], compact matrix estimation [2], matrix normalisation [3], [70], [71], and kernel-based extension [6]. These advances improve both the effectiveness of SPD visual representations and image recognition performance.

Despite the above successes, end-to-end learning of SPD visual representations within CNN remains a computationally challenging task because (i) the number of coefficients of covariance matrix increases quadratically with the number of channels in a convolutional feature map, and (ii) eigen-decomposition is often needed to normalise the spectrum of covariance matrix for each training sample during end-to-end learning. The latter factor leads to significant computational overheads, especially considering that recent advanced deep neural networks use many channels in their final convolutional layers. Efforts have recently been made to address this situation. For example, a dimension reduction layer is used to reduce the channel number beforehand for decreasing the size of the resulting covariance matrix [4]. In addition, square rooting of a covariance matrix via so-called Newton-Schulz iterations [3], [5], and the spectral power normalisation via matrix-matrix multiplications, called MaxExp(F) [70], [71], have been shown to be effective and fast. Orthogonal to this line of research, we are interested in whether this computational challenge can be mitigated by better exploiting the properties of eigen-decomposition of an SPD matrix.

With the above motivation, we propose a method to efficiently carry out matrix normalisation for end-to-end learning of large SPD visual representation. Our method, called "fast covariance (FastCOV)", is inspired by the fact that the eigensystems of two SPD matrices $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$ are intrinsically connected [94]. To take advantage of the connection between these two eigensystems, we first compute a *position-wise* covariance matrix from a convolutional feature map instead of the *channel-wise* covariance matrix i.e., instead of the commonly used outer product over features, we apply the inner product. This eigen-decomposition of such a matrix captures the spectrum of the feature maps of deep networks very fast if the spatial size of feature maps (e.g., 7×7) is smaller than their channel number (e.g., 1024), which is very common in CNNs. Moreover, the eigen-decomposition on this much smaller position-wise covariance matrix is not only faster but it does not suffer from the rank deficiency. Finally, the matrix normalisation is performed on eigenvalues, and the eigenvalues and eigenvectors of the normalised channel-wise covariance matrix

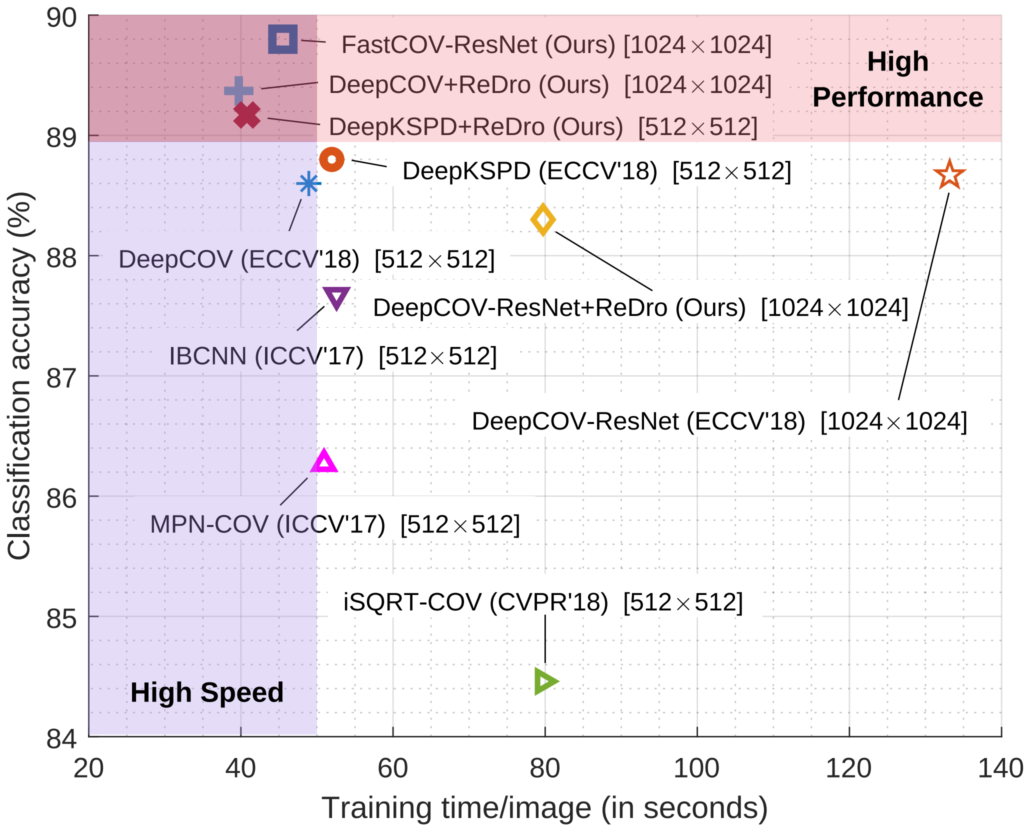

**Figure 4.1:** The classification accuracy (average over Airplane [87], Birds [86] and Car [88] datasets) *vs.* training time (per single image) of our methods and popular SPD representation methods. The size of the SPD matrix produced by different methods is displayed in square brackets. Among the methods that produce the same SPD matrix size, our methods achieve comparable or better performance than competitors while taking less time for training. More details are given in Sec. 4.4.

can be retrieved by the connection of the two eigensystems. This step is very efficient as it only requires a matrix-matrix multiplication. Finally, we derive the required matrix gradients to facilitate end-to-end training of this method via back-propagation, and discuss computational savings of FastCOV.

Similar to ReDro in the previous chapter, FastCOV facilitates efficient learning of SPD representation and provides improved image recognition performance (Fig. 4.1 illustrates this point and Sec. 4.4 provides further experimental details). The proposed scheme can be formulated as network layers to be readily inserted into the existing SPD representation methods with almost no changes in the network architecture.

Our main contributions are summarised as follows:

- To improve the efficiency of learning large SPD visual representation with matrix normalisation, FastCOV scheme is proposed. It utilises the connection between the eigensystems of the SPD matrices $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$. To the best of our knowledge, the proposed scheme is new with respect to the related literature.

- FastCOV is designed to ensure only minimal changes to the original network and negligible computational overheads. We provide the forward and backward propagation formulas for the proposed method, and discuss its properties.

Extensive experiments are conducted on one scene classification dataset and three fine-grained image classification datasets to verify the network training efficiency and image recognition performance of the proposed scheme.

## 4.2 Related Work

In this section, we provide a review of the existing methods relevant to our proposed FastCOV for SPD visual representation learning. In particular, we will review the methods that perform covariance matrix computation since it is directly related to our proposed method. We will analyse those methods from the perspective of their covariance matrix computation and matrix normalisation.

In the literature, several works have proposed methods for conducting efficient covariance representation learning. Based on their focus they can be divided into three main groups: efficient matrix normalisation, compact covariance estimation and dimension reduction. Below we present a brief review of methods in three groups.

**Efficient matrix normalisation methods** focus on developing efficient and effective normalisation methods for covariance representation estimation. The early ideas were based on element-wise normalisations [1], [70], [95]. More recent ideas are based on matrix logarithm [11] and matrix power normalisation [4], [70], [95], [96] because they usually produce better covariance representation. However, both matrix-logarithm and matrix-power normalisations typically involve the eigen-decomposition of a covariance matrix, whose worst-case computational complexity $\mathcal{O}(d^3)$ is prohibitive. As the matrix normalisation has to be applied for each training sample in both forward and backward propagation steps, the long runtime of matrix normalisation, therefore, becomes a computational bottleneck in the end-to-end learning of SPD visual representation.

Therefore, recent literature has focused on reducing the computation of matrix normalisation. For example, a special case of matrix power normalisation, called matrix square-root normalisation [3], [5], [24], approximately calculates the matrix square root by the Newton-Schulz iteration. In [3], the Newton-Schulz iteration makes the forward step computationally more efficient (only matrix-matrix multiplications are involved) but the backward propagation step still relied on solving a Lyapunov equation, which has the complexity at the same level as eigen-decomposition. In contrast, the work in [5] solved matrix square-rooting by unrolling the Newton-Schulz iteration for forward and backward propagation steps. Orthogonal to these steps, a more versatile case of matrix power normalisation, called MaxExp, performs an approximate matrix-power normalisation [70]. It has been solved by a loop of matrix-matrix multiplications [70] and a tree-structured set of matrix-matrix multiplications [71].

Although approaches in [3], [5] have computational advantage and achieve promising performance by using matrix square-rooting, their methods do not generalise to a generic matrix power normalisation. Moreover, while approaches [70], [71] can realise an approximate matrix power normalisation by mere matrix-matrix multiplications, their spectral profile cannot generalise to an unrestricted family of normalisation functions. These limitations motivate us to mitigate the computational issue of matrix normalisation from different perspectives.

**Compact covariance estimation methods** focus on developing lower dimensional covariance representation. Three pioneering methods in this line are compact bilinear pooling (CBP) [2], factorised bilinear pooling (FBP) [97] and low-rank bilinear pooling (LRBP) [28]. CBP uses random projection based Random Maclaurin [27] and Tensor Sketch [26] techniques to reduce the matrix dimension. FBP and LRBP use low-rank projections of the covariance matrix in the classifier. Two more recent works [7], [98] improved the efficiency of CBP while maintaining its effectiveness. Different from the CBP, FBP and LRBP methods, DBT-Net [73] proposes to compute compact group-wise covariance matrices. Their idea was to partition the feature channels into several groups based on their similarity and calculate a covariance matrix from each group. They show that a simple concatenation of those group-wise covariance matrices results in a lower dimensionality. Several works such as [99]–[101] also adopted channels grouping. Our FastCOV, instead of competing with these works, complements them and could be jointly used to mitigate the computational issue of matrix normalisation of these works.

**Dimension reduction methods** focus on reducing the computation cost of covariance matrix estimation from a different perspective. The primary motivation of these methods is to help the above methods to reduce their feature dimension. One of the dimension reduction techniques widely used in covariance matrix based representation literature is a linear projection of higher dimensional feature channels to lower dimensional feature channels using $1 \times 1$ convolution operation. The idea was first seen in a seminal work by Li et al. [4] and the motivation was to obtain a compact channel-wise covariance matrix from a less number of channels for efficiency since existing deep networks such as ResNet have a large number of feature channels. In a more recent work by the authors of [4] , the idea of group convolution is proposed for reducing the dimension of the covariance matrix further without much sacrifice on the performance [29]. Orthogonal to the ideas in [4], [29], other transformations are also proposed such as non-linear structured matrix transformation [76]–[79]. Our work is compatible with these techniques and can be jointly used to obtain better covariance representation.

## 4.3 The Proposed Method

In this section, we describe our fast covariance (FastCOV) method. We also provide its forward and backward propagation steps. Furthermore, we discuss its computational advantages.

### 4.3.1 Fast Covariance (FastCOV)

Let $\mathbf{X}_{d\times n}$ be a data matrix consisting of $n = wh$ feature vectors of dimension $d$. It is well known that the eigensystems of $\mathbf{X}^\top\mathbf{X}$ and $\mathbf{X}\mathbf{X}^\top$ have the following relationship [94]:

i. The two matrices share the same set of non-zero eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_r\}$, where $r$ is the rank of $\mathbf{X}$;

ii. Let $\mathbf{u}_i$ be the eigenvector of $\mathbf{X}^\top\mathbf{X}$ corresponding to eigenvalue $\lambda_i$. The unnormalised and normalised eigenvector $\mathbf{v}_i'$ and $\mathbf{v}_i$ of $\mathbf{X}\mathbf{X}^\top$ corresponding to $\lambda_i$ can be expressed as $\mathbf{v}_i' = \mathbf{X}\mathbf{u}_i$ and $\mathbf{v}_i = \frac{1}{\sqrt{\lambda_i}}\mathbf{X}\mathbf{u}_i$ respectively.

During learning the SPD visual representation, $d$ is the number of channels while $n = wh$, which is the product of the width and height of a convolutional feature map. Since typically $n \ll d$, the above relationship between eigensystems motivates us to efficiently conduct matrix normalisation as follows. For a given convolutional feature map, we first compute $\mathbf{X}^\top\mathbf{X}$ and obtain its eigenvalues and eigenvectors. Subsequently, matrix normalisation can be conducted based on these eigenvalues as they are shared between two eigensystems according to point (i). Finally, the eigenvectors for the full covariance matrix $\mathbf{X}\mathbf{X}^\top$ are obtained according to the relationship detailed in point (ii) above. The whole process incurs significantly less computation as eigen-decomposition of $\mathbf{X}^\top\mathbf{X}$ of size $n \times n$ is much faster than eigen-decomposition of $\mathbf{X}\mathbf{X}^\top$ of size $d \times d$.

An overview of the proposed FastCOV is shown in Figure 4.2. From the left end, an image is fed into a CNN backbone, and the last convolutional feature map with a size of $w \times h$ and with $d$ channels is obtained. FastCOV firstly computes a position-wise covariance matrix $\mathbf{\Sigma}_0$ with a size of $n \times n$ (where $n = wh$). The eigenvalue matrix $\mathbf{D}$ and eigenvector matrix $\mathbf{U}$ of matrix $\mathbf{\Sigma}_0$ are then obtained by applying eigen-decomposition. A normalised position-wise covariance matrix $\mathbf{\Sigma}_1$ is then created by applying matrix normalisation operation on the eigenvalues as $\mathbf{\Sigma}_1 = \mathbf{U}f(\mathbf{D})\mathbf{U}^\top$, where $f(\cdot)$ can be any valid normalisation function such as matrix-square rooting [3] or matrix logarithm [11], simply applied to diagonal elements of $\mathbf{D}$. Finally, the normalised channel-wise covariance matrix is restored by left and right multiplying the data matrix $\mathbf{X}$ to $\mathbf{\Sigma}_1$ as $\mathbf{\Sigma}_2 = \mathbf{X}\mathbf{\Sigma}_1\mathbf{X}^\top$. This completes the proposed FastCOV scheme.

To conclude, when the number of channels, $d$, in a convolutional feature map is large in comparison to $n$, computing the channel-wise covariance matrix yields a large matrix,

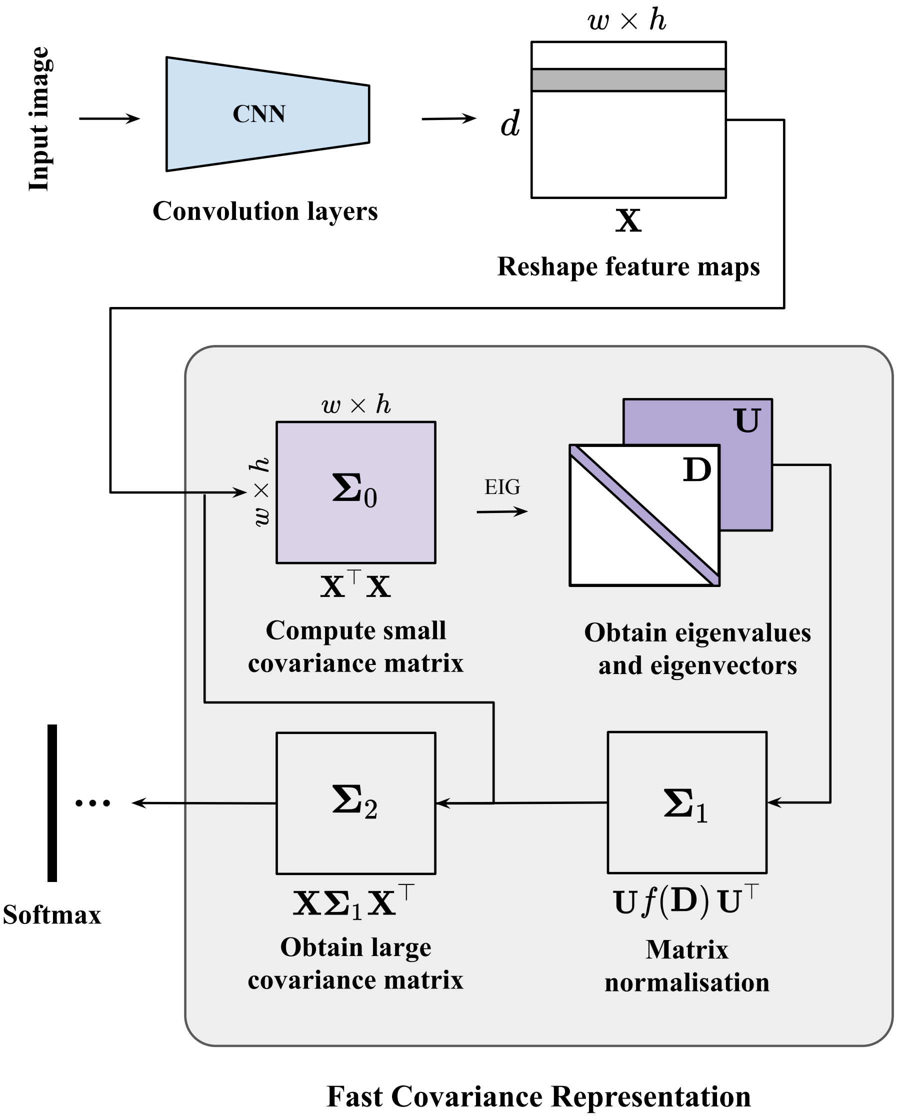
Input image
CNN
Convolution layers
$w \times h$
$d$
$\mathbf{X}$
Reshape feature maps
$w \times h$
$w \times h$
$\mathbf{\Sigma}_0$
$\mathbf{X}^\top\mathbf{X}$
Compute small covariance matrix
EIG
$\mathbf{U}$
$\mathbf{D}$
Obtain eigenvalues and eigenvectors
$\mathbf{\Sigma}_2$
$\mathbf{X}\mathbf{\Sigma}_1\mathbf{X}^\top$
Obtain large covariance matrix
$\mathbf{\Sigma}_1$
$\mathbf{U}f(\mathbf{D})\,\mathbf{U}^\top$
Matrix normalisation
...
Softmax
Fast Covariance Representation

**Figure 4.2:** Proposed FastCOV scheme for mitigating the computational issue of the matrix normalisation in learning large SPD visual representation. EIG denotes eigendecomposition.

and the matrix normalisation step is costly due to the high cost of eigen-decomposition. By computing the position-wise covariance matrix instead, FastCOV obtains a small matrix whose eigen-decomposition is very fast by comparison. To recover the normalised channel-wise covariance matrix, mere two matrix-matrix multiplications are required.

### The forward propagation in FastCOV

Recall that $\mathbf{X}_{d\times n}$, where $n = wh$, is the data matrix obtained by reshaping a convolutional feature map. The position-wise covariance matrix $\mathbf{\Sigma}_0$ is computed as

$$\mathbf{\Sigma}_0 = \mathbf{X}^\top \mathbf{X}. \tag{4.1}$$

Matrix normalisation is applied to matrix $\mathbf{\Sigma}_0$ as

$$\mathbf{\Sigma}_1 = f(\mathbf{\Sigma}_0) = \mathbf{U} f(\mathbf{D}) \mathbf{U}^\top, \tag{4.2}$$

where matrices $\mathbf{D}$ and $\mathbf{U}$ consist of the eigenvalues and eigenvectors of $\mathbf{\Sigma}_0$, respectively, and $f(\cdot)$ is a matrix normalisation function.

The normalised channel-wise covariance matrix is obtained by left and right multiplying the matrix $\mathbf{X}$ to the normalised position-wise covariance matrix $\mathbf{\Sigma}_1$ as

$$\mathbf{\Sigma}_2 = \mathbf{X}\mathbf{\Sigma}_1\mathbf{X}^\top. \tag{4.3}$$

Algorithm 2 summarises the forward steps of FastCOV.

**Algorithm 2:** The forward steps of proposed Fast Covariance representation scheme (FastCOV)

```
Input: Convolutional feature map $\mathbf{X}_{d\times n}$.
1. Compute a position-wise covariance matrix $\mathbf{\Sigma}_0 = \mathbf{X}^\top\mathbf{X}$;
2. Apply matrix normalisation operation on $\mathbf{\Sigma}_0$ and obtain normalised
   position-wise covariance matrix $\mathbf{\Sigma}_1 = \mathbf{U}f(\mathbf{D})\mathbf{U}^\top$;
3. Obtain normalised channel-wise covariance matrix $\mathbf{\Sigma}_2 = \mathbf{X}\mathbf{\Sigma}_1\mathbf{X}^\top$.
```

### The backward propagation in FastCOV

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

$$\mathbf{X} \rightarrow \underbrace{\mathbf{X}^\top\mathbf{X}}_{\mathbf{\Sigma}_0} \rightarrow \underbrace{f(\mathbf{\Sigma}_0)}_{\mathbf{\Sigma}_1} \rightarrow \underbrace{\mathbf{X}\mathbf{\Sigma}_1\mathbf{X}^\top}_{\mathbf{\Sigma}_2} \rightarrow \cdots \text{layers} \cdots \rightarrow \underbrace{J(\mathbf{X})}_{\text{Objective}}. \tag{4.4}$$

$\mathbf{\Sigma}_2$ is used by the subsequent fully connected and softmax layers. Our goal is to derive $\frac{\partial J}{\partial \mathbf{X}}$. Once obtained, all gradients before the convolutional feature map $\mathbf{X}$ can be routinely obtained.

According to Equation (4.4), $J$ is a composite function that has been applied to $\mathbf{X}$ and it can be equally expressed as a function of each of the intermediate variables as follows.

$$J(\mathbf{X}) = J_1(\mathbf{\Sigma}_0) = J_2(\mathbf{\Sigma}_1) = J_3(\mathbf{\Sigma}_2) \tag{4.5}$$

By the rules of differentiation, the following results can be obtained

$$\delta\mathbf{\Sigma}_0 = (\delta\mathbf{X})^\top\mathbf{X} + \mathbf{X}^\top(\delta\mathbf{X}) \tag{4.6}$$
$$\delta\mathbf{\Sigma}_1 = f(\delta\mathbf{\Sigma}_0) \tag{4.7}$$
$$\delta\mathbf{\Sigma}_2 = \mathbf{X}(\delta\mathbf{\Sigma}_1)\mathbf{X}^\top + \mathbf{X}\mathbf{\Sigma}_1(\delta\mathbf{X})^\top + (\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^\top \tag{4.8}$$

By the differentiation rule of a scalar-valued matrix function, we know that

$$\delta J = \left\langle \mathrm{vec}\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right), \mathrm{vec}(\delta\mathbf{\Sigma}_2)\right\rangle = \mathrm{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^\top \delta\mathbf{\Sigma}_2\right) \tag{4.9}$$

where $\mathrm{vec}(\cdot)$ denotes the vectorization of a matrix and $\langle\cdot,\cdot\rangle$ denotes the inner product. Combining the result with $\delta\mathbf{\Sigma}_2 = \mathbf{X}(\delta\mathbf{\Sigma}_1)\mathbf{X}^\top + \mathbf{X}\mathbf{\Sigma}_1(\delta\mathbf{X})^\top + (\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^\top$ in Equation (4.8), we can obtain

$$\begin{aligned}\delta J &= \mathrm{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^\top \delta\mathbf{\Sigma}_2\right)\\ &= \mathrm{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^\top (\mathbf{X}(\delta\mathbf{\Sigma}_1)\mathbf{X}^\top + \mathbf{X}\mathbf{\Sigma}_1(\delta\mathbf{X})^\top + (\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^\top)\right)\end{aligned} \tag{4.10}$$

By the identity that $\mathrm{trace}(\mathbf{A}+\mathbf{B}) = \mathrm{trace}(\mathbf{A}) + \mathrm{trace}(\mathbf{B})$, we can obtain

$$\delta J = \mathrm{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^\top \mathbf{X}(\delta\mathbf{\Sigma}_1)\mathbf{X}^\top\right) + \mathrm{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^\top (\mathbf{X}\mathbf{\Sigma}_1(\delta\mathbf{X})^\top + (\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^\top)\right) \tag{4.11}$$

Denoting $\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}$ with $\mathbf{A}$, we can further simplify Equation (4.11) as

$$\begin{aligned}
\delta J &= \operatorname{trace}\Big(\mathbf{A}^{\top}(\mathbf{X}(\delta\mathbf{\Sigma}_1)\mathbf{X}^{\top})\Big) + \operatorname{trace}\Big(\mathbf{A}^{\top}(\mathbf{X}\mathbf{\Sigma}_1(\delta\mathbf{X})^{\top} + (\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^{\top})\Big) \\
&= \operatorname{trace}\Big(\mathbf{A}^{\top}\mathbf{X}(\delta\mathbf{\Sigma}_1)\mathbf{X}^{\top}\Big) + \operatorname{trace}\Big(\mathbf{A}^{\top}\mathbf{X}\mathbf{\Sigma}_1(\delta\mathbf{X})^{\top}\Big) + \operatorname{trace}\Big(\mathbf{A}^{\top}(\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^{\top}\Big) \\
&= \operatorname{trace}\Big(\mathbf{X}^{\top}\mathbf{A}^{\top}\mathbf{X}(\delta\mathbf{\Sigma}_1)\Big) + \operatorname{trace}\Big((\delta\mathbf{X})\mathbf{\Sigma}_1\mathbf{X}^{\top}\mathbf{A}\Big) + \operatorname{trace}\Big(\mathbf{\Sigma}_1\mathbf{X}^{\top}\mathbf{A}^{\top}(\delta\mathbf{X})\Big) \\
&= \operatorname{trace}\Big(\mathbf{X}^{\top}\mathbf{A}^{\top}\mathbf{X}(\delta\mathbf{\Sigma}_1)\Big) + \operatorname{trace}\Big(\mathbf{\Sigma}_1\mathbf{X}^{\top}(\mathbf{A}+\mathbf{A}^{\top})(\delta\mathbf{X})\Big) \\
&= \operatorname{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_1}\right)^{\top}(\delta\mathbf{\Sigma}_1)\right) + \operatorname{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{X}}\right)^{\top}(\delta\mathbf{X})\right) \qquad (4.12)
\end{aligned}$$

The last equality holds because from Equations (4.5) and (4.9) we know that $\delta J$ can also be written as $\operatorname{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_1}\right)^{\top}(\delta\mathbf{\Sigma}_1)\right) + \operatorname{trace}\left(\left(\frac{\partial J_3}{\partial \mathbf{X}}\right)_2^{\top}(\delta\mathbf{X})\right)$. Substituting the value of $\mathbf{A}$, from Equation (4.12) we can therefore derive that

$$\frac{\partial J_3}{\partial \mathbf{\Sigma}_1} = (\mathbf{X}^{\top}\mathbf{A}^{\top}\mathbf{X})^{\top} = \mathbf{X}^{\top}\mathbf{A}\mathbf{X} = \mathbf{X}^{\top}\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\mathbf{X} \qquad (4.13)$$

$$\frac{\partial J_3}{\partial \mathbf{X}} = (\mathbf{\Sigma}_1\mathbf{X}^{\top}(\mathbf{A}+\mathbf{A}^{\top}))^{\top} = (\mathbf{A}^{\top}+\mathbf{A})\mathbf{X}\mathbf{\Sigma}_1 = \left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^{\top} + \left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)\right)\mathbf{X}\mathbf{\Sigma}_1 \qquad (4.14)$$

Note that $\frac{\partial J_2}{\partial \mathbf{\Sigma}_0}$ can be computed as $\frac{\partial J_2}{\partial \mathbf{\Sigma}_0} = \mathbf{U}\big(\mathbf{G}\circ(\mathbf{U}^{\top}\frac{\partial J_3}{\partial \mathbf{\Sigma}_1}\mathbf{U})\big)\mathbf{U}^{\top}$, where $\mathbf{U}$ and $\mathbf{D}$ are the eigenvectors and eigenvalues, respectively, and $\mathbf{G}$ is a matrix whose $(i,j)$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 $\lambda_i$ is the $i$th diagonal element of $\mathbf{D}$. Readers are referred to [6] and [11] for the proof of $\frac{\partial J_2}{\partial \mathbf{\Sigma}_0}$.

Again, combing $\delta J = \operatorname{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^{\top}\delta\mathbf{\Sigma}_0\right)$ with $\delta\mathbf{\Sigma}_0 = (\delta\mathbf{X})^{\top}\mathbf{X}+\mathbf{X}^{\top}(\delta\mathbf{X})$ in Equation (4.6), it can be obtained that

$$\begin{aligned}
\delta J &= \operatorname{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^{\top}\delta\mathbf{\Sigma}_0\right) \\
&= \operatorname{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^{\top}((\delta\mathbf{X})^{\top}\mathbf{X}+\mathbf{X}^{\top}(\delta\mathbf{X}))\right) \\
&= \operatorname{trace}\left(\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)\mathbf{X}^{\top} + \left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^{\top}\mathbf{X}^{\top}\right)(\delta\mathbf{X})\right) \\
&= \operatorname{trace}\left(\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right) + \left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^{\top}\right)\mathbf{X}^{\top}(\delta\mathbf{X})\right) \\
&= \operatorname{trace}\left(\left(\frac{\partial J_1}{\partial \mathbf{X}}\right)^{\top}(\delta\mathbf{X})\right) \qquad (4.15)
\end{aligned}$$

The last equality holds because from equations (4.5) and (4.15) we know that $\delta J$ can also be equally written as trace$\left(\left(\frac{\partial J_1}{\partial \mathbf{X}}\right)^\top (\delta \mathbf{X})\right)$. Noting that Equation (4.15) is true for any $\delta\mathbf{X}$, we can obtain that

$$\begin{aligned}\frac{\partial J_1}{\partial \mathbf{X}} &= \left(\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right) + \left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^\top\right)\mathbf{X}^\top\right)^\top \\ &= \mathbf{X}\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^\top + \left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)\right) \end{aligned} \tag{4.16}$$

However, since $\mathbf{\Sigma}_2$ involves matrix $\mathbf{X}$ and $\mathbf{\Sigma}_1$, i.e., $\mathbf{\Sigma}_2 = \mathbf{X}\mathbf{\Sigma}_2\mathbf{X}^\top$, $\frac{\partial J}{\partial \mathbf{X}}$ is comprises of $\frac{\partial J_3}{\partial \mathbf{X}}$ from Equation (4.12) and $\frac{\partial J_1}{\partial \mathbf{X}}$ from Equation (4.15). Combining $\frac{\partial J_3}{\partial \mathbf{X}}$ and $\frac{\partial J_1}{\partial \mathbf{X}}$, it can be obtained that

$$\begin{aligned}\frac{\partial J_1}{\partial \mathbf{X}} &= \frac{\partial J_1}{\partial \mathbf{X}} + \frac{\partial J_3}{\partial \mathbf{X}} \\ &= \mathbf{X}\left(\left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)^\top + \left(\frac{\partial J_1}{\partial \mathbf{\Sigma}_0}\right)\right) + \left(\left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)^\top + \left(\frac{\partial J_3}{\partial \mathbf{\Sigma}_2}\right)\right)\mathbf{X}\mathbf{\Sigma}_1 \end{aligned} \tag{4.17}$$

This completes the proof. In this way, when $\frac{\partial J}{\partial \mathbf{\Sigma}_2}$ is computed, we can obtain $\frac{\partial J}{\partial \mathbf{\Sigma}_1}$, $\frac{\partial J}{\partial \mathbf{\Sigma}_0}$, and then $\frac{\partial J}{\partial \mathbf{X}}$. Only $\mathbf{U}$ and $\mathbf{D}$ are needed in the whole process. They can be efficiently obtained via the proposed FastCOV scheme described in Sec. 4.3.1. An end-to-end learning can be readily implemented with this result of back-propagation.

### Discussion on computational savings by FastCOV

The computational complexity of eigen-decomposition of a $d \times d$ matrix is generally in the order of $\mathcal{O}(d^3)$. Since FastCOV performs eigen-decomposition on a position-wise covariance matrix with a size of $n \times n$, where $n = wh$, the computational complexity becomes $\mathcal{O}((wh)^3)$. Given that in recent CNNs such as ResNet [12], $wh$ is much smaller than $d$, that is $wh \ll d$, the computational cost of performing eigen-decomposition on a position-wise covariance matrix is substantially lower than that performed on a channel-wise covariance matrix with a size of $d \times d$. Thus, the improvement is in the order of $(\frac{d}{wh})^3$ times.

The implementation of FastCOV involves one extra step with two matrix-matrix multiplications (i.e., $\mathbf{X}\mathbf{\Sigma}_1\mathbf{X}^\top$) to compute the normalised channel-wise matrix $\mathbf{\Sigma}_2$. However, the step is insignificant compared to eigen-decomposition overheads of ordinary SPD representations. The gradient computation of $\frac{\partial J}{\partial \mathbf{X}}$ also involves the extra step of computing $\frac{\partial J}{\partial \mathbf{\Sigma}_1}$, some additional matrix multiplications with $\mathbf{X}$ and $\mathbf{\Sigma}_1$, and matrix addition with the term related to $\frac{\partial J}{\partial \mathbf{\Sigma}_2}$. These matrix addition and multiplication operations are computationally inexpensive. In fact, the step for obtaining $\frac{\partial J}{\partial \mathbf{\Sigma}_0}$ is required for any based matrix normalisation method based on eigen-decomposition, regardless of whether FastCOV is used or not. Moreover, since the size of the matrix $\frac{\partial J}{\partial \mathbf{\Sigma}_1}$ is smaller in our case, the computation of $\frac{\partial J}{\partial \mathbf{\Sigma}_0}$ takes significantly less runtime.

## 4.4 Experimental Results

We conduct extensive experiments on scene classification and fine-grained image classification datasets to investigate the proposed ReDro and FastCOV schemes. For scene classification, the *MIT Indoor* dataset [85] is used. For fine-grained image classification, the commonly used *Birds* [86], *Airplanes* [87], and *Cars* [88] 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 [3], [5], we resize all images to 448×448 during training and testing. More details on datasets and implementation of FastCOV are provided in the Appendix B and C.

Our experiments consist of two main parts. Sec. 4.4.1 describes the experiments with FatCOV, including its computational advantage, and its performance on several typical deep architectures for learning SPD representation. Sec. 4.4.2 compares the classification performance achieved by the two proposed methods with that of the state-of-the-art SPD representation based methods.

### 4.4.1 Experiments with the FastCOV scheme

This section presents experiments with the FastCOV scheme. We divide the experiments in two main parts. The first part shows the computational advantage brought by the proposed FastCOV scheme. The second part shows the performance of the FastCOV scheme with the two main-stream CNN networks widely used in the literature.

**Computational advantage of FastCOV**

This part compares the computational cost of obtaining covariance representation with FastCOV and other methods that use matrix normalisation. Specifically, we compare it with four typical covariance matrix based representation methods, namely, iSQRT-COV, IBCNN, MPN-COV, and DeepCOV. We do not compare with DeepKSPD since it focuses on SPD representation based on kernel matrix. All four covariance matrix based methods compute the *channel*-wise covariance matrix while the FastCOV method computes *position*-wise covariance matrix. In this comparison, we implement all methods with the ResNet-50 network as backbone.

Similarly to the experiments in Sec. 3.4.1 of previous chapter, we use the total computational time taken by the covariance computation and matrix normalisation for comparison.

| No. of feat. channels ($d$) | 128 | | | 256 | | |
|---|---|---|---|---|---|---|
| Size of feat. maps ($w \times h$) | 7×7 | 14×14 | 28×28 | 7×7 | 14×14 | 28×28 |
| iSQRT-COV [5] | 0.001 | 0.001 | 0.006 | 0.006 | 0.006 | 0.013 |
| I-BCNN [3] | 0.004 | 0.004 | 0.006 | 0.010 | 0.012 | 0.016 |
| MPN-COV [4] | 0.005 | 0.006 | 0.007 | 0.013 | 0.016 | 0.017 |
| DeepCOV [6] | 0.006 | 0.007 | 0.009 | 0.014 | 0.017 | 0.019 |
| FastCOV (ours) | 0.005 | 0.011 | 0.077 | **0.006** | 0.012 | 0.073 |
| No. of feat. channels ($d$) | 512 | | | 1024 | | |
| Size of feat. maps ($w \times h$) | 7×7 | 14×14 | 28×28 | 7×7 | 14×14 | 28×28 |
| iSQRT-COV [5] | 0.028 | 0.030 | 0.041 | 0.094 | 0.097 | 0.114 |
| I-BCNN [3] | 0.027 | 0.030 | 0.050 | 0.110 | 0.115 | 0.182 |
| MPN-COV [4] | 0.029 | 0.040 | 0.047 | 0.125 | 0.133 | 0.157 |
| DeepCOV [6] | 0.033 | 0.040 | 0.058 | 0.137 | 0.147 | 0.199 |
| FastCOV (ours) | **0.007** | **0.014** | 0.106 | **0.016** | **0.027** | 0.153 |

**Table 4.1:** Comparison of computational time (in second) for covariance computation and matrix normalisation by using *vs.* not using the proposed FastCOV scheme. The reported time is the sum of the times taken by the forward and backward propagations. The first four methods represent the cases not using FastCOV. All methods use the ResNet-50 network as backbone. The boldface shows that FastCOV saves computational time.

During comparison, when necessary, we apply the 1×1 convolution operation to reduce the number of channels and change the size of the input image to vary the size of feature maps. We perform this experiment with a computer with a Tesla P100 GPU, 12-core CPU, and 12 GB RAM. FastCOV is implemented with the MatConvNet library [44] on MATLAB 2019a.

Table 4.1 shows the timing result. Along the number of feature channels and the spatial size of feature maps, we can observe that (i) when the number of feature channels is small (i.e., 128) or the spatial size of the feature maps is large (i.e., 28×28), FastCOV is not needed. In the first case, a small channel-wise covariance matrix is computed and its eigen-decomposition cost is insignificant. In the second case, computing a position-wise covariance matrix incurs high matrix dimensions and eigen-decomposition cost of such a matrix is high; (ii) however, when the number of feature channels increases and the spatial size of feature maps decreases, the advantage of FastCOV becomes pronounced. For example, when the number of feature channels is 512 and the spatial size of feature maps is 7×7 or 14×14, FastCOV is ∼400% and ∼250% faster than that of other methods,

| Methods | Backbone | MIT | Airplane | Cars | Birds |
|---|---|---|---|---|---|
| DeepCOV [6] | VGG-16 | 79.2 | 88.7 | 91.7 | 85.4 |
| FastCOV-VGG (ours) | | **79.9** | **90.9** | 90.8 | **86.0** |
| DeepCOV-ResNet | ResNet-50 | 84.6 | 87.7 | 91.6 | 86.7 |
| FastCOV-ResNet (ours) | | **84.9** | **90.1** | **92.8** | 86.5 |

**Table 4.2:** Performance comparison of our FastCOV method with typical CNNs.

respectively. Furthermore, when the number of feature channels increases to 1024 and the spatial size of feature maps is 7×7 or 14×14, FastCOV becomes ~580% – ~860% and ~360% – ~540% faster than that of other methods, respectively. These two cases of feature maps correspond exactly to the situation in the advanced deep convolutional networks used in recent literature. Therefore, the above results demonstrate the computational advantage of the proposed FastCOV method. In addition, more details on the reduction of training time by FastCOV for learning large covariance matrix representation is given in the Appendix B.

### On the performance of FastCOV with typical CNNs

Based on the findings of the previous experiment, we now focus on learning large covariance matrix representation using our FastCOV. For this experiment, we choose ResNet-50 as it produces 2048 feature channels and 14×14-sized feature maps when fed with an image of 448×448 pixels. We call this method as "FastCOV-ResNet". The obtained results are compared with our DeepCOV-ResNet network from the previous chapter. Additionally, we perform experiments with VGG-16 and compare them with the DeepCOV network. Note that FastCOV incurs more computational cost with VGG-16 due to its large feature map size, i.e., 28×28, with 448×448 pixels input image, and lower feature channels, i.e., 512. We call this VGG-16 based FastCOV method as "FastCOV-VGG".

Table 4.2 shows that (i) Overall, FastCOV based methods achieve comparable performance with the DeepCOV and DeepCOV-ResNet across all datasets; (ii) Regardless of the size of the input feature maps in VGG-16 and ResNet-50, the performance of FastCOV based methods are comparable to DeepCOV based methods; (iii) Though the performance of FastCOV-ResNet is comparable with DeepCOV-ResNet, it is significantly faster than the other variant due to the efficient normalisation with FastCOV. Furthermore, these results suggest that our FastCOV performs similarly to regular covariance representation schemes such as DeepCOV.

| Methods | Backbone | Airplane | Cars | Birds |
|---|---|---|---|---|
| VGG-16 [18] | VGG-16 | 76.6 | 79.8 | 70.4 |
| NetVLAD [91] | | 81.8 | 88.6 | 81.6 |
| NetFV [92] | | 79.0 | 86.2 | 79.9 |
| BCNN [1] | | 83.9 | 90.6 | 84.0 |
| CBP [2] | | 84.1 | 91.2 | 84.3 |
| LRBP [28] | | 87.3 | 90.9 | 84.2 |
| KP [25] | | 86.9 | 92.4 | 86.2 |
| HIHCA [93] | | 88.3 | 91.7 | 85.3 |
| Improved BCNN [3] | | 88.5 | 92.0 | 85.8 |
| SMSO [79] | | – | – | 85.0 |
| MPN-COV [29] | | 89.9 | 92.2 | 86.7 |
| G$^2$DeNet [29] | | 89.0 | **92.5** | **87.1** |
| iSQRT-COV [5] (reproduced)$^\dagger$ | | 88.5 | 86.4 | 78.5 |
| DeepCOV [6] | | 88.7 | 91.7 | 85.4 |
| DeepKSPD [6] | | 90.0 | 91.6 | 84.8 |
| DeepCOV+ ReDro (Chapter 3) | | 89.2 | 92.2 | 86.7 |
| FastCOV-VGG (ours) | | **90.9** | 90.8 | 86.0 |
| CBP [2] | ResNet-50 | 81.6 | 88.6 | 81.6 |
| KP [25] | | 85.7 | 91.1 | 84.7 |
| SMSO [79] | | – | – | 85.8 |
| iSQRT-COV [5] | | 89.5 | 91.7 | **87.3** |
| DeepCOV-ResNet (Chapter 3) | | 87.7 | 91.6 | 86.7 |
| DeepCOV-ResNet+ ReDro (Chapter 3) | | 86.8 | **92.0** | 86.9 |
| FastCOV-ResNet (ours) | | **90.1** | **92.8** | 86.5 |

**Table 4.3:** Comparison between the proposed methods and other SPD representation methods in terms of classification accuracy (%). The performance of existing SPD representation methods are quoted from the original papers. The best methods across datasets and CNN backbones are marked in bold. $^\dagger$With the same hyper-parameter settings, CNN backbone and training strategy as [6], and default iSQRT-COV iterations of 5 times.

### 4.4.2 Comparison with Other Methods

In this section, we compare our proposed method with the existing SPD representation methods and ReDro based methods proposed in the previous chapter. Specifically, we compare with fifteen state-of-the-art methods, namely, VGG-16 [18], NetVLAD [91], NetFV [92], BCNN [1], CBP [2], LRBP [28], KP [25], HIHCA [93], Improved BCNN [3], SMSO [79], MPN-COV [29], G$^2$DeNet [29], iSQRT-COV [5], DeepCOV [6], and DeepKSPD [6]. For fairness, we divide them into two groups based on which CNN backbone they have used, i.e., VGG-16 or ResNet-50.

The comparison is shown in Table 4.3. For ReDro based DeepCOV methods, we report the best results from Table 3.2. The top part of Table 4.3 shows the results of the methods that uses the VGG-16 network as the backbone. As shown in the table, (i) our FastCOV-VGG achieves comparable results with the existing methods on all three datasets; (ii) on Airplane dataset, our FastCOV-VGG method performs better than many existing methods, i.e., Improved BCNN, MPN-COV, G$^2$DeNet, iSQRT-COV, DeepCOV, and DeepKSPD. However, in terms efficiency, our FastCOV is slightly lower than the existing methods due to the larger spatial feature map size of VGG-16. The better result of G$^2$DeNet [29] on the Cars and Birds datasets than our proposed method could be due to the fact that its backbone network is pre-trained with the second-order representation while the backbone networks pre-trained in our methods are just the commonly used ones which do not involve any second-order representation. Using a backbone network pre-trained in the way of G$^2$DeNet may further improve the performance of our proposed method.

The bottom part of Table 4.3 shows the results of the methods that use the ResNet-50 network as the backbone. As can be seen, (i) our FastCOV-ResNet method performs comparably to other methods across all datasets. Note that, our FastCOV-ResNet is significantly faster than the existing methods; (ii) On the Birds dataset, iSQRT-COV attains better results than the proposed methods. This is again due to the use of a backbone network that has been pre-trained with second-order representation in the method. However, our FastCOV-ResNet is significantly faster than iSQRT-COV and will enjoy the performance improvement if the same pre-trained backbone model is used.

## 4.5 Conclusion

In this chapter, we propose a method dubbed FastCOV for efficient eigen-based normalisation in SPD visual representations. Opposite to ReDro in the previous chapter which uses channel-wise block-diagonal matrix trick to reduce the matrix normalisation time, FastCOV uses a position-wise matrix trick to reduce the matrix normalisation time. In recent deep networks such as ResNet [12], the spatial resolution of feature maps is low and the size of position-wise SPD matrices on those smaller feature maps is significantly smaller than channel-wise SPD matrices, e.g., a channel-wise SPD matrix from 1024 channels of $14 \times 14$ sized feature maps has a size of $1024 \times 1024$ but a position-wise SPD matrix has a size of only $196 \times 196$. Performing matrix normalisation on such a small SPD matrix is computationally less expensive.

Very recent deep networks such as ResNet and ConvNeXt [102] which have small spatial feature map resolution but a higher number of channels can take advantage of FastCOV for effective second-order statistics estimation. By principle, for networks with small spatial feature maps, FastCOV can be significantly faster than ReDro in estimating SPD matrix estimation. In addition, it is general purpose and has no learnable parameters, therefore, can be directly integrated with existing SPD representation frameworks without any difficulty for better second-order representation learning. Extensive experiments on multiple datasets demonstrate that FastCOV is able to achieve competitive results compared with existing methods, including ReDro.

In our current work, we have applied FastCOV on fine-grained image classification and scene classification problems. However, FastCOV is a general-purpose SPD representation method. In the future, we plan to apply it to problems beyond image classification, such as image segmentation, object detection and video classification.

# Chapter 5

# Learning SICE based Visual Representation

*The work of this chapter has been accepted at the 2023 IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR 2023) to be held in Vancouver, Canada.*

Covariance matrix based visual representation with convolutional neural networks (CNNs) has attracted significant research interest in the past several years. Despite the success, reliable covariance matrix estimation from convolutional feature maps is still a challenging issue due to the small spatial size of feature maps and the large number of feature channels in advanced CNNs. Sparse inverse covariance estimation (SICE) has been developed in the literature to address this situation. However, how to integrate this process into the CNNs to address this covariance estimation issue for visual representation has not been investigated before. This chapter integrates SICE optimisation as a novel structured layer into CNN. To realise end-to-end training of the resultant CNN, we develop an iterative method based on Newton-Schulz iteration to solve the SICE optimisation during backpropagation. By doing so, we mitigate the small sample problem for the covariance estimation in CNN. On top of that, our proposed method is fully compatible with GPU parallelisation and a large number of CNN feature channels. We perform extensive experiments with various hyperparameters of the proposed method using one scene and three fine-grained image datasets to assess its robustness. Our experimental results confirm that the proposed method significantly outperforms its covariance matrix based counterparts.

## 5.1 Introduction

Learning effective visual representation has been a challenging issue in computer vision. In the past several years, describing images with local features and pooling them into a global representation has shown promising performance. Among the pooling methods, the covariance matrix has attracted much attention in recent years due to its exploitation of second-order information of features. Due to the symmetric positive definite (SPD) characteristic of the covariance matrix, the resulting representation is also known as SPD visual representation. Various tasks such as fine-grained image classification, image segmentation, generic image classification, image set classification, action recognition and few-shot learning have shown excellent performance using covariance matrix based representation. Motivated by the powerful representation ability of deep learning, a few pioneering works have integrated covariance matrix as a pooling method with convolutional neural network (CNN) and investigated several associated issues such as matrix function backpropagation [11], matrix normalisation [4], [92], compact matrix estimation [25] and kernel based extension [6]. This progress has improved the effectiveness of covariance matrix based representation in the tasks mentioned above.

Regardless of these achievements, estimating a reliable covariance matrix from the local descriptors of a CNN feature map is still a challenging task. The challenges primarily come from small spatial size, i.e., sample size, and higher number of channels, i.e., feature dimension, which becomes more pronounced for advanced CNN models. These challenges critically affect the effectiveness and precision of the covariance matrix as a visual representation. Furthermore, it also sometimes leads to the issue of matrix singularity. One might argue that by increasing the size of the input image and using the dimension reduction layer to reduce the number of feature channels, the above issues can be resolved. In this paper, we are interested in whether these challenges can be reduced from a different point of view.

The work in this paper is motivated by the use of prior knowledge in the covariance matrix estimation process to address the above challenges. Prior knowledge is a form of domain knowledge that can come from the domain theory of a specific vision application. For example, the "structure sparsity" of tree-like shaped structure of human skeletons and the general principles of estimating the structure of high dimensional data such as "Bet on Sparsity". This leads to the following technique, called sparse inverse covariance estimation (SICE). It solves an optimisation on the SPD matrix to estimate the inverse covariance matrix by imposing a sparsity constant on its entries. It has demonstrated excellent effectiveness in dealing with the small sample problem. In the literature, some researchers have used SICE to improve covariance estimation on handcrafted and pre-extracted CNN features to demonstrate the promising performance of SPD visual representation [13]. Their work has demonstrated that SICE has the following advantages over the covariance matrix: (1) it has more robustness against a smaller sample; (2) it is guaranteed to be free from matrix singularity issue; (3) it can better characterise the underlying structure of high dimensional data with the use of prior information; (4) it can give better classification performance; (5) when it comes to interpretation of feature relationship, SICE reveals it better than covariance matrix.

However, SICE is applied only to handcrafted or pre-extracted CNN features. It has not been integrated into CNN for end-to-end training. Without fully integrating SICE into CNN in this way, it can not fully take advantage of the automatic feature learning capability offered by CNN. Unlike the covariance matrix, which is obtained with simple arithmetic operations, SICE is obtained by solving an SPD matrix based convex optimisation problem. How to incorporate this optimisation problem into CNN as a layer is an interesting issue. This SICE needs to be done for each image during each forward and backward phase. An off-the-shelf package such as cvxpylayers [103] can be used for SICE, but they still largely depend on the CPU and are not scalable to large SICE problems (will be discussed in more detail in the following sections). To improve this condition, we propose a fast end-to-end training method for SICE by taking inspiration from Newton-Schulz iteration [104] and gradient descent algorithm. Our method solves the SICE optimisation with a smooth convex cost function. The cost function does not have any non-smooth term as the standard SICE cost function (please see the Eq. (5.1)) and can be optimised with standard optimisation algorithms such as gradient descent. Furthermore, it effectively enforces SPD constraints during the optimisation process so that the optimisation solution remains SPD. We call our method "Iterative Sparse Inverse Covariance Estimation (iSICE)". We will discuss our method in detail in the following sections. iSICE involves simple matrix arithmetic operations and is fully compatible with GPU. It can approximately solve large SICE problems with CNN effectively.

The main contributions of this paper are summarised as follows.

i. To address the challenges of covariance matrix estimation in the presence of small feature maps and a higher number of channels in CNN, this paper proposes a method called iSICE for SICE. To the best of our survey, we are the first to develop an end-to-end trainable method for conducting SICE within deep neural networks.

ii. Our iSICE method requires minimal change in the existing network architecture, so it can be integrated with existing covariance matrix based representation without any significant effort. It is easily implementable with modern deep learning libraries such as PyTorch and trainable with automatic differentiation.

We have conducted extensive experiments on one scene recognition dataset and three fine-grained image recognition datasets to demonstrate the effectiveness of the proposed method.

## 5.2 Related works

Since the advent of deep learning based covariance representation methods such as Bilinear CNN (BCNN) [1] and compact BCNN [2], reliable and robust estimation of the covariance matrix from CNN feature channels remains an issue. Early work such as BCNN and compact BCNN focused more attention on the end-to-end learning of the covariance matrix with CNN. They used signed square root normalisation [1] to minimise the impact of unreliability, e.g., the biased issue of eigenvalue. More recent works use matrix power normalisation which normalises the eigenvalues of the covariance matrix. Matrix power normalisation helps to estimate reliable covariance matrix by reducing the swelling effect, i.e., large eigenvalues become more large and small eigenvalues become smaller. One group of these methods include DeepO$_2$P [11], Improved BCNN [92], MPN-COV [4] and DeepKSPD [6]. These methods perform matrix square root or logarithm normalisation using eigendecomposition. Two major issues encountered by these methods are (1) *Numerical instability during backpropagation:* when two eigenvalues become very close to each other, one cancels out the other which raises the infinity issue while performing backpropagation. Most works appended a small value to the eigenvalues and the work in [6] used the Daleckii-Krein formula to avoid this problem. (2) *Limited acceleration with GPU*: the existing algorithm implementation of eigendecomposition in well-known libraries such as PyTorch has limited GPU acceleration support. The work in [5] conducted a thorough investigation on the acceleration of eigendecomposition with GPU and commented that the CPU is more efficient than the GPU to perform eigendecomposition.

The other group of matrix power normalisation methods includes Improved BCNN [92], iSQRT-COV [5] and RUN [24]. These methods use Newton-Schulz or other iterative methods to perform approximate matrix square root normalisation. Compared to methods in the previous group, Newton-Schulz iteration based methods are magnitude faster with GPU. In terms of performance, they give comparable performance to eigen-decomposition based matrix power normalisation methods with only a few iterations. Furthermore, they do not face any numerical instability issues during backpropagation. There are also other types of methods such as the ones in [105], [106] that are related to the works of the above two groups.

Research progress in the above works shows that reliable covariance estimation from small feature maps and higher number channels in modern CNN is an important task. The following parts try to achieve that with our proposed method.

## 5.3 Proposed method

In this section, we begin by discussing the background of SICE. Then we discuss how it can be estimated from CNN feature descriptors. Finally, we describe our proposed method which is trainable end-to-end with a CNN.

### 5.3.1 The basic idea of sparse inverse covariance estimation

As a representation, the covariance matrix captures the underlying structure of a visual feature set. Assuming a Gaussian model, it uses a covariance estimated from samples to capture this structure. SICE focuses on the suitability of such estimation of covariance. Specifically, it focuses on the following two issues: (1) instability or singularity of estimated sample based covariance from a small number of higher dimensional feature vectors. These conditions make it less effective in capturing the underlying structure of data. As an example, under these conditions the smaller and larger eigenvalues of covariance become biased, therefore, suitable regularisation is needed to reduce the bias; (2) rigid estimation of covariance from the complex structure of higher dimensional feature vectors is not always appropriate. Sometimes, there is task-specific prior knowledge available. The incorporation of such knowledge improves the covariance estimation from a small number of samples.

In terms of prior knowledge of higher-dimensional data, structure sparsity [107] and ‘bet on sparsity’ principle [108] are the two common knowledge used in the literature. Suppose that we have a probabilistic graphical model and we describe its distribution with graphs, where each node belongs to a feature and the statistical dependence between two nodes is expressed with an edge. Structure sparsity would specify how sparse a graph is, e.g., only a few edges are connected in a skeletal model of a human body. More generally, the bet on sparsity principle focuses on estimating the structure of the graph by imposing a sparsity prior. If the graph is indeed sparse, it will estimate its underlying structure with a correct prior and if the graph is dense, it will not estimate the underlying structure accurately. However, in the latter case, the loss will not be significant because we have known that we do not have enough samples to estimate the dense structure. SICE tries to improve the covariance estimation with the use of prior knowledge.

To incorporate prior knowledge, SICE switches to the inverse of the covariance matrix from the covariance matrix. In principle, the covariance matrix captures the correlation between feature components without paying attention to their correlation type, i.e., direct and indirect. In comparison, the inverse of the covariance matrix only captures direct (partial) correlations between feature components by removing the irrelevant ones and allows the convenient incorporation of sparsity priors. Below we discuss the details of computing SICE.

### 5.3.2 SICE estimation with CNN

Suppose, we have the covariance matrix $\mathbf{\Sigma}$ computed from CNN descriptors (the details of computing covariance matrix will be discussed in the following section) and $\mathbf{\Sigma}^{-1}$ is its inverse. The off-diagonal entries of $\mathbf{\Sigma}^{-1}$ capture the partial (i.e. direct) correlation between different descriptor components. They will be zero if two features are independent in terms of conditions in the presence of the remaining ones. In the literature, [13], the estimation of $\mathbf{\Sigma}^{-1}$ has been effectively resolved by the maximisation of a penalised log-likelihood of data with an SPD constraint on $\mathbf{\Sigma}^{-1}$. The optimal solution to the above problem is known as SICE.

$$\mathbf{S}^* = \arg\max_{\mathbf{S}\succ 0} \log[\det(\mathbf{S})] - \text{trace}(\mathbf{\Sigma}\mathbf{S}) - \lambda||\mathbf{S}||_1, \tag{5.1}$$

where $\mathbf{\Sigma}$ is a sample based covariance matrix, and $\det(\cdot)$, $\text{trace}(\cdot)$ and $||\cdot||_1$ denote the determinant, trace and $\ell 1$-norm of a matrix, respectively. To obtain reliable and faithful SICE, the term $||\mathbf{S}||_1$ imposes structure sparsity on $\mathbf{S}$. $\lambda$ is the sparsity value which controls the trade-off between the amount of sparsity and the log-likelihood estimation. A large value of $\lambda$ induces a sparser $\mathbf{S}$ and q small value of $\lambda$ induces a less sparse $\mathbf{S}$. The maximisation problem in Eq. (5.1) is convex and can be straightforwardly solved by the off-the-shelf packages such as GLASSO [109] and CVXPY [110]. However, these optimisation packages can not be readily used with CNN layers to conduct training with backpropagation. A recent extension of CVXPY called cvxpylayers [103] provides differentiable optimisation layers but based on our investigation, it has the following issues: (1) it can not efficiently solve large SICE problems, i.e., of size $128 \times 128$ or higher; (2) it relies on multiple CPU based libraries including CVXPY to solve the optimisation problem and obtain gradients for backpropagation. This greatly limits its efficiency with GPU. The above limitations motivate us to develop a SICE method suitable for end-to-end training with GPU. Below we discuss the details of our proposed method.

### 5.3.3 Proposed end-to-end trainable SICE method

Let $J$ be the objective function of Eq. (5.1). It can be optimised by taking the gradient with respect to $\mathbf{S}$ as follows.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{S}} &= \frac{\partial}{\partial \mathbf{S}}(\log[\det(\mathbf{S})]) - \frac{\partial}{\partial \mathbf{S}}\text{trace}(\mathbf{\Sigma}\mathbf{S}) - \lambda\frac{\partial}{\partial \mathbf{S}}||\mathbf{S}||_1 \\ &= \mathbf{S}^{-1} - \mathbf{\Sigma} - \lambda(\frac{\partial}{\partial \mathbf{S}}\mathbf{S}^+ - \frac{\partial}{\partial \mathbf{S}}\mathbf{S}^-) \\ &= \mathbf{S}^{-1} - \mathbf{\Sigma} - \lambda(\text{sign}(\mathbf{S}^+) - \text{sign}(\mathbf{S}^-)) \end{aligned} \tag{5.2}$$

where $\mathbf{S}^+ = \max(0, \mathbf{S})$ and $\mathbf{S}^- = \max(0, -\mathbf{S})$ contain the positive and negative parts of $\mathbf{S}$, respectively. Eq. (5.2) is optimisable with projected gradient descent. These algorithms have native backpropagation support and can take advantage of GPU parallel computing to improve their speed. Below discuss how Eq. (5.2) can be effectively optimised using a few consecutive structured CNN layers in four steps. We begin from the initial phase.

The overview of our method is given in Fig. 5.1. From the left, we give an input image

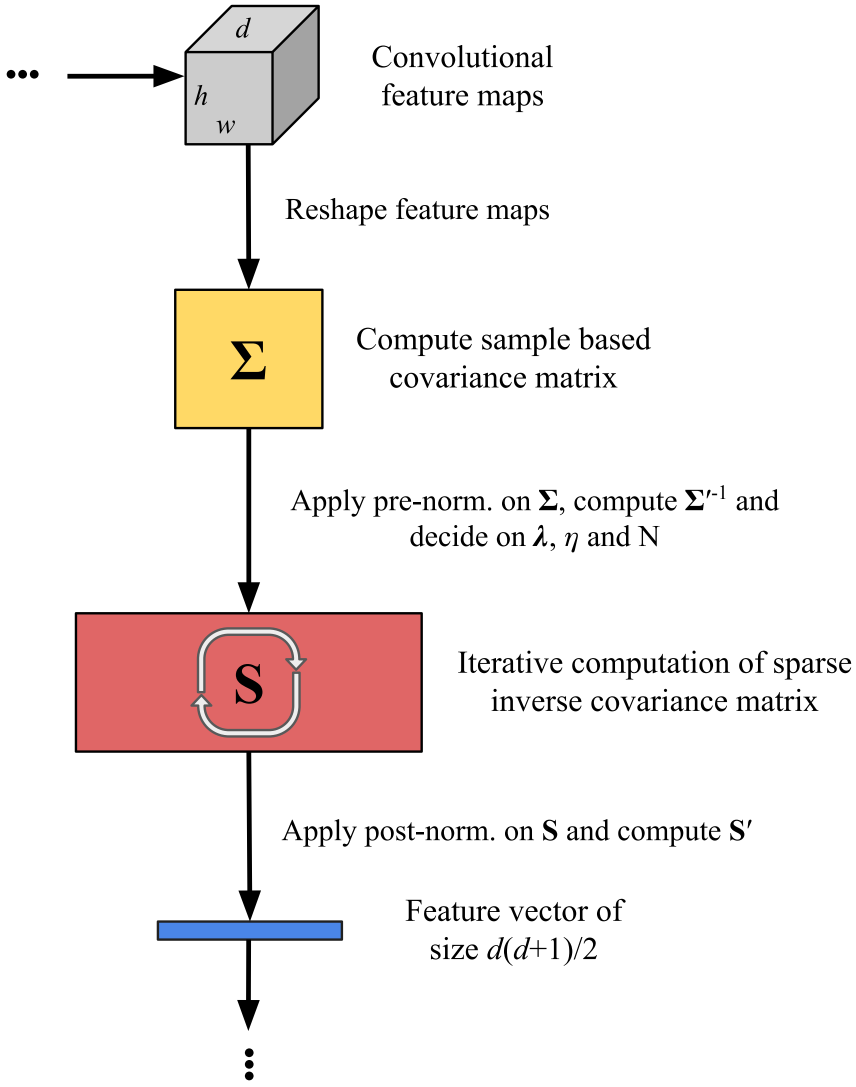
d
h
w
Convolutional feature maps
Reshape feature maps
Σ
Compute sample based covariance matrix
Apply pre-norm. on $\mathbf{\Sigma}$, compute $\mathbf{\Sigma}'^{-1}$ and decide on $\boldsymbol{\lambda}$, $\eta$ and N
S
Iterative computation of sparse inverse covariance matrix
Apply post-norm. on $\mathbf{S}$ and compute $\mathbf{S}'$
Feature vector of size $d(d+1)/2$

**Figure 5.1:** Proposed iterative sparse inverse covariance estimation (iSICE) method.

to the CNN and process it till the last convolution layer to obtain the feature maps. The feature maps are obtained as a $h \times w \times d$ tensor, where $h$ is the height, $w$ is the width, and $d$ is the number of channels. By converting the feature maps to long vectors of $n$-dimension, where $n = w \times h$ and stacking them as a row, we can create a $d \times n$ dimensional data matrix $\mathbf{X}$. A covariance matrix $\mathbf{\Sigma}$ based on the samples in $\mathbf{X}$ is computed as follows.

$$\mathbf{\Sigma} = \mathbf{X}\bar{\mathbf{I}}\mathbf{X}, \tag{5.3}$$

where $\bar{\mathbf{I}} = \frac{1}{n}(\mathbf{I} - \frac{1}{n}\mathbf{1})$ performs centering of matrix $\mathbf{X}$, $\mathbf{I}$ and $\mathbf{1}$ are $n \times n$ dimensional identity matrix and matrix of ones, respectively. Now, we apply the following four steps in sequential order.

**Pre-normalisation of sample based covariance matrix $\mathbf{\Sigma}$ with trace.** Similar to Newton-Schulz method [5], our proposed method ensures the local convergence of SICE under the condition that $||\mathbf{\Sigma}-\mathbf{I}|| < 1$. To fulfil this condition, we normalise the covariance matrix $\mathbf{\Sigma}$ with its trace and obtain the trace normalised covariance matrix $\mathbf{\Sigma}'$

$$\mathbf{\Sigma}' = \frac{\mathbf{\Sigma}}{\text{trace}(\mathbf{\Sigma})} \tag{5.4}$$

**Estimation of inverse covariance matrix $\mathbf{\Sigma}'^{-1}$.** We assume $\mathbf{\Sigma}'$ is an invertible covariance matrix and estimate its inverse $\mathbf{\Sigma}'^{-1}$ using an algorithm based on Newton-Schulz iterations. We resort to Newton-Schulz iterations due to its computational benefits with GPU, as shown in the work of Li et al. [5]. The algorithm works in three steps: (1) Pre-normalisation with trace, (2) Approximate matrix root estimation with power $-\frac{1}{2}$, (3) post-normalisation with trace and (4) Estimation of $\mathbf{\Sigma}'^{-1}$ with the output of the third step. To improve the reliability of $\mathbf{\Sigma}'^{-1}$ estimation, we add a small constant (i.e., 1e-9) to the diagonal entries of $\mathbf{\Sigma}'$ before Newton-Schulz iterations. We iterate our algorithm seven times. Algorithm 3 shows the details of the above steps.

**Algorithm 3:** Computation of matrix inverse with Newton-Schulz iterations (matInv)

```
Input: Covariance matrix $\mathbf{\Sigma}$, Number of iterations N
Output: Inverse covariance matrix $\mathbf{\Sigma}^{-1}$
$\mathbf{\Sigma}' = \mathbf{\Sigma}/\text{trace}(\mathbf{\Sigma})$ ;        /* Pre-normalisation using trace */
$\mathbf{P} = \frac{1}{2}(3\mathbf{I} - \mathbf{\Sigma}')$;
$\mathbf{Y}_0 = \mathbf{\Sigma}'\mathbf{P}$;
$\mathbf{Z}_0 = \mathbf{P}$;
for $i = 1$ to $\text{N} - 1$ do
    $\mathbf{P} = \frac{1}{2}(3\mathbf{I} - \mathbf{Z}_{i-1}\mathbf{Y}_{i-1})$;
    $\mathbf{Y}_i = \mathbf{Y}_{i-1}\mathbf{P}$;
    $\mathbf{Z}_i = \mathbf{P}\mathbf{Z}_{i-1}$;
$\mathbf{Q} = \frac{1}{2}(3\mathbf{I} - \mathbf{Z}_{\text{N}-2}\mathbf{Y}_{\text{N}-2})\mathbf{Z}_{\text{N}-2}$;
$\mathbf{Q}' = \mathbf{Q}/\sqrt{\text{trace}(\mathbf{\Sigma})}$ ;        /* Post-normalisation using trace */
$\mathbf{\Sigma}^{-1} = \mathbf{Q}'\mathbf{Q}'^{\top}$ ;              /* Inverse covariance matrix */
```

**Estimation of sparse inverse covariance S.** We apply sparsity to the $\mathbf{\Sigma}'^{-1}$ by solving the optimisation problem in Eq. (5.2) with projected gradient descent (PGD). We decompose $\mathbf{\Sigma}'^{-1}$ into its positive and negative parts and optimise them separately with PGD in several iterations by enforcing the sparsity and SPD constraint. Denoting $\mathbf{\Sigma}'^{-1}$ with $\mathbf{\Lambda}_0$, for $i = 1, ..., N$, the solution of optimisation problem in Eq. (5.2) with PGD takes the following form:

$$\begin{aligned}
\mathbf{\Lambda}_i^+ &= \max(0, \max(0, \mathbf{\Lambda}_{i-1}^+) - \eta\beta\nabla\mathbf{\Lambda}_{i-1}^+) \\
\mathbf{\Lambda}_i^- &= \max(0, \max(0, \mathbf{\Lambda}_{i-1}^-) - \eta\beta\nabla\mathbf{\Lambda}_{i-1}^-) \\
\mathbf{\Lambda}_i &= \mathbf{\Lambda}_i^+ - \mathbf{\Lambda}_i^-, \mathbf{\Lambda}_i = (\mathbf{\Lambda}_i + \mathbf{\Lambda}_i^\top)/2,
\end{aligned} \tag{5.5}$$

where $\max(0,\cdot)$ is a projection function replaceable with $\mathrm{ReLU}(\cdot)$ operator, $\eta$ is learning rate, $\beta$ is a regularisation parameter, and $\nabla\mathbf{\Lambda}^+$ & $\nabla\mathbf{\Lambda}^-$ are the positive and negative parts of $\nabla\mathbf{\Lambda}$, respectively, and are defined as follows: $\nabla\mathbf{\Lambda}^+ = \mathbf{\Sigma}'^{-1} + \mathbf{\Sigma}_{Sym+} + \lambda$, $\nabla\mathbf{\Lambda}^- = -(\mathbf{\Sigma}'^{-1} + \mathbf{\Sigma}_{Sym+}) + \lambda$.

The sparse inverse covariance $\mathbf{S}$ can be obtained from sample based covariance and the optimised $\mathbf{\Lambda}$ with PGD as follows.

$$\mathbf{S} = \mathbf{\Lambda}_n \tag{5.6}$$

**Post-normalisation of sparse inverse covariance matrix S with trace.** To ensure that the changes of magnitude of the data by the pre-normalisation step earlier do not affect the performance or convergence of the network, we perform post-normalisation of $\mathbf{S}$ with trace

$$\mathbf{S}' = \frac{\mathbf{S}}{\sqrt{\mathrm{trace}(\mathbf{S})}} \tag{5.7}$$

Based on Eq. (5.5), $\mathbf{S}'$ is SPD, therefore, only its upper-triangular entries can be processed by fully connected layers for classification purposes. The above steps are further summarised in Algorithm 4. Algorithm 4 is fully supported by GPU and we optimise it with the autograd package. Due to the involvement of iterations for solving $\mathbf{S}$, we call our method iterative SICE (iSICE).

## 5.4 Experiments

In this section, we first describe our datasets and then discuss the implementation of our proposed method. After that, we present our experimental results and ablation study on key hyper-parameters. Finally, we compare our proposed method with the existing methods.

### 5.4.1 Datasets

We conduct experiments using one scene and three fine-grained image datasets. For scene image classification, we have used the MIT Indoor dataset [85]. For fine-grained

```
Algorithm 4: Iterative sparse inverse covariance estimation (iSICE)
Input: Sample based covariance matrix $\mathbf{\Sigma}$, Sparsity constant $\lambda$, Learning rate $\eta$,
  Number of iterations N, Small constant $\alpha$, i.e., $\alpha$ = 1e-9
Output: Sparse inverse covariance matrix $\mathbf{S}'$ $\mathbf{\Sigma}' = \mathbf{\Sigma}/\text{trace}(\mathbf{\Sigma})$ ;
  /* Pre-normalisation using trace */
$\mathbf{\Sigma}^{-1} = \text{matInv}(\mathbf{\Sigma}' + \alpha\mathbf{I})$ ;          /* Compute inverse of $\mathbf{\Sigma}'$ */
$\mathbf{\Lambda}_0 = \mathbf{\Sigma}^{-1}$;
for $i = 1$ to N do
    $\mathbf{\Lambda}_i^+ = \text{ReLU}(\mathbf{\Lambda}_{i-1})$, $\mathbf{\Lambda}_i^- = \text{ReLU}(-\mathbf{\Lambda}_{i-1})$ ;      /* Projection onto set */
    $\nabla_1 = \text{matInv}(\mathbf{\Sigma}' + \alpha\mathbf{I})$;
    $\nabla_2 = (\mathbf{\Sigma}' + \mathbf{\Sigma}'^{\top})/2$;
    $\nabla_{12} = \nabla_1 + \nabla_2$;
    $\nabla_3^+ = \nabla_{12} + \lambda$, $\nabla_3^- = -\nabla_{12} + \lambda$;
    $\beta = 1 - (i/\text{N} - 1)$;
    $\mathbf{\Lambda}_i^+ = \text{ReLU}(\mathbf{\Lambda}_i^+ - \eta\beta\nabla_3^+)$, $\mathbf{\Lambda}_i^- = \text{ReLU}(\mathbf{\Lambda}_i^- - \eta\beta\nabla_3^-)$;   /* Re-projection */
    $\mathbf{\Lambda}_i = \mathbf{\Lambda}_i^+ - \mathbf{\Lambda}_i^-$;
    $\mathbf{\Lambda}_i = (\mathbf{\Lambda}_i + \mathbf{\Lambda}_i^{\top})/2$;
$\mathbf{S} = \mathbf{\Lambda}_n$;
$\mathbf{S}' = \mathbf{S}/\sqrt{\text{trace}(\mathbf{S})}$ ;          /* Post-normalisation using trace */
```

image classification, we have used airplane [87], birds [86] and cars [88] datasets. For all datasets, we have used their pre-defined splits for training and testing in our experiments. We do not take advantage of the provided annotations such as bounding boxes or parts. The details of the datasets are given in Appendix C.

### 5.4.2 Implementation details

We implement our method using the PyTorch library. We use the pretrained CNN models provided in the torchvision package. We resize the images to 448×448 pixels following the widely used experimental protocol introduced by [1]. We implement our iSICE method in four computational blocks: pre-normalisation, inverse covariance estimation, SICE optimisation and post-normalisation. All of our blocks are implemented using Python to take advantage of the native GPU support provided in PyTorch. We experiment with VGG-16 [18] and ResNet-50 [12] CNN architectures. On both architectures, for computing iSICE efficiently, we add a $1 \times 1$ convolution layer (with ReLU and batch normalisation) after their last convolution layer to reduce the number of feature channels to 256. This results in a $28 \times 28 \times 256$ sized tensor in the case of VGG-16 and a $14 \times 14 \times 256$ sized tensor in the case of ResNet-50. Therefore, for both architectures, we compute a $256 \times 256$ size SICE matrix and use only the upper-triangular entries as a visual representation for image classification.

For training our model, we have used only horizontal flipping as a data augmentation technique. We have a batch size of 50 for all VGG-16 based experiments and a batch size

| Name of hyper-parameter | Values |
|---|---|
| Sparsity constant $\lambda$ | 1.0, 0.5, 0.1, **0.01**, 0.001, 0.0001, 0.00001 |
| Learning rate $\eta$ | 0.001, 0.01, 0.1, **1.0**, 5.0, 10.0, 20.0 |
| No. of iterations N | 1, **5**, 10 |

**Table 5.1:** iSICE hyper-parameters values used in our experiments. For the 'iSICE (Bold HP)' variant, we choose the middle values of each hyper-parameter (highlighted in bold).

of 40 for all ResNet-50 based experiments. All of our models are trained using twelve CPUs and four P100 GPUs for 100 epochs with an initial learning rate of 0.00012. We also use learning rate decay with a factor of 0.1 at the 15th and 30th epochs. All of our CNN architectures are optimised using the AdamW optimiser [111].

### 5.4.3 On the performance of the proposed method with its related counterparts

Our first experiment is to compare our proposed iSICE method with the regular covariance and inverse covariance representations. Since our method is inspired by Newton-Schulz (NS) iterations, for consistency, we compare it with NS iterations based on covariance and inverse covariance representation methods. We use the iSQRT-COV method [5] for obtaining covariance representation and Algorithm 3 for obtaining inverse covariance representation. For all methods, after obtaining convolutional feature channels, we compute the appropriate SPD matrix representation and process its upper-triangular entries with a fully connected layer for obtaining classification scores.

To understand the robustness of our iSICE method, we run experiments with its different hyper-parameters, i.e., sparsity constant $\lambda$, learning rate $\eta$ and number of iterations N. The combinations of iSICE hyper-parameters we experiment with are shown in Table 5.1. Since there are 147 combinations of hyper-parameter values shown in the Table, for the case of presentation, we report the performance of iSICE in two variants. The first variant is with the median values from the range of hyper-parameter values shown in Table 5.1, i.e., $\lambda = 0.01$, $\eta = 1.0$ and N = 5. This variant shows how our proposed method performs across different datasets are CNN backbones with the consistent hyper-parameter setting. We denote this variant by 'iSICE (Bold HP)'. The second variant is with the inconsistent hyper-parameter setting. We run our method with all hyper-parameter combinations from Table 5.1 and choose the combination that gives the highest performance (i.e., on the test splits) among others. This variant shows how our method would perform when the best hyper-parameters are available. We denote this variant with 'iSICE (Best HP)'. More discussion on the performance of our proposed method using other hyper-parameters will be given in the following paragraphs.

| | Based on VGG-16 backbone | | | | Based on ResNet-50 backbone | | | | Average |
|---|---|---|---|---|---|---|---|---|---|
| Method | MIT | Airplane | Birds | Cars | MIT | Airplane | Birds | Cars | |
| COV | 76.12 | 90.01 | 84.47 | 91.21 | 78.81 | 90.88 | 84.26 | 92.13 | 85.99 |
| Inv. COV | 80.15 | 89.44 | 83.36 | 92.04 | 80.75 | 91.15 | 84.67 | 91.99 | 86.69 |
| iSICE (Bold HP) | 78.66 | **92.23** | **86.52** | **94.03** | **80.52** | **92.74** | **85.90** | **93.51** | **88.01** |
| iSICE (Best HP) | **80.52** | **93.28** | **86.90** | **94.11** | **82.09** | **93.55** | **86.61** | **93.92** | **88.87** |

**Table 5.2:** Classification accuracy of our proposed method using two CNN backbones. We report two variants of iSICE. The iSICE (Same HP) variant is based on the bold hyper-parameters (HP) in Table 5.1 and iSICE (Best HP) variant is based the HP combinations that gives the best performance in our experiments. Our performance is compared with iSQRT-COV [5] (denoted with 'COV') and Algorithm 3 (denoted with 'Inv. COV').

Table 5.2 compares the performance of our method with its covariance counterparts. On the left side, we compare both of our iSICE variants with regular covariance and inverse covariance methods. It is clear that on average, our proposed method clearly achieves better performance than its covariance counterparts. This achievement is consistent across our iSICE variant with the same hyper-parameters, i.e., iSICE (Bold HP), and iSICE variant with best hyper-parameters, i.e., iSICE (Best HP). The iSICE (Best HP) outperforms iSICE (Bold HP) by $\sim 0.9\%$. It is interesting to see that except a few cases, the inverse covariance method, i.e., Inv. COV, performs slightly better than the covariance method, i.e., COV. This improved performance shows that the inverse covariance matrix improves the effectiveness of the covariance matrix by characterising partial correlations of feature components instead of indirect correlations of feature components. Our iSICE makes the inverse covariance matrix more robust and reliable by enforcing sparsity, as shown by the improved performance.

From the top, we divide our performance on various datasets based on CNN architectures. The results using both CNN architectures are comparable. VGG-16 based iSICE appears to receive better performance than the ResNet-50 based iSICE. This is probably due to the larger feature map size, i.e., sample size, of VGG-16 than ResNet-50 which results in a more reliable sample based covariance estimate. Across some fine-grained datasets, inverse covariance appears to have an adverse effect on classification performance. This is maybe due to the unreliable inverse matrix estimation on these datasets. On the contrary, the MIT dataset receives significant performance improvement using an inverse matrix. The improvement is possibly due to the better discriminative capability of resultant representation with the characterisation of partial corrections between feature components by an inverse matrix. iSICE improves the performance of inverse matrix on fine-grained datasets by applying sparsity. Another interesting observation is the performance of iSICE on MIT when the Bold HP variant is used.

If we have the most optimal iSICE hyper-parameters available, the performance on MIT can be further improved. The deterioration of iSICE performance with respect to its covariance counterparts on MIT with the Bold HP variant was due to the use of a lower learning rate (more discussion on this is provided in the following section). However, in other datasets, we did not experience the same. Overall, our method is robust to a large range of hyper-parameters (it will be demonstrated in the following experiments). We experimentally found that the performance of iSICE significantly drops across all datasets only when the learning rate becomes excessively large, i.e., 5.0, 10.0 and 20.0, and the sparsity constant becomes higher, i.e., 0.1, 0.5 and 1.0.

### 5.4.4 On the impact of changing hyper-parameters to the performance of iSICE

Based on hyper-parameters selected for the iSICE (Bold HP) variant, we perform the following experiments to demonstrate the robustness of our proposed method: 1) changing the sparsity constant, 2) changing the learning rate and 3) changing the number of iterations (i.e., for computing SICE).

**Changing the sparsity constant.** In this experiment, we change the sparsity constant while keeping the learning rate and the number of iterations fixed. Our experimental results are shown in Table 5.3. We present results with both VGG-16 and ResNet-50 CNN backbones. From the results, we can clearly see that across all datasets, the change of sparsity constants does not significantly impact the performance of iSICE. The VGG-16 based iSICE shows more robustness toward sparsity constant changes. The classification performance of the MIT dataset appears to have been more significantly impacted due to sparsity constant changes than the other three fine-grained datasets. Specifically, on fine-grained datasets, the standard deviation of results is significantly low, i.e., less than 0.51. This confirms that our method can be used for fine-grained image classification purposes with a reasonable sparsity constant.

**Changing the learning rate.** In this experiment, we change the learning rate while keeping the sparsity constant and the number of iterations fixed. Our experimental results are shown in Table 5.4. From the results, we can see that across all datasets, the change in learning rate does not significantly affect performance. Two datasets, namely Birds and Cars, have shown less impact on performance, as shown by the standard deviation of 0.15. The other two datasets also show a low standard deviation of results. The low standard deviation across a wide range of rates (0.001 to 20.0) shows that our method is robust to the changes in learning rate and a small learning rate such as 0.01 can be used for computing SICE with our method.

| Dataset | Backbone | COV | Inv. COV | iSICE | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Value of Sparsity constant | | | | | | | Mean ± Std. |
| | | | | 1.0 | 0.5 | 0.1 | 0.01 | 0.001 | 0.0001 | 0.00001 | |
| MIT | VGG-16 | 76.12 | 80.15 | 77.46 | 78.13 | 78.13 | 78.66 | 78.58 | 78.96 | 78.96 | 78.41±0.54 |
| | ResNet-50 | 78.81 | 80.75 | 78.43 | 80.75 | 80.45 | 80.52 | 80.90 | 80.37 | 81.34 | 80.39±0.93 |
| Airplane | VGG-16 | 90.01 | 89.44 | 92.26 | 92.71 | 92.77 | 92.23 | 92.83 | 92.74 | 92.44 | 92.56 ±0.25 |
| | ResNet-50 | 90.88 | 91.15 | 92.89 | 92.65 | 92.89 | 92.74 | 92.83 | 92.56 | 92.56 | 92.73±0.14 |
| Birds | VGG-16 | 84.47 | 83.36 | 86.04 | 86.47 | 86.35 | 86.52 | 85.59 | 86.31 | 86.28 | 86.22±0.32 |
| | ResNet-50 | 84.26 | 84.67 | 84.62 | 85.16 | 85.30 | 85.90 | 86.05 | 85.90 | 85.59 | 85.50±0.51 |
| Cars | VGG-16 | 91.21 | 92.04 | 93.60 | 93.98 | 94.06 | 94.03 | 93.88 | 93.91 | 93.50 | 93.85±0.22 |
| | ResNet-50 | 92.13 | 91.99 | 93.01 | 93.36 | 93.69 | 93.51 | 93.22 | 93.72 | 93.40 | 93.41±0.25 |

**Table 5.3:** Performance of iSICE with two CNN backbones on changing the sparsity constant $\lambda$ while fixing the learning rate $\eta$ and the number of iterations N to 1.0 and 5, respectively. The mean and standard deviation (std.) are also shown for a better understanding of $\lambda$ in our proposed method.

| Dataset | Backbone | COV | Inv. COV | iSICE | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Learning rate | | | | | | | Mean ± Std. |
| | | | | 0.001 | 0.01 | 0.1 | 1.0 | 5.0 | 10.0 | 20.0 | |
| MIT | VGG-16 | 76.12 | 80.15 | 78.81 | 79.33 | 77.91 | 78.66 | 78.28 | 80.52 | 77.76 | 78.75±0.95 |
| | ResNet-50 | 78.81 | 80.75 | 80.82 | 80.82 | 80.75 | 80.52 | 81.19 | 79.33 | 78.96 | 80.34±0.85 |
| Airplane | VGG-16 | 90.01 | 89.44 | 92.32 | 92.38 | 92.98 | 92.23 | 93.28 | 92.50 | 92.26 | 92.56±0.41 |
| | ResNet-50 | 90.88 | 91.15 | 92.77 | 93.01 | 92.89 | 92.74 | 92.65 | 92.95 | 92.38 | 92.77±0.21 |
| Birds | VGG-16 | 84.47 | 83.36 | 86.54 | 86.31 | 86.40 | 86.52 | 86.73 | 86.59 | 86.45 | 86.51±0.14 |
| | ResNet-50 | 84.26 | 84.67 | 85.69 | 85.88 | 85.81 | 85.90 | 85.59 | 85.71 | 85.97 | 85.79±0.14 |
| Cars | VGG-16 | 91.21 | 92.04 | 93.83 | 93.65 | 93.69 | 94.03 | 93.66 | 93.82 | 93.93 | 93.80±0.14 |
| | ResNet-50 | 92.13 | 91.99 | 93.60 | 93.57 | 93.32 | 93.51 | 93.57 | 93.35 | 93.60 | 93.50±0.12 |

**Table 5.4:** Performance of iSICE with two CNN backbones on changing the learning rate $\eta$ while fixing the sparsity constant $\eta$ and number of iterations N to 0.01 and 5, respectively.

| Method | No. of iterations | Based on VGG-16 backbone | | | |
|---|---|---|---|---|---|
| | | MIT | Airplane | Birds | Cars |
| COV | – | 76.12 | 90.01 | 84.47 | 91.21 |
| Inv. COV | – | 80.15 | 89.44 | 83.36 | 92.04 |
| iSICE | 2 | 78.28 | 92.68 | 86.66 | 93.63 |
| | 5 | 78.66 | 92.23 | 86.52 | 94.03 |
| | 10 | 78.36 | 92.56 | 86.62 | 93.89 |
| | Mean ± Std. | 78.43±0.20 | 92.49±0.23 | 86.60±0.07 | 93.85±0.20 |
| Method | No. of iterations | Based on ResNet-50 backbone | | | |
| | | MIT | Airplane | Birds | Cars |
| COV | – | 78.81 | 90.88 | 84.26 | 92.13 |
| Inv. COV | – | 80.75 | 91.15 | 84.67 | 91.99 |
| iSICE | 2 | 80.52 | 92.89 | 93.68 | 85.92 |
| | 5 | 80.52 | 92.74 | 93.51 | 85.90 |
| | 10 | 80.22 | 92.74 | 93.30 | 85.74 |
| | Mean ± Std. | 80.42±0.17 | 92.79±0.09 | 93.50 ±0.19 | 85.85 ± 0.10 |

**Table 5.5:** Performance of iSICE with two CNN backbones on changing the learning rate $\eta$ while fixing the sparsity constant $\eta$ and number of iterations N to 0.01 and 5, respectively.

**Changing the number of iterations.** In this experiment, we change the number of iterations while keeping the sparsity constant and the learning rate fixed. Our experimental results are shown in Table 5.5. From the results, we can see that regardless of the CNN backbones used, the change in the number of iterations can vary the performance to 0.20%. It is also noticeable that our method is able to give good performance with only two iterations. This experiment shows that our method is robust to the changes in the number of iterations.

### 5.4.5 Comparison with the state-of-the-art SPD representation methods

In this section, we compare our proposed method with existing state-of-the-art SPD representation methods and the proposed methods in the previous two chapters. Specifically, we compare with fifteen state-of-the-art methods, namely, VGG-16 [18], NetVLAD [91], NetFV [92], BCNN [1], CBP [2], LRBP [28], KP [25], HIHCA [93], Improved BCNN

| Methods | Backbone | Airplane | Birds | Cars |
|---|---|---|---|---|
| VGG-16 [18] | VGG-16 | 76.6 | 70.4 | 79.8 |
| NetVLAD [91] | | 81.8 | 81.6 | 88.6 |
| NetFV [92] | | 79.0 | 79.9 | 86.2 |
| BCNN [1] | | 83.9 | 84.0 | 90.6 |
| CBP [2] | | 84.1 | 84.3 | 91.2 |
| LRBP [28] | | 87.3 | 84.2 | 90.9 |
| KP [25] | | 86.9 | 86.2 | 92.4 |
| HIHCA [93] | | 88.3 | 85.3 | 91.7 |
| Improved BCNN [3] | | 88.5 | 85.8 | 92.0 |
| SMSO [112] | | – | 85.0 | – |
| MPN-COV [29] | | 89.9 | 86.7 | 92.2 |
| G$^2$DeNet [29] | | 89.0 | **87.1** | 92.5 |
| iSQRT-COV [5] (reproduced)$^\dagger$ | | 90.0 | 84.5 | 91.2 |
| DeepCOV [6] | | 88.7 | 85.4 | 91.7 |
| DeepKSPD [6] | | 90.0 | 84.8 | 91.6 |
| DeepCOV+ ReDro (Chapter 3) | | 89.2 | 86.7 | 92.2 |
| FastCOV-VGG (Chapter 4) | | 90.9 | 86.0 | 90.8 |
| Inverse COV (Alg. 3) | | 89.4 | 83.4 | 92.0 |
| iSICE (Bold HP) | | 92.2 | 86.5 | 94.0 |
| iSICE (Best HP) | | **93.3** | 86.9 | **94.1** |
| CBP [2] | ResNet-50 | 81.6 | 81.6 | 88.6 |
| KP [25] | | 85.7 | 84.7 | 91.1 |
| SMSO [112] | | – | 85.8 | – |
| iSQRT-COV [5] | | 89.5 | **87.3** | 91.7 |
| iSQRT-COV [5] (reproduced)$^\dagger$ | | 90.9 | 84.3 | 92.1 |
| DeepCOV-ResNet | | 87.7 | 86.7 | 91.6 |
| DeepCOV-ResNet+ ReDro (Chapter 3) | | 86.8 | 86.9 | 92.0 |
| FastCOV-ResNet (Chapter 4) | | 90.1 | 86.5 | 92.8 |
| Inverse COV (Alg. 3) | | 91.2 | 84.7 | 92.0 |
| iSICE (Bold HP) | | 92.7 | 85.9 | 93.5 |
| iSICE (Best HP) | | **93.6** | 86.6 | **93.9** |

**Table 5.6:** Comparison between the proposed methods and other SPD representation methods in terms of classification accuracy (%). The performance of existing SPD representation methods is quoted from the original papers. The best methods across datasets and CNN backbones are marked in bold. $^\dagger$With the same hyper-parameter settings, CNN backbone, and default iSQRT-COV iterations of 5 times.

[3], SMSO [112], MPN-COV [29], G$^2$DeNet [29], iSQRT-COV [5], DeepCOV [6], and DeepKSPD [6]. We only compare these methods on fine-grained image datasets since a majority of these methods perform evaluations on only fine-grained image datasets. For fairness, we divide them into two groups based on which CNN backbone they have used, i.e., VGG-16 or ResNet-50.

We present our comparison in Table 5.6. The top part of the table shows VGG-16 based methods. It is clear to see that both our iSICE (Bold HP) and iSICE (Best HP) methods significantly outperformed other methods on Airplane and Cars datasets. For the Birds dataset, the performance of our methods is lower than MPN-COV and G$^2$DeNet. These two methods use CNN backbones pre-trained with second-order pooling and we use CNN backbones pre-trained with first-order pooling. Other methods use same backbone as ours and we achieve better performance than them. Overall, on Birds dataset, our methods rank second among others. The bottom part of the Table shows ResNet-50 based methods. Here also we can see that our iSICE (Bold HP) and iSICE (Best HP) methods give better performance on Airplane and Cars datasets. On the Birds dataset, we receive slightly poor performance than the original iSQRT-COV method. As mentioned above, the original iSQRT-COV method is based on a CNN model pre-trained with second-order pooling. We re-train iSQRT-COV with the backbone we used in our methods which is pre-trained with first-order pooling and find that it performs inferior to our methods on the Birds dataset. Our method may further improve with second-order pooling based on pre-trained backbones.

### 5.4.6 Ablation study on the size of sparse inverse matrix

In the above experiments, we use the $256 \times 256$-dimensional iSICE matrix. In Table 5.7, we show our results when the dimension of the iSICE matrix becomes double, i.e., $512 \times 512$ while keeping the hyper-parameter setting the same, and compare them to its covariance counterparts. It is interesting to see that except in a few cases, switching to a higher dimensional matrix improves the performance. The improvement is significant for the MIT dataset in the case of our iSICE. This indicates that in the case of the MIT dataset, it is better to apply sparsity to a larger inverse covariance for obtaining better performance.

On average, both VGG-16 and ResNet-50 based iSICE and its covariance counterparts have received performance improvement by switching to a higher dimensional matrix. Overall, the improvement is more significant with the VGG-16 backbone. It is also noticeable that iSICE performs better with VGG-16. However, its covariance counterparts perform better with ResNet-50 than VGG-16. It is important to mention that switching to a larger iSICE significantly increases the computation cost.

| Dataset | Backbone | COV Dim. | | Inv. COV Dim. | | iSICE (Bold HP) Dim. | |
|---|---|---|---|---|---|---|---|
| | | 256×256 | 512×512 | 256×256 | 512×512 | 256×256 | 512×512 |
| MIT | VGG-16 | 76.12 | **76.87** | 80.15 | **80.67** | 78.66 | **81.12** |
| | ResNet-50 | 78.81 | **82.76** | 80.75 | **82.69** | 80.52 | **81.72** |
| Airplane | VGG-16 | 90.01 | **91.45** | 89.44 | **90.10** | 92.23 | **92.86** |
| | ResNet-50 | 90.88 | **91.15** | 91.15 | **91.48** | **92.74** | 92.59 |
| Birds | VGG-16 | 84.47 | **85.02** | 83.36 | **84.90** | 86.52 | **86.83** |
| | ResNet-50 | 84.26 | **84.48** | **84.67** | 83.98 | 85.90 | **85.95** |
| Cars | VGG-16 | 91.21 | **92.18** | 92.04 | **92.54** | 94.03 | **94.62** |
| | ResNet-50 | **92.13** | 92.10 | 91.99 | **92.56** | 93.51 | **93.81** |
| Average | VGG-16 | 85.45 | **86.38** | 86.25 | **87.05** | 87.86 | **88.86** |
| | ResNet-50 | 86.52 | **87.62** | 87.14 | **87.68** | 88.17 | **88.52** |

**Table 5.7:** Performance of covariance (COV), inverse of covariance (Inv. COV) and iSICE (Bold HP) on various datasets when different matrix dimensions (dim.) are used.

## 5.5 Conclusion

In this paper, we propose a method for learning sparse inverse covariance representation with CNN. Our method estimates SICE within the CNN layers and facilitates backpropagation for end-to-end training. We show that our proposed method significantly outperforms its covariance counterparts and state-of-the-art covariance representation methods on several datasets. Our method is of general purpose and can be readily applied to existing SPD representation methods to improve their performance. In the future, we plan to extend our method to other vision tasks such as image retrieval, image segmentation and object tracking.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusion

This thesis focuses on developing CNN based robust and efficient SPD visual representation learning methods. Due to the small sized feature maps (i.e., samples) and the large number of channels (i.e., dimensionality) at the last convolutional layer, the SPD matrix computed from CNN often suffers from the matrix swelling issue and becomes unreliable. In the literature, matrix normalisation is used for combating this issue which requires eigen-decomposition operation. However, the computation cost of eigen-decomposition increases significantly as the size of the covariance matrix increases. This makes it difficult to learn large-sized SPD representation. This thesis contributes to the existing efforts of applying matrix normalisation for obtaining reliable SPD matrix based visual representation. It proposes two novel methods to make the matrix normalisation step more efficient for large-sized SPD matrices. In addition to that, it proposes sparse inverse covariance estimation (SICE) for more reliable SPD matrix estimation and presents a novel method for integrating the SICE process into the CNN.

The first method is ReDro which performs eigen-decomposition on block-diagonal SPD matrices instead of regular full-sized SPD matrix to reduce the computation time of matrix normalisation. It is based on the fact that the computational cost of performing eigen-decomposition on a block-diagonal SPD matrix is less than the eigen-decomposition of a full SPD matrix. In the experimental study, it has been shown that the benefits of ReDro are more significant when the size of SPD matrices becomes large. Apart from saving the computation time, it also helps to reduce the bias issue in SPD matrix estimation. The block matrices are estimated for a small number of the components of the descriptors, therefore, their eigenvalues become less biased. Also, ReDro can be seen as a dropout-like regularisation for SPD representation which helps CNNs to achieve better classification performance.

The second method is FastCOV which exploits the intrinsic connection between eigen systems of $\mathbf{XX}^\top$ and $\mathbf{X}^\top\mathbf{X}$ to reduce the computation time by applying matrix normalisation on a position-wise covariance matrix. In CNNs that have smaller feature maps such as ResNets [12], the position-wise covariance matrix becomes small and the eigendecomposition time of the small covariance matrix is low. The advanced CNNs such as ResNet and DenseNet [20] have a higher number of channels with small feature maps in the last convolutional layers. The proposed FastCOV method can efficiently compute covariance representation from those CNNs than the existing methods. In contrast, the first method ReDro is more advantageous when the spatial size of feature maps becomes large since it does not take the spatial size of feature maps into the account.

The third method is iSICE which computes end-to-end sparse inverse covariance representation from CNN feature maps. It applies sparsity on the prior structure of covariance to estimate sparse inverse covariance. Based on our survey, sparse inverse covariance representation has not been previously learned in an end-to-end manner in the previous works. The direct (i.e., partial) relationships between feature components characterised by sparse inverse covariance have significantly improved the covariance estimation in terms of its reliability and discriminative capacity, as evidenced by the experimental results. Furthermore, iSICE is robust with respect to its hyper-parameters. This method validates that exploiting prior information can effectively improve the covariance representation and lead to competitive performance with the existing SPD representation methods.

## 6.2 Future Work

This thesis proposes methods for efficient learning of SPD matrix based visual representation with CNNs. While the proposed methods achieve competitive performance with state-of-the-art, there are still many important aspects of SPD matrix based visual representation that require further investigation. One of the key aspects is the better understanding of global representation modelled from CNN descriptors by SPD visual representation and how it is different from other representation methods. Another key aspect is how to learn optimal or effective SPD representation from visual descriptors. iSICE method proposed in Chapter 5 of this thesis leverages prior knowledge to learn effective SPD visual representation. In the literature, there are also efforts on SPD matrix optimisation on manifold and kernel learning for effective SPD visual representation learning. Finally, the computation aspect of SPD matrix based representation requires more investigations. Existing SPD visual representations have higher dimensionality and require high computing time. More investigations are required to design compact and effective SPD representation. It is also worth mentioning that SPD representation captures second-order information, however, even higher-order information can be captured with rectangular matrix based representation methods for obtaining better performance.

The following extension plan can be used to improve the proposed methods. The first method ReDro can be extended as follows. Conceptually, ReDro can be applied to very high dimensional SPD matrices to reduce their normalisation or processing time. In our experiments, we consider ReDro with medium sized matrices. However, it can be used in cases where the computation of a large matrix from a big data corpus is needed but cannot be performed due to storage limitations. The ReDro idea can effectively solve this issue by splitting data into groups and computing separate matrices from each group and processing them independently. However, deep learning of those matrices requires further research investigation. Furthermore, the channel partitioning of the CNN feature map in ReDro does not consider any semantic information such as parts related to objects in the image. The semantic information based grouping may further improve the discriminativeness of the SPD representation produced by ReDro. The semantic information can be pre-generated or produced on the fly with an appropriate method. Additionally, the attention mechanism during the block-diagonal matrix construction can also be incorporated to further improve the SPD representation.

The second method FastCOV can be extended as follows. Theoretically, the concepts of FastCOV are based on eigensystems of $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$, and applicable to only covariance representation. Such connections may also exist for other types of SPD matrices such as kernel matrices and may be leveraged to perform efficient matrix normalisation of those matrices. In FastCOV, we used eigen-decomposition based matrix normalisation. However, by principle, we can also use iterative matrix normalisation such as Newton-Schulz iteration [5] for better GPU support.

The third method iSICE can be extended as follows. Currently, hyper-parameter selection in iSICE is done empirically. However, they can be jointly learned with iSICE by a simple neural network based on single or multiple perceptrons. The perceptron based network can be added as a branch network to the CNN and it will receive its input from the last convolution layer. The output of the perceptron based network will be transformed with a simple squashing function to ensure its meaningful learning. In our work, we applied iSICE only to image classification problems. However, iSICE can have more applications including graph models, skeleton models, etc., where domain knowledge can improve the task performance.

This chapter summarises the key contributions of this thesis. Besides these contributions, a literature review covering the existing progress of SPD representation is also provided. The future extensions of the proposed methods are also discussed.

# Bibliography

[1] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1449–1457.

[2] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell, "Compact bilinear pooling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 317–326.

[3] T.-Y. Lin and S. Maji, "Improved bilinear pooling with cnns," *arXiv preprint arXiv:1707.06772*, 2017.

[4] P. Li, J. Xie, Q. Wang, and W. Zuo, "Is second-order information helpful for large-scale visual recognition?" In *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2070–2078.

[5] P. Li, J. Xie, Q. Wang, and Z. Gao, "Towards faster training of global covariance pooling networks by iterative matrix square root normalization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 947–955.

[6] M. Engin, L. Wang, L. Zhou, and X. Liu, "Deepkspd: Learning kernel-matrix-based spd representation for fine-grained image recognition," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 612–627.

[7] T. Yu, X. Li, and P. Li, "Fast and compact bilinear pooling by shifted random maclaurin," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 3243–3251.

[8] O. Tuzel, F. Porikli, and P. Meer, "Region covariance: A fast descriptor for detection and classification," in *Proceedings of the European Conference on Computer Vision*, Springer, 2006, pp. 589–600.

[9] K. Guo, P. Ishwar, and J. Konrad, "Action recognition from video using feature covariance matrices," *IEEE Transactions on Image Processing*, vol. 22, no. 6, pp. 2479–2494, 2013.

[10] O. Tuzel, F. Porikli, and P. Meer, "Human detection via classification on Riemannian manifolds," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE, 2007, pp. 1–8.

[11] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Matrix backpropagation for deep networks with structured layers," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2965–2973.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[13] J. Zhang, L. Wang, L. Zhou, and W. Li, "Beyond covariance: Sice and kernel based visual feature representation," *International Journal of Computer Vision*, vol. 129, no. 2, pp. 300–320, 2021.

[14] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International Journal of Computer Vision*, vol. 105, no. 3, pp. 222–245, 2013.

[15] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2010, pp. 3304–3311.

[16] H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," *Advances in Neural Information Processing Systems*, vol. 19, 2006.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[20] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[21] M. Cimpoi, S. Maji, and A. Vedaldi, "Deep filter banks for texture recognition and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3828–3836.

[22] Q. Wang, P. Li, W. Zuo, and L. Zhang, “Raid-g: Robust estimation of approximate infinite dimensional gaussian with application to material recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4433–4441.

[23] O. Ledoit and M. Wolf, “A well-conditioned estimator for large-dimensional covariance matrices,” *Journal of multivariate analysis*, vol. 88, no. 2, pp. 365–411, 2004.

[24] T. Yu, Y. Cai, and P. Li, “Toward faster and simpler matrix normalization via rank-1 update,” in *European Conference on Computer Vision*, Springer, 2020, pp. 203–219.

[25] Y. Cui, F. Zhou, J. Wang, X. Liu, Y. Lin, and S. Belongie, “Kernel pooling for convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2921–2930.

[26] N. Pham and R. Pagh, “Fast and scalable polynomial kernels via explicit feature maps,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 239–247.

[27] P. Kar and H. Karnick, “Random feature maps for dot product kernels,” in *Artificial intelligence and statistics*, PMLR, 2012, pp. 583–591.

[28] S. Kong and C. Fowlkes, “Low-rank bilinear pooling for fine-grained classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 365–374.

[29] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li, “Deep cnns meet global covariance pooling: Better representation and generalization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 8, pp. 2582–2597, 2020.

[30] Q. Wang, P. Li, and L. Zhang, “G2denet: Global gaussian distribution embedding network and its application to visual recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2730–2739.

[31] C. Yu, X. Zhao, Q. Zheng, P. Zhang, and X. You, “Hierarchical bilinear pooling for fine-grained visual recognition,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 574–589.

[32] Z. Gao, J. Xie, Q. Wang, and P. Li, “Global second-order pooling convolutional networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3024–3033.

[33] S. Min, H. Yao, H. Xie, Z.-J. Zha, and Y. Zhang, “Multi-objective matrix normalization for fine-grained visual recognition,” *IEEE Transactions on Image Processing*, vol. 29, pp. 4996–5009, 2020.

[34] Y. Song, N. Sebe, and W. Wang, “Why approximate matrix square root outperforms accurate svd in global covariance pooling?” In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1115–1123.

[35] Q. Liao, D. Wang, H. Holewa, and M. Xu, “Squeezed bilinear pooling for fine-grained visual categorization,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[36] X. Li, C. Yang, S.-L. Chen, C. Zhu, and X.-C. Yin, “Semantic bilinear pooling for fine-grained recognition,” in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 3660–3666.

[37] Q. Wang, K. Zhang, J. Fan, S. Huang, and L. Zhang, “Multi-order feature statistical model for fine-grained visual categorization,” in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 7379–7386.

[38] D. López-Sánchez, A. G. Arrieta, and J. M. Corchado, “Compact bilinear pooling via kernelized random projection for fine-grained image categorization on low computational power devices,” *Neurocomputing*, vol. 398, pp. 411–421, 2020.

[39] Z. Gao, Y. Wu, X. Zhang, J. Dai, Y. Jia, and M. Harandi, “Revisiting bilinear pooling: A coding perspective,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3954–3961.

[40] M. Tan, F. Yuan, J. Yu, G. Wang, and X. Gu, “Fine-grained image classification via multi-scale selective hierarchical biquadratic pooling,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 18, no. 1s, pp. 1–23, 2022.

[41] S. Min, H. Xie, Y. Tian, H. Yao, and Y. Zhang, “Adaptive bilinear pooling for fine-grained representation learning,” in *Proceedings of the ACM Multimedia Asia*, 2019, pp. 1–6.

[42] F. Li, Q. Xu, Z. Sun, Y. Mei, Q. Zhang, and B. Luo, “Multi-layer weight-aware bilinear pooling for fine-grained image classification,” in *Advances in Brain Inspired Cognitive Systems: 10th International Conference, BICS 2019, Guangzhou, China, July 13–14, 2019, Proceedings 10*, Springer, 2020, pp. 443–453.

[43] Y. Liu, H. Gu, C. Li, Q. Xu, and L. Yang, “Hybrid-order and multi-stream convolutional neural network for fine-grained visual recognition,” in *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, IEEE, 2019, pp. 1–6.

[44] Y. Song, N. Sebe, and W. Wang, “Fast differentiable matrix square root,” *arXiv preprint arXiv:2201.08663*, 2022.

[45] W. Wang, Z. Dang, Y. Hu, P. Fua, and M. Salzmann, "Robust differentiable svd," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5472–5487, 2021.

[46] H. Zhang and P. Koniusz, "Power normalizing second-order similarity network for few-shot learning," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2019, pp. 1185–1193.

[47] H. Zhang, J. Zhang, and P. Koniusz, "Few-shot learning via saliency-guided hallucination of samples," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE, 2019, pp. 2770–2779.

[48] H. Zhang, P. H. Torr, and P. Koniusz, "Few-shot learning with multi-scale self-supervision," *arXiv preprint arXiv:2001.01600*, 2020.

[49] H. Huang, J. Zhang, J. Zhang, J. Xu, and Q. Wu, "Low-rank pairwise alignment bilinear network for few-shot fine-grained image classification," *IEEE Transactions on Multimedia*, vol. 23, pp. 1666–1680, 2020.

[50] H. Zhang, H. Li, and P. Koniusz, "Multi-level second-order few-shot learning," *IEEE Transactions on Multimedia*, 2022.

[51] C. Simon, P. Koniusz, R. Nock, and M. Harandi, "Adaptive subspaces for few-shot learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4136–4145.

[52] H. Zhang, P. H. Torr, and P. Koniusz, "Improving few-shot learning by spatially-aware matching and crosstransformer," in *Proceedings of the Asian Conference on Computer Vision*, 2022, pp. 1298–1315.

[53] L. Wang and P. Koniusz, "Self-supervising action recognition by statistical moment and subspace descriptors," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 4324–4333.

[54] P. Koniusz, L. Wang, and A. Cherian, "Tensor representations for action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[55] P. Koniusz, L. Wang, and K. Sun, "High-order tensor pooling with attention for action recognition," *arXiv preprint arXiv:2110.05216*, 2021.

[56] W. Xiang, H. Yong, J. Huang, X.-S. Hua, and L. Zhang, "Second-order camera-aware color transformation for cross-domain person re-identification," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[57] P. Wang, Z. Zhao, F. Su, X. Zu, and N. V. Boulgouris, "Horeid: Deep high-order mapping enhances pose alignment for person re-identification," *IEEE Transactions on Image Processing*, vol. 30, pp. 2908–2922, 2021.

[58] P. Koniusz, Y. Tas, and F. Porikli, “Domain adaptation by mixture of alignments of second-or higher-order scatter tensors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4478–4487.

[59] B. Chen, W. Deng, and J. Hu, “Mixed high-order attention network for person re-identification,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 371–381.

[60] T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang, “Second-order attention network for single image super-resolution,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 065–11 074.

[61] B. N. Xia, Y. Gong, Y. Zhang, and C. Poellabauer, “Second-order non-local attention networks for person re-identification,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3760–3769.

[62] T. Ng, V. Balntas, Y. Tian, and K. Mikolajczyk, “Solar: Second-order loss and attention for image retrieval,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*, Springer, 2020, pp. 253–270.

[63] P. Fang, J. Zhou, S. K. Roy, L. Petersson, and M. Harandi, “Bilinear attention networks for person retrieval,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 8030–8039.

[64] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, “Multimodal compact bilinear pooling for visual question answering and visual grounding,” *arXiv preprint arXiv:1606.01847*, 2016.

[65] S. Jayasumana, R. Hartley, M. Salzmann, H. Li, and M. Harandi, “Kernel methods on riemannian manifolds with gaussian rbf kernels,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 12, pp. 2464–2477, 2015.

[66] R. Wang, H. Guo, L. S. Davis, and Q. Dai, “Covariance discriminative learning: A natural and efficient approach to image set classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2012, pp. 2496–2503.

[67] X. Zhu, C. Xu, L. Hui, C. Lu, and D. Tao, “Approximated Bilinear Modules for Temporal Modeling,” in *Proceedings of the International Conference on Computer Vision*, IEEE, 2019, pp. 3494–3503.

[68] P. Koniusz, L. Wang, and A. Cherian, “Tensor representations for action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[69] H. Zhang, L. Zhang, X. Qui, H. Li, P. H. S. Torr, and P. Koniusz, “Few-shot action recognition with permutation-invariant attention,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[70] P. Koniusz, H. Zhang, and F. Porikli, “A deeper look at power normalizations,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, 2018, pp. 5774–5783.

[71] P. Koniusz and H. Zhang, “Power normalizations in fine-grained image, few-shot image and graph classification,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[72] A. Y. Ng, M. I. Jordan, and Y. Weiss, “On spectral clustering: Analysis and an algorithm,” in *Proceedings of the Advances in Neural Information Processing Systems*, 2002, pp. 849–856.

[73] H. Zheng, J. Fu, Z.-J. Zha, and J. Luo, “Learning deep bilinear transformation for fine-grained image representation,” in *Proceedings of the Advances in Neural Information Processing Systems*, 2019, pp. 4279–4288.

[74] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[75] F. Porikli, O. Tuzel, and P. Meer, “Covariance tracking using model update based on Lie algebra,” in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE, vol. 1, 2006, pp. 728–735.

[76] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord, “Riemannian batch normalization for SPD neural networks,” in *Advances in Neural Information Processing Systems*, 2019, pp. 15 489–15 500.

[77] Z. Huang and L. Van Gool, “A Riemannian network for SPD matrix learning,” in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[78] X. S. Nguyen, L. Brun, O. Lézoray, and S. Bougleux, “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*, 2019, pp. 12 036–12 045.

[79] K. Yu and M. Salzmann, “Second-order convolutional neural networks,” *arXiv preprint arXiv:1703.06817*, 2017.

[80] R. Chakraborty, J. Bouza, J. Manton, and B. C. Vemuri, “A deep neural network for manifold-valued data with applications to neuroimaging,” in *International Conference on Information Processing in Medical Imaging*, Springer, 2019, pp. 112–124.

[81] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE, 2015, pp. 648–656.

[82] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Dropblock: A regularization method for convolutional networks," in *Proceedings of the Advances in Neural Information Processing Systems*, 2018, pp. 10 727–10 737.

[83] S. Hou and Z. Wang, "Weighted channel dropout for regularization of deep convolutional neural network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 8425–8432.

[84] J. Demmel, I. Dumitriu, and O. Holtz, "Fast linear algebra is stable," *Numerische Mathematik*, vol. 108, no. 1, pp. 59–91, 2007.

[85] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, IEEE, 2009, pp. 413–420.

[86] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "CalTech-UCSD Birds 200," California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.

[87] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," *arXiv preprint arXiv:1306.5151*, 2013.

[88] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D object representations for fine-grained categorization," in *Proceedings of the International Conference on Computer Vision Workshops*, IEEE, 2013, pp. 554–561.

[89] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional Neural Networks for MATLAB," in *Proceedings of the 23rd ACM International Conference on Multimedia*, 2015, pp. 689–692.

[90] S. Rahman, L. Wang, C. Sun, and L. Zhou, "ReDro: Efficiently Learning Large-sized SPD Visual Representation," in *European Conference on Computer Vision*, Springer, 2020, pp. 1–17.

[91] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5297–5307.

[92] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear convolutional neural networks for fine-grained visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1309–1322, 2017.

[93] S. Cai, W. Zuo, and L. Zhang, "Higher-order integration of hierarchical convolutional activations for fine-grained visual categorization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 511–520.

[94] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[95] P. Koniusz, F. Yan, P.-H. Gosselin, and K. Mikolajczyk, "Higher-order occurrence pooling for bags-of-words: Visual concept detection," *PAMI*, vol. 39, no. 2, pp. 313–326, 2017.

[96] P. Koniusz, A. Cherian, and F. Porikli, "Tensor representations via kernel linearization for action recognition from 3d skeletons," in *ECCV*, Springer, 2016, pp. 37–53.

[97] Y. Li, N. Wang, J. Liu, and X. Hou, "Factorized bilinear models for image recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2079–2087.

[98] T. Yu, Y. Cai, and P. Li, "Efficient compact bilinear pooling via kronecker product," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 3170–3178.

[99] W. Luo, H. Zhang, J. Li, and X.-S. Wei, "Learning semantically enhanced feature for fine-grained image classification," *IEEE Signal Processing Letters*, vol. 27, pp. 1545–1549, 2020.

[100] R. Zeng and J. He, "Grouping bilinear pooling," in *32nd British Machine Vision Conference 2021, BMVC 2021, Online, November 22-25, 2021*, BMVA Press, 2021, p. 183. [Online]. Available: `https://www.bmvc2021-virtualconference.com/assets/papers/0876.pdf`.

[101] ——, "Grouping bilinear pooling for fine-grained image classification," *Applied Sciences*, vol. 12, no. 10, p. 5063, 2022.

[102] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 976–11 986.

[103] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[104] N. J. Higham, *Functions of matrices: theory and computation*. SIAM, 2008.

[105] P. Koniusz, H. Zhang, and F. Porikli, "A deeper look at power normalizations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5774–5783.

[106] T.-Y. Lin, S. Maji, and P. Koniusz, “Second-order democratic aggregation,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 620–636.

[107] J. Huang, T. Zhang, and D. Metaxas, “Learning with structured sparsity.,” *Journal of Machine Learning Research*, vol. 12, no. 11, 2011.

[108] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[109] J. Friedman, T. Hastie, and R. Tibshirani, “Sparse inverse covariance estimation with the graphical lasso,” *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.

[110] S. Diamond and S. Boyd, “Cvxpy: A python-embedded modeling language for convex optimization,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2909–2913, 2016.

[111] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.

[112] K. Yu and M. Salzmann, “Statistically-motivated second-order pooling,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 600–616.

# Appendix A

# Implementation Details and Computational Advantage of ReDro

## A.1 Implementation of ReDro

In this section, we provide the implementation steps of ReDro mentioned in Section of the Chapter 3. Following the Algorithm 1 in the Chapter 3, we implement ReDro in four phases:

i. *Channel permutation.* Given feature channels from the last convolutional layer of the backbone CNN model and a permutation matrix randomly generated at a run-time, we perform permutation of the feature channels with the permutation matrix.

ii. *Block-diagonal matrix computation.* The permuted feature channels in phase 1 are partitioned into $k$ equally sized groups and on each group, a small covariance matrix is computed. Next, eigen-decomposition is computed for each of the small covariance matrices, and the resultant eigenvalues and eigenvectors are assembled in a block-diagonal manner.

iii. *Back-permutation of eigenvectors.* Using the permutation matrix from phase 1, we then permute back the eigenvectors obtained in phase 2 to make the random permutation in ReDro transparent to the subsequent network layers.

iv. *Matrix normalisation.* Given the eigenvalues and eigenvectors from phases 2 and 3, we then apply the matrix normalisation.

Depending on the SPD visual representation methods (used in Section 3.4.1 of 3) and the availability of source code, we implement the above four phases by MatConvNet [89]. *Our source code containing the layer implementations, experimental frameworks and dataset protocols will be available online.*

## A.2 Computational advantage of ReDro

In this section, we provide the forward and backward propagation time of the methods compared in Table 3.1 of the Chapter 3. Table A.1 in this supplement material shows the forward propagation time (indicated with “F.P.”) and backward propagation time (indicated with “B.P.”) alongside with the total (i.e., forward propagation+backward propagation) time (indicated with “Total”).

From the table, in addition to the discussions provided in the Section 3.4.1 of Chapter 3, we can observe that ReDro saves the computation time of forward propagation in learning large-sized covariance matrices, i.e., of the size of $512\times512$ and $1024\times1024$. Meanwhile, note that as expected, ReDro will not save any computation time for backward propagation since it is only applied to deal with the eigen-decomposition in the forward propagation.

| SPD Matrix Dimensions | Mode | No ReDro used | | | | | DeepCOV [6] using ReDro | | | | DeepKSPD [6] using ReDro | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Deep -COV [6] | Deep -KSPD [6] | MPN -COV [4] | IBCNN [3] | iSQRT -COV [5] | with $k = 2$ | with $k = 4$ | with $k = 8$ | with $k = 16$ | with $k = 2$ | with $k = 4$ | with $k = 8$ | with $k = 16$ |
| 128×128 | Total | 0.004 | 0.008 | 0.004 | 0.006 | 0.001 | 0.007 | 0.009 | 0.011 | 0.015 | 0.010 | 0.012 | 0.018 | 0.031 |
| | F.P. | 0.003 | 0.0035 | 0.003 | 0.003 | 0.0006 | 0.004 | 0.006 | 0.008 | 0.012 | 0.005 | 0.008 | 0.014 | 0.027 |
| | B.P. | 0.001 | 0.0041 | 0.001 | 0.003 | 0.0006 | 0.003 | 0.003 | 0.003 | 0.003 | 0.004 | 0.004 | 0.004 | 0.004 |
| 256×256 | Total | 0.013 | 0.016 | 0.013 | 0.014 | 0.006 | 0.011 | 0.011 | 0.013 | 0.020 | **0.015** | 0.016 | 0.022 | 0.037 |
| | F.P. | 0.011 | 0.011 | 0.011 | 0.011 | 0.0053 | 0.007 | 0.007 | 0.009 | 0.016 | 0.010 | 0.011 | 0.016 | 0.031 |
| | B.P. | 0.002 | 0.005 | 0.002 | 0.003 | 0.0012 | 0.004 | 0.004 | 0.004 | 0.004 | 0.005 | 0.005 | 0.005 | 0.005 |
| 512×512 | Total | 0.031 | 0.045 | 0.031 | 0.032 | 0.030 | 0.030 | **0.022** | **0.023** | 0.030 | **0.031** | **0.029** | **0.035** | 0.054 |
| | F.P. | 0.025 | 0.034 | 0.025 | 0.025 | 0.0270 | 0.023 | **0.015** | **0.016** | 0.023 | **0.020** | **0.018** | **0.024** | 0.043 |
| | B.P. | 0.006 | 0.011 | 0.006 | 0.007 | 0.0033 | 0.007 | 0.007 | 0.007 | 0.007 | 0.011 | 0.011 | 0.011 | 0.011 |
| 1024×1024 | Total | 0.097 | 0.189 | 0.097 | 0.121 | 0.097 | **0.090** | **0.076** | **0.056** | **0.062** | **0.087** | **0.084** | **0.082** | **0.094** |
| | F.P. | 0.089 | 0.157 | 0.089 | 0.111 | 0.0872 | **0.072** | **0.058** | **0.039** | **0.045** | **0.058** | **0.054** | **0.053** | **0.065** |
| | B.P. | 0.008 | 0.032 | 0.008 | 0.010 | 0.0095 | 0.018 | 0.018 | 0.017 | 0.017 | 0.030 | 0.030 | 0.030 | 0.030 |

**Table A.1:** Comparison of computational time (in second) for SPD matrix estimation and matrix normalisation by using or not using the proposed ReDro scheme. The forward propagation time (F.P.), backward propagation time (B.P.) and the sum of forward and backward propagation time (Total) are reported. The five methods to the left represent the case not using ReDro. The case using ReDro is implemented upon DeepCOV [6] and DeepKSPD [6] methods with various $k$. The boldface shows that ReDro saves computational time of baseline.

# Appendix B

# Implementation Details and Computational Advantage of FastCOV

## B.1 Implementation of FastCOV

In this section, we provide the implementation steps of FastCOV mentioned in Sec. 4.4. Following the Algorithm 2 in chapter 4, we implement FastCOV in three phases:

i. *Postion-wise covariance matrix computation.* Given feature channels from the last convolutional layer of the backbone CNN mode, we reshape it to a data matrix whose rows contain feature channels. Using the data matrix, we compute a position-wise covariance matrix.

ii. *Normalisation of postion-wise covariance matrix.* Matrix normalisation is applied on the postion-wise covariance matrix computed in phase 1.

iii. *Restoration of normalised channel-wise covariance matrix.* By multiplying data matrix from phase 1 to the left and right of the normalised postion-wise covariance matrix obtained in phase 2, we recover the normalised channel-wise covariance matrix.

We implement the above three phases using MatConvNet [89]. *Our source code containing the layer implementations, experimental frameworks, and dataset protocols will be available online.*

## B.2 Computational advantage of FastCOV

In this section, we discuss the reduction of training time by the proposed FastCOV method for learning large-sized covariance matrix representation mentioned in Sec. 4.4.1 of the chapter 4. To show the computational savings of FastCOV, we performed the following

| Methods | Backbone | Training time/epoch | Processing time/image |
|---|---|---|---|
| DeepCOV-ResNet | ResNet-50 | 26.49 mins. | 0.29 secs. |
| FastCOV-ResNet | | **8.51** mins. | **0.11** secs. |

**Table B.1:** Comparison of computation time for covariance representation using DeepCOV-ResNet and FastCOV-ResNet on the MIT Indoor dataset. The boldface results show that FastCOV saves the computation time.

experiment. We train DeepCOV-ResNet and FastCOV-ResNet on the MIT Indoor dataset and compare their training times for the completion of one training epoch. Both DeepCOV-ResNet and FastCOV-ResNet produce 1024×1024 sized covariance representation, which is large.

Table B.1 shows the results. As can be seen, FastCOV-ResNet computes covariance representation more than three times faster than the DeepCOV-ResNet in terms of training time. Also, it is significantly efficient than DeepCOV-ResNet in computing covariance representation per image.

# Appendix C

# Dataset Information

In this section, we provide the dataset information. We perform experiments on four widely used public image datasets, namely, MIT Indoor [85], Stanford Cars [88], CalTech-UCSD Birds (CUB 200-2011) [86], and FGVC-Aircraft [87] to demonstrate the performance of the proposed methods. Figure C.1 shows the sample images from these datasets and more details are given below.

The *MIT Indoor* dataset is one of the most widely used datasets in the literature for scene classification. It has a total of 15,620 images and 67 classes. Each image class contains a minimum of 100 images. The images are collected from various types of stores (e.g., grocery and bakery), private places (e.g., bedroom and living room), public places (e.g., prison cell, bus, and library), recreational places (e.g. restaurant, bar, and cinema hall) and working environments (e.g. office and studio).

The *Caltech-UCSD Birds* dataset or simply 'Birds' is one of the most reported datasets in the fine-grained image classification (FGIC) literature. It has a total of 11,788 images and 200 image classes. There are subtle differences between these classes and they are indistinguishable by human observers. This dataset comes with bounding box annotations; however, we do not use any annotations in our experiments.

The *FGVC-Aircraft* or 'Airplane' dataset is a relatively small dataset but widely used in recent FGIC methods. It has only 10,000 images distributed among 100 aircraft classes, and each class has precisely 100 images. Similar to the Birds dataset, the classes have subtle differences between them and are hard for humans to distinguish from each other. However, compared to the Birds dataset, the size of airplanes, i.e., objects, are relatively larger in each image.

The *Stanford Cars* or simply 'Cars' dataset has a total of 16,185 images and 196 classes. The classes are organized as per the car production year, car manufacturer, and car model. The Cars dataset has relatively small objects, i.e., cars, than those of the airplane dataset. Furthermore, the objects are in cluttered backgrounds. Table **??** gives a more concise summary of the datasets.

| | | | Predefined protocol | | |
|---|---|---|---|---|---|
| Dataset | Total classes | Total images | Training images | Testing images | Major difficulty |
| MIT Indoor | 67 | 6700 | 5,360 | 1,340 | difficult environment |
| Birds | 200 | 11,788 | 5,994 | 5,794 | subtle class difference |
| Airplane | 100 | 10,000 | 6,600 | 3,400 | subtle class difference |
| Cars | 196 | 16,185 | 8,144 | 8,041 | cluttered background |

**Table C.1:** Summary of datasets.

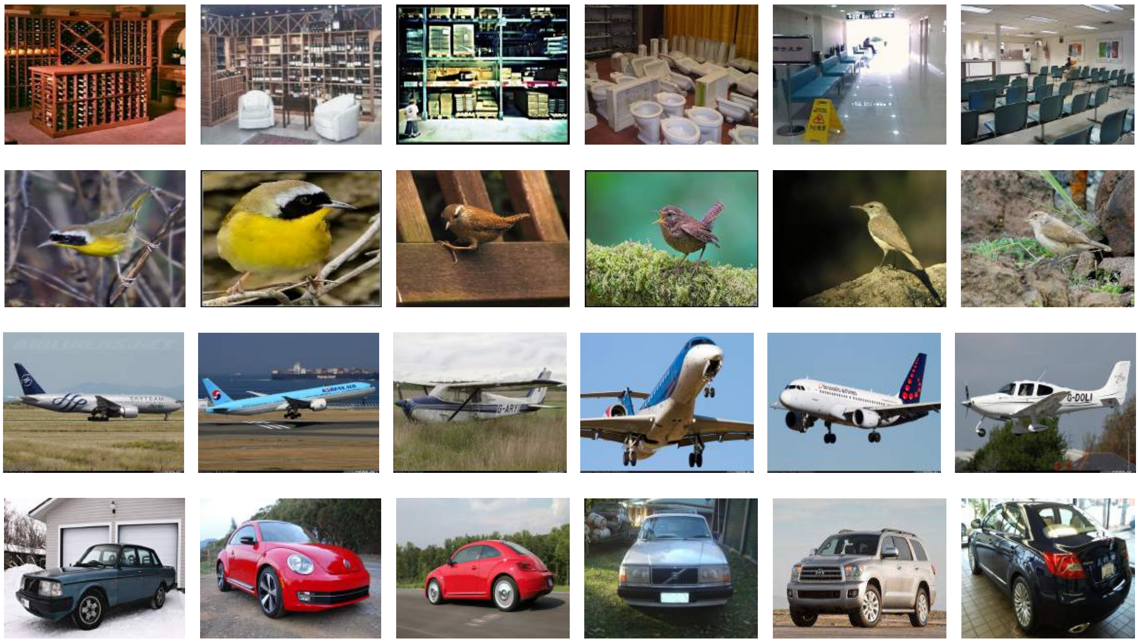

**Figure C.1:** Sample images from the datasets used in our experiments. Rows 1, 2, 3, and 4 have the images from the MIT Indoor, CalTech-UCSD Birds, FGVC-Aircraft, and Stanford Cars datasets, respectively.