Positive Definite Matrices: Data Representation and Applications to Computer Vision

Anoop Cherian and Suvrit Sra

Abstract Numerous applications in computer vision and machine learning rely on representations of data that are compact, discriminative, and robust while satisfying several desirable invariances. One such recently successful representation is offered by *symmetric positive definite* (SPD) matrices. However, the modeling power of SPD matrices comes at a price: rather than a flat Euclidean view, SPD matrices are more naturally viewed through curved geometry (Riemannian or otherwise) which often complicates matters. We focus on models and algorithms that rely on the geometry of SPD matrices, and make our discussion concrete by casting it in terms of covariance descriptors for images. We summarize various commonly used distance metrics on SPD matrices, before highlighting formulations and algorithms for solving sparse coding and dictionary learning problems involving SPD data. Through empirical results, we showcase the benefits of mathematical models that exploit the curved geometry of SPD data across a diverse set of computer vision applications.

1 Introduction

Efficient representations that compactly capture salient properties of data form the basis of every algorithm in computer vision and machine learning. Consider for instance the task of tracking a person across video frames given (i) a video sequence, and (ii) a bounding box around the object to be tracked (see Figure 1). Methods for this task typically first generate a representation (descriptor) of the image patch inside the bounding box and then proceed further. Success of the tracking application relies on several desirable properties, for instance (i) ability of the representation to

Anoop Cherian
ARC Centre of Excellence for Robotic Vision, Australian National University, Canberra, Australia
http://www.roboticvision.org e-mail: anoop.cherian@anu.edu.au

Suvrit Sra
Massachusetts Institute of Technology, Cambridge, MA, USA e-mail: suvrit@mit.edu

uniquely characterize the contents of the tracked patch against any other patch in a video frame; (ii) robustness of the descriptor to camera sensor noise; (iii) stability despite changes in illuminations; (iv) tolerance for occlusions; and (v) robustness to affine deformations of the object.

A simple descriptor for tracking could be the normalized color histogram of an image patch. This choice partially satisfies properties such as (i), (ii), and (iii). We can improve this naive color descriptor by adding additional statistics of the image patch, such as the shape and pose of the object, texture of the patch, edge orientations, etc. However, each of these additional data features requires high-dimensional descriptors for its representation, whereby the final augmented data descriptor can be extremely large. This in turn raises storage concerns, while also complicating learning, recognition, and tracking due to the curse of dimensionality.

Fig. 1 An illustrative application: people tracking in a video sequence. We are given a video sequence and a bounding box of the person to be tracked (dark box), and the problem is to track the person along the sequence, i.e., to generate the lighter (yellow) boxes in subsequent frames.

A simple but effective alternative representation that fuses such multi-modal cues was introduced in [37], dubbed *region covariance descriptors*. The key idea is to model an image patch using correlations between different low-level features. To this end, we extract raw image features such as color, intensity gradients, etc., from every pixel within the desired patch. Then we stack these raw features and compute their covariance matrix, resulting in the *covariance descriptor* (for the patch).

Although it seems surprising that such a simple approach can lead to a powerful descriptor, we must note that the diagonal of the covariance matrix captures the statistical variance of each individual feature and the off-diagonals capture the correlation between different features. Thus, this descriptor captures second-order co-occurrences of multimodal features. Since this matrix has size quadratic only in the number of features, it is independent of the number of pixels in the patch and is thus very compact. Also, since the features means are subtracted when computing the covariance, this descriptor implicitly applies mean-filtering to the data, providing noise robustness. Due to these unique properties, the use of covariance descriptors has proliferated into several applications in computer vision and beyond.

We note that SPD matrices also play an important role as data descriptors in several other non-vision applications, such as, diffusion tensor imaging [3], brain-computer interfaces [17], sound compression [45], polarimetric image modeling [24], virus classification [21], and as quantum density matrices [20]. While, some of these matrices are not covariances, they are indeed symmetric positive definite and the algorithms proposed in this chapter are directly applicable.

While constructing covariance descriptors is simple (see Section 1.1), using them can be demanding. The key difficulty arises due to the non-Euclidean nature of covariances. Although it is tempting to simply view SPD matrices using the geometry of their embedding (Euclidean) space [2], the corresponding manifold is not compact which can lead to irrelevant solutions for certain applications [4]. Therefore, several alternative geometries for SPD matrices have been considered—Section 1.2 outlines some of the more widely used choices.

Subsequently, in Section 2 we illustrate use of these geometries on the concrete tasks of sparse coding and dictionary learning. These problems have been widely studied (for vectors) and are important to a variety of applications [34]. However, their extension to SPD matrix data is non-trivial and less studied; we describe two different frameworks that achieve these extensions.

1.1 Covariance Descriptors and Example Applications

Before delving into theoretical details, we recall below details on construction of covariance descriptors and summarize two illustrative applications.

Definition 1 (Covariance Descriptor). Suppose we have a data instance $\mathbf{I} \in \mathbb{R}^n$ (image patch, video snippet, etc.). Let $f_1^j, f_2^j, \cdots, f_d^j$, (each $f_i^j \in \mathbb{R}, j = 1,2,\cdots,n$) represent d features computed for the j-th component of $\mathbf{I}$ (such as image intensity gradients, filter outputs, etc.). Let $\boldsymbol{f}^j = [f_1^j, f_2^j, \cdots, f_d^j]^T$; then the *Covariance Descriptor* S for $\mathbf{I}$ is the $d \times d$ covariance matrix given by:

$$S = \frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{f}^j - \boldsymbol{\mu})(\boldsymbol{f}^j - \boldsymbol{\mu})^T, \tag{1}$$

where $\mu = \frac{1}{n}\sum_{j=1}^{n}\boldsymbol{f}^j$ is the mean feature vector. We will assume that the features are linearly independent, in which case $S \in \mathbb{S}_+^d$, the cone of $d \times d$ SPD matrices.

1.1.1 Illustrative Application 1: People Tracking

We continue with our example of people tracking described in Figure 1. One important choice that we must make is what features to use for constructing the descriptor. For people tracking, we will first consider a texture based approach as described in [37] that uses raw image intensity features from each pixel. Given a patch $\mathbf{I}$, let $I_r(x,y), I_g(x,y), I_b(x,y)$ represent the red, green, and blue color intensities respec-

tively of pixel at spatial coordinates (x,y) in the patch. Further, let $I_x(x,y), I_y(x,y)$ represent the gray scale image intensity gradients in the horizontal and vertical directions respectively. Then, we define

$$\boldsymbol{f}(x,y) = [I_r(x,y), I_g(x,y), I_b(x,y), I_x(x,y), I_y(x,y)]^T \tag{2}$$

as the feature vector. A covariance descriptor capturing color and shape of the objects in $\mathbf{I}$ is then given by:

$$S_{tracking} = \frac{1}{|\mathbf{I}|}\sum_{x,y}(\boldsymbol{f}(x,y)-\boldsymbol{\mu})(\boldsymbol{f}(x,y)-\boldsymbol{\mu})^T, \tag{3}$$

where $\mu = \frac{1}{|\mathbf{I}|}\sum_{x,y}\boldsymbol{f}(x,y)$ is the mean feature, and $|\mathbf{I}|$ is the patch size. Note that the order of the pixels in the patch are ignored when computing the covariance matrix, thus providing invariance to changes in the pose of the object. However, we may associate additional spatial correlations between the pixel features by including the (x,y) coordinates as well into (2). In case, rotational invariance is desired, instead of including (x,y) coordinates, we could include $\sqrt{x^2+y^2}$ as an additional feature for a pixel at location (x,y). Curvature information could be easily included by computing the second-order gradients for every pixel and augmenting (2). This illustration shows the flexibility of the covariance descriptor to blend any feature of choice into the framework easily. The next example shows a richer set of features.

1.1.2 Illustrative Application 2: Face Recognition

While using raw features such as gradients could be sufficient to capture textures, recognizing faces require much more expressive and discriminative feature sets. In [39], the authors propose to use covariance descriptors for this task where the features are generated from Gabor wavelets. These filters are believed to reflect the human visual system [35] and measures the energy distribution in the image patch at various scales and orientations. These filters have been previously shown to be useful to extract discriminative and subtle features suitable for recognition [31]; using them within a covariance descriptor setup is shown to lead to significantly better recognition performance in [39], via capturing their second-order statistics. To this end, 40 Gabor wavelet filters are first designed consisting of 8 orientations and 5 different scales. Next, small patches centered at each pixel in the face patch are convolved with these filters, thus forming a 40 dimensional feature vector for each pixel, which precedes applying the steps described above to generate 40×40 covariance descriptors.

Covariance descriptors have been found to be useful in several other vision applications, such as visual surveillance [33, 36, 51], object recognition [22, 27], action recognition [47, 25, 50], image set classification [52], and emotion classification [53], to name a few. Almost all these works use a similar setup for computing the covariances, except that they use various features suitable for the application.

1.2 Geometry of SPD Matrices

To effectively use covariance descriptors we must next define schemes to compute similarity / distance between two descriptors. Intuitively, the distance measure enforces some geometry on the space of these descriptors, and which particular geometry is preferable depends on the application. As alluded to previously, a variety of SPD geometries have been explored, the simplest of which is the usual Euclidean geometry where the distance between SPD matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ is simply the Frobenius distance given by

$$\mathrm{d_F}(\boldsymbol{X},\boldsymbol{Y}) := \|\boldsymbol{X}-\boldsymbol{Y}\|_F\,. \tag{4}$$

However, $\mathrm{d_F}$ is neither affine invariant nor does it lead to a complete metric space (due to singular boundary). Affine invariance is important in applications such as diffusion MRI [40] while completeness is crucial when defining convergent sequences on the SPD manifold.

Two basic alternatives popular in computer vision are (i) the affine invariant Riemannian metric (AIRM) [40]; and (ii) the log-Euclidean Riemannian metric (LERM) [4]. Both measures induce a Riemannian geometry; the former induces a curved geometry while the latter "flattens" the manifold by mapping into the tangent space (which is Euclidean). The corresponding distance functions are

$$\mathrm{d_R}(\boldsymbol{X},\boldsymbol{Y}) := \left\|\operatorname{Log}\boldsymbol{X}^{-1/2}\boldsymbol{Y}\boldsymbol{X}^{-1/2}\right\|_F, \tag{5}$$

$$\mathrm{d_{LE}}(\boldsymbol{X},\boldsymbol{Y}) := \|\operatorname{Log}\boldsymbol{X}-\operatorname{Log}\boldsymbol{Y}\|_F\,, \tag{6}$$

where $\boldsymbol{X}^{-1/2}$ is the matrix square-root of SPD matrix $\boldsymbol{X}^{-1}$ and Log is the principal matrix logarithm. Distance (5) is affine invariant, and enjoys a host of other remarkable properties [7, Ch. 6]. LERM, however, is not affine invariant though it is rotation and scale invariant separately. Both AIRM and LERM are computationally demanding: for $d\times d$ SPD matrices, assuming approximately $4d^3/3$ flops for eigendecomposition and d^3 for matrix multiplication, these distances requires approximately $14d^3/3$ flops.

To reduce the computational cost, while preserving affine invariance and other geometric properties, the Stein distance was introduced in [15, 48]; it is defined as

$$\mathrm{d_S}(\boldsymbol{X},\boldsymbol{Y}) := \left[\log\det\left(\tfrac{\boldsymbol{X}+\boldsymbol{Y}}{2}\right)-\tfrac{1}{2}\log\det(\boldsymbol{X}\boldsymbol{Y})\right]^{1/2}. \tag{7}$$

Computing $\mathrm{d_S}$ requires only three Cholesky decompositions, at a total cost of d^3 flops. Moreover, $\mathrm{d_S}$ is analytically simpler to work with, as its gradients are also simpler to obtain than either AIRM or LERM. Consequently, it has found application is a number of recent works, some of which we will refer to in the sequel.

2 Application to Sparse Coding and Dictionary Learning

After outlining a few basic SPD geometries, we are now ready to describe concrete applications where specific properties of SPD matrices play a role. In particular, we discuss sparse coding and dictionary learning for SPD valued data. The first model we cover is *generalized dictionary learning* (GDL), a direct extension of usual (vector) dictionary learning. Our second model uses the natural Riemannian geometry of SPD matrices, and measures sparse reconstruction loss using the AIRM distance. There are other formulations for dictionary learning and sparse coding on SPD matrices, for instance [25, 26, 46]; these differ from our model primarily in the formulation of the sparse reconstruction loss.

2.1 Dictionary Learning with SPD atoms

Traditional dictionary learning takes vectors $\boldsymbol{x}_i \in \mathbb{R}^p$ $(1 \leq i \leq m)$ and constructs a matrix $\boldsymbol{B} \in \mathbb{R}^{p\times n}$ and code vectors $\boldsymbol{\alpha}_i \in \mathbb{R}^n$ (usually $n \gg p$), so that

$$\boldsymbol{x}_i \approx \boldsymbol{B}\boldsymbol{\alpha}_i, \quad \text{and} \quad \boldsymbol{\alpha}_i \text{ is sparse}, \quad \text{for } 1 \leq i \leq m.$$

The sparsity requirement on $\boldsymbol{\alpha}_i$ is commonly enforced using ℓ_0- or ℓ_1-norm penalties or constraints. Since both $\boldsymbol{B}$ and $\boldsymbol{\alpha}_i$ are unknown dictionary learning usually results in a difficult nonconvex optimization task. Nevertheless, it has found remarkable success toward sparse coding and other applications [18].

We depart from the above setup in that we consider input matrices $\boldsymbol{X}_i \in \mathbb{R}^{p\times q}$, $1 \leq i \leq m$. Then, instead of a dictionary matrix $\boldsymbol{B}$ we learn a tensor B, which we identify with a linear operator $\mathsf{B} : \mathbb{R}^{n\times r} \to \mathbb{R}^{p\times q}$ so that

$$\boldsymbol{X}_i \approx \mathsf{B}(\boldsymbol{A}_i), \quad \text{and} \quad \boldsymbol{A}_i \text{ is sparse}, \quad \text{for } 1 \leq i \leq m. \tag{8}$$

Using (8), one model of GDL is the following [49]:

$$\min_{\boldsymbol{A}_1,\ldots,\boldsymbol{A}_m,\mathsf{B}} \quad \tfrac{1}{2}\sum_{i=1}^m \|\boldsymbol{X}_i - \mathsf{B}(\boldsymbol{A}_i)\|_{\mathrm{F}}^2 + \sum_{i=1}^m \beta_i \mathrm{sp}(\boldsymbol{A}_i), \tag{9}$$

where $\beta_i > 0$ are scalar hyperparameters while $\mathrm{sp}(\boldsymbol{A})$ enforces some notion of sparsity. For instance, $\mathrm{sp}(\boldsymbol{A})$ could be the cardinality function $\|\boldsymbol{A}\|_0$, its convex relaxation $\|\boldsymbol{A}\|_1$, the matrix rank $\mathrm{rank}(\boldsymbol{A})$ or its convex relaxation, the trace-norm $\|\boldsymbol{A}\|_{\mathrm{tr}}$.

Formulation (9) requires one more modification for SPD valued inputs. Specifically, we ensure that the approximation $\mathsf{B}(\boldsymbol{A}_i)$ is also SPD by defining B via

$$\mathsf{B}(\boldsymbol{A}) := \boldsymbol{B}\boldsymbol{A}\boldsymbol{B}^T, \quad \text{for some matrix } \boldsymbol{B}, \tag{10}$$

and additionally *restricting* to $\boldsymbol{A} \succeq 0$. Observe that (10) can be written as

$$\mathrm{vec}(\boldsymbol{B}\boldsymbol{A}\boldsymbol{B}^T) = (\boldsymbol{B}\otimes\boldsymbol{B})\,\mathrm{vec}(\boldsymbol{A}), \tag{11}$$

where vec stacks columns of its argument and the operator B is encoded (isomorphically) by the product $\boldsymbol{B} \otimes \boldsymbol{B}$.

It is easy to show that (11) requires $md^2 + d^2n + mn$ storage for m covariance matrices of size $d \times d$, while (10) takes $md^2 + dn + mn$. Computationally, also the first formulation is cheaper, so we prefer it. As to $\boldsymbol{A}$, we consider two choices:

1. $\boldsymbol{A} = \mathrm{Diag}(\alpha_1, \ldots, \alpha_n)$ where $\alpha_i \geq 0$; and
2. $\boldsymbol{A} = \sum_{j=1}^{k} \boldsymbol{\alpha}_j \boldsymbol{\alpha}_j^T$, a potentially low-rank (if $k < n$) SPD matrix.

Although diagonal $\boldsymbol{A}$ might appear to be simple, it is quite powerful. Indeed, with it GDL models SPD matrices as weighted sums of rank-one matrices since

$$\boldsymbol{X} \approx \boldsymbol{B}\boldsymbol{A}\boldsymbol{B}^T = \sum_{i=1}^{n} \alpha_i \boldsymbol{b}_i \boldsymbol{b}_i^T, \quad \text{where } \alpha_i = \boldsymbol{A}_{ii}, \tag{12}$$

which offers a rich yet computationally tractable model.

2.1.1 Stochastic Gradient for GDL

Now we derive a stochastic-gradient procedure for approximately solving GDL. We use the convex function $\mathrm{sp}(\boldsymbol{A}) = \|\boldsymbol{A}\|_1$ for enforcing sparsity. Then, using representation (12) with diagonal $\boldsymbol{A}_i$, the GDL optimization problem (9) becomes

$$\min_{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_N \geq 0, \boldsymbol{B}} \quad \tfrac{1}{2} \sum_{i=1}^{m} \|\boldsymbol{X}_i - \boldsymbol{B}\boldsymbol{A}_i\boldsymbol{B}^T\|_{\mathrm{F}}^2 + \sum_{i=1}^{m} \beta_i \|\boldsymbol{A}_i\|_1. \tag{13}$$

Problem (13) is nonconvex and difficult. However, for a fixed dictionary $\boldsymbol{B}$, it is individually convex in $(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m)$. It is thus amenable to the idea of alternating between updating $\boldsymbol{B}$ and optimizing over $(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_m)$. But in many applications the number of input data points m is very large, so the alternating steps can easily become rather costly. Therefore, we follow a stochastic gradient approach that can scale to large data sets, as long as the stochastic gradients can be obtained efficiently.

To prevent degenerate solutions we also impose normalization constraints $\|\boldsymbol{b}_j\|_2 \leq 1$ on each column of matrix $\boldsymbol{B}$. We denote these requirements by the feasible set $\mathscr{B}$. We run stochastic gradient using K "mini-batches," for which we rewrite (13) as

$$\min_{\boldsymbol{B} \in \mathscr{B}} \quad \Phi(\boldsymbol{B}) := \sum_{b=1}^{K} \phi_b(\boldsymbol{B}), \tag{14}$$

where ϕ_b denotes the objective function for batch b. Let k_b be the size of batch b ($1 \leq b \leq K$) containing the matrices $\{\boldsymbol{X}_{j(i)} | 1 \leq i \leq k_b\}$, where $j(i)$ is an appropriate index in $1, \ldots, m$. With this notation, the objective function for batch b is

$$\phi_b(\boldsymbol{B}) := \min_{\boldsymbol{A}_{j(1)}, \ldots, \boldsymbol{A}_{j(k)} \geq 0} \quad \tfrac{1}{2} \sum_{i=1}^{k_b} \|\boldsymbol{X}_{j(i)} - \boldsymbol{B}\boldsymbol{A}_{j(i)}\boldsymbol{B}^T\|_{\mathrm{F}}^2 + \beta_{j(i)} \|\boldsymbol{A}_{j(i)}\|_1. \tag{15}$$

We apply stochastic-gradient to (14), which performs the iteration

$$\boldsymbol{B}_{t+1} = \Pi_{\mathscr{B}}(\boldsymbol{B}_t - \eta_t \nabla_{\boldsymbol{B}} \phi_{b(t)}(\boldsymbol{B}_t)), \quad b(t) \in [1..K],\ t = 0, 1, \ldots, \tag{16}$$

where $\Pi_{\mathscr{B}}$ denotes orthogonal projection onto $\mathscr{B}$. Assuming (15) has a unique solution, the gradient $\nabla_{\boldsymbol{B}}\phi_{b(t)}$ is well defined. Specifically, let $(\boldsymbol{A}^*_{j(1)},\ldots,\boldsymbol{A}^*_{j(k)})$ be the argmin of (15). Then, writing $b \equiv b(t)$, we have

$$\nabla_{\boldsymbol{B}}\phi_b(\boldsymbol{B}) = 2\sum_{i=1}^{k_b}\Big(\boldsymbol{B}\boldsymbol{A}^*_{j(i)}\boldsymbol{B}^T - \boldsymbol{X}_{j(i)}\Big)\boldsymbol{B}\boldsymbol{A}^*_{j(i)}. \tag{17}$$

The computationally intensive part is to compute (17), which we now consider.

2.1.2 Sparse Coding: Computing $\nabla\phi_b$

Observe that (15) is a sum of k_b independent problems, so it suffices to describe the computation for a subproblem of the form

$$\min_{\boldsymbol{A}\geq 0} \quad f(\boldsymbol{A}) := \tfrac{1}{2}\|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{A}\boldsymbol{B}^T\|_{\mathrm{F}}^2 + \beta\|\boldsymbol{A}\|_1. \tag{18}$$

Since $\boldsymbol{A} \geq 0$ and diagonal, problem (18) is nothing but a regularized nonnegative least-squares (NNLS) problem. There exist a variety of solvers for NNLS, for example, LBFGS-B [32], or Spectral Projected-Gradient (SPG) [8]. We prefer to use the latter, as it is not only simple, but also exhibits excellent empirical performance.

In Section 3, we will apply this sparse coding scheme to the problem of nearest neighbor retrieval on covariance datasets. But, before proceeding to the experiments, we will elucidate a much richer and more powerful dictionary learning and sparse coding scheme on SPD matrices that leads to significantly better results on various applications; this scheme uses the natural Riemannian geometry for the sparse construction loss instead of the Euclidean distance as in (9).

2.2 Riemannian Dictionary Learning and Sparse Coding

Recall that we wish to compute a dictionary with "SPD atoms". We work on manifold $\mathbb{M}_n^d = \prod_{i=1}^n \mathbb{S}_+^d \subset \mathbb{R}^{d\times d\times n}$, which is the Cartesian product of n SPD manifolds. Our goals are (i) to learn a dictionary $\mathsf{B} \in \mathbb{M}_n^d$ in which each slice represents an SPD dictionary atom $\boldsymbol{B}_j \in \mathbb{S}_+^d$ $1 \leq j \leq n$; and (ii) to approximate each $\boldsymbol{X}_i$ as a sparse conic combination of atoms in B; i.e., $\boldsymbol{X}_i \sim \mathsf{B}(\boldsymbol{\alpha}_i)$ where $\alpha_i \in \mathbb{R}_+^n$ and $\mathsf{B}(\boldsymbol{\alpha}) := \sum_{i=1}^n \boldsymbol{\alpha}_i\boldsymbol{B}_i$. With this notation our *dictionary learning and sparse coding* (DLSC) problem is

$$\min_{\mathsf{B}\in\mathbb{M}_n^d,\boldsymbol{\alpha}\in\mathbb{R}_+^{n\times m}} \tfrac{1}{2}\sum_{j=1}^m \mathrm{d_R}^2(\boldsymbol{X}_j, \mathsf{B}\boldsymbol{\alpha}_j) + \mathrm{Sp}(\boldsymbol{\alpha}_j) + \Omega(\mathsf{B}), \tag{19}$$

where Sp and Ω regularize the codes $\boldsymbol{\alpha}_j$ and the dictionary tensor B respectively.

Formulation (19) is a direct SPD analog of the vector DL setup. Instead of learning a dictionary matrix for vectors, we learn a third-order tensor dictionary as our input is matricial data. We constraint the sparse codes to be non-negative to ensure that

the linear combination $\mathsf{B}(\boldsymbol{\alpha})$ remains SPD. In contrast to usual DL problems where the dictionary learning and sparse coding subproblems are convex, problem (19) is much harder: it is neither convex in itself nor are its subproblems convex.

Pragmatically speaking, this lack of subproblem convexity is not too damaging: we just need a set of dictionary atoms that can sparse code the input data, and such a set can still be computed by performing an alternating minimization (actually, just descent here). We describe the details below.

2.2.1 Dictionary Learning Subproblem

Assuming that the sparse code vectors $\boldsymbol{\alpha}$ are available, the subproblem of updating the dictionary atoms can be separated from (19) and written as:

$$\begin{aligned}\min_{\mathsf{B}\in\mathbb{M}_n^d} \Theta(\mathsf{B}) &:= \tfrac{1}{2}\sum_{j=1}^m \mathrm{d_R}^2(\boldsymbol{X}_j, \mathsf{B}\alpha_j) + \Omega(\mathsf{B}),\\ &= \tfrac{1}{2}\sum_{j=1}^m \left\|\mathrm{Log}\left(\boldsymbol{X}_j^{-1/2}(\mathsf{B}\alpha_j)\boldsymbol{X}_j^{-1/2}\right)\right\|_F^2 + \Omega(\mathsf{B}). \end{aligned} \tag{20}$$

Due to its good empirical performance we choose $\Omega(\mathsf{B}) := \lambda_{\mathsf{B}}\sum_{i=1}^n \mathrm{Tr}(\boldsymbol{B}_i)$.

Riemannian CG

Among several first-order alternatives for optimizing over the SPD atoms (such as the steepest-descent, trust-region methods [1], etc.), the Riemannian Conjugate Gradient (CG) method [2, Ch. 8], was found to be empirically more stable and faster. Below, we provide a short exposition of the CG method in the context of minimizing over B which belongs to an SPD product manifold.

For an arbitrary non-linear function $\theta(x),\ x\in\mathbb{R}^n$, the CG method uses the following recurrence at step $k+1$

$$x_{k+1} = x_k + \gamma_k \xi_k, \tag{21}$$

where the direction of descent ξ_k is

$$\xi_k = -\operatorname{grad}\theta(x_k) + \mu_k \xi_{k-1}, \tag{22}$$

with $\operatorname{grad}\theta(x_k)$ defining gradient of θ at x_k ($\xi_0 = -\operatorname{grad}\theta(x_0)$), and μ_k given by

$$\mu_k = \frac{(\operatorname{grad}\theta(x_k))^T(\operatorname{grad}\theta(x_k) - \operatorname{grad}\theta(x_{k-1}))}{\operatorname{grad}\theta(x_{k-1})^T \operatorname{grad}\theta(x_{k-1})}, \tag{23}$$

The step-size γ_k in (21) is usually found via an efficient line-search method [6]. It can be shown that [6, Sec. 1.6] when θ is quadratic with a Hessian Q, the directions generated by (22) are Q-conjugate to previous directions of descent $\xi^0, \xi^1, \cdots, \xi^{k-1}$; thereby (21) providing the exact minimizer of f in fewer than d iterations (d is the manifold dimension).

For $\mathsf{B} \in \mathbb{M}_n^d$ and referring back to (20), the recurrence in (21) uses the Riemannian retraction [2, Ch. 4] and the gradient $\operatorname{grad}\Theta(\mathsf{B}_k)$ is the Riemannian gradient (here B_k represents the dictionary tensor at the k-th iteration). This leads to an important issue: the gradients $\operatorname{grad}\Theta(\mathsf{B}_k)$ and $\operatorname{grad}\Theta(\mathsf{B}_{k-1})$ belong to two different tangent spaces $T_{\mathsf{B}_k}\mathbb{M}$ and $T_{\mathsf{B}_{k-1}}\mathbb{M}$ respectively, and thus cannot be combined as in (23). Thus, following [2, Ch. 8] we resort to vector transport – a scheme to transport a tangent vector at $P \in \mathbb{M}$ to a point $\operatorname{Exp}_P(S)$ where $S \in T_P\mathbb{M}$ and Exp is the exponential map. The resulting formula for the direction update becomes

$$\xi_{\mathsf{B}_k} = -\operatorname{grad}\Theta(\mathsf{B}_k) + \mu_k \mathfrak{T}_{\gamma_k \xi_{k-1}}(\xi_{k-1}), \tag{24}$$

where

$$\mu_k = \frac{\left\langle \operatorname{grad}\Theta(\mathsf{B}_k), \operatorname{grad}\Theta(\mathsf{B}_k) - \mathfrak{T}_{\gamma_k \xi^{k-1}}(\operatorname{grad}\Theta(\mathsf{B}_{k-1})) \right\rangle}{\langle \operatorname{grad}\Theta(\mathsf{B}_{k-1}), \operatorname{grad}\Theta(\mathsf{B}_{k-1}) \rangle}. \tag{25}$$

Here for $Z_1, Z_2 \in T_P\mathbb{M}$, the map $\mathfrak{T}_{Z_1}(Z_2)$ defines the vector transport given by:

$$\mathfrak{T}_{Z_1}(Z_2) = \frac{d}{dt} \exp_P(Z_1 + tZ_2)\Big|_{t=0}. \tag{26}$$

It remains to derive an expression for the Riemannian gradient $\operatorname{grad}\Theta(\mathsf{B})$.

Riemannian Gradient

Lemma 1 connects the Riemannian gradient to the Euclidean gradient of $\Theta(\mathsf{B})$.

Lemma 1. *For a dictionary tensor $\mathsf{B} \in \mathbb{M}_n^d$, let $\Theta(\mathsf{B})$ be a differentiable function. Then the Riemannian gradient* $\operatorname{grad}\Theta(\mathsf{B})$ *satisfies the following equation:*

$$\langle \operatorname{grad}\Theta(\mathsf{B}), \zeta \rangle_{\mathsf{B}} = \langle \nabla\Theta(\mathsf{B}), \zeta \rangle_I, \forall \zeta \in T_P\mathbb{M}_n^d, \tag{27}$$

where $\nabla\Theta(\mathsf{B})$ is the Euclidean gradient of $\Theta(\mathsf{B})$. The Riemannian gradient for the i-th dictionary atom is given by $\operatorname{grad}_i\Theta(\mathsf{B}) = \boldsymbol{B}_i \nabla_{\boldsymbol{B}_i}\Theta(\mathsf{B})\boldsymbol{B}_i$.

Proof. See [2, Ch. 5]. The latter expression is obtained by substituting the inner product on the LHS of (27) by its definition in (5).

The Euclidean gradient $\nabla\Theta(\mathsf{B})$ is obtained as follows: let $\boldsymbol{S}_j = \boldsymbol{X}_j^{-1/2}$ and $M_j(\mathsf{B}) := \mathsf{B}(\boldsymbol{\alpha}_j) = \sum_{i=1}^n \boldsymbol{\alpha}_j^i \boldsymbol{B}_i$. Then,

$$\Theta(\mathsf{B}) = \tfrac{1}{2}\sum_{j=1}^m \operatorname{Tr}(\operatorname{Log}(\boldsymbol{S}_j M_j(\mathsf{B}) \boldsymbol{S}_j)^2) + \lambda_{\mathsf{B}} \sum_{i=1}^n \operatorname{Tr}(\boldsymbol{B}_i). \tag{28}$$

The derivative $\nabla_{\boldsymbol{B}_i}\Theta(\mathsf{B})$ of (28) w.r.t. to atom $\boldsymbol{B}_i$ is:

$$\sum_{j=1}^m \alpha_j^i \big(\boldsymbol{S}_j \operatorname{Log}(M_j(\mathsf{B}))\big(M_j(\mathsf{B})\big)^{-1} \boldsymbol{S}_j\big) + \lambda_{\mathsf{B}} I. \tag{29}$$

2.2.2 Sparse Coding Subproblem

Referring back to (19), we now consider the sparse coding subproblem. Given a dictionary tensor B and a data matrix $\boldsymbol{X}_j \in \mathbb{S}_+^d$, this subproblem requires solving

$$\begin{aligned} &\min_{\boldsymbol{\alpha}_j \geq 0} \quad \phi(\boldsymbol{\alpha}_j) := \tfrac{1}{2}\,\mathrm{d_R}^2(\boldsymbol{X}_i, \mathsf{B}(\boldsymbol{\alpha}_j)) + \mathrm{Sp}(\boldsymbol{\alpha}_j) \\ &= \tfrac{1}{2}\left\|\mathrm{Log}\big(\textstyle\sum_{i=1}^n \alpha_j^i \boldsymbol{X}^{-1/2}\boldsymbol{B}_j\boldsymbol{X}^{-1/2}\big)\right\|_F^2 + \mathrm{Sp}(\boldsymbol{\alpha}_j), \end{aligned} \tag{30}$$

where α_j^i is the i-th component of $\boldsymbol{\alpha}_j$ and Sp is a sparsity inducing function. For simplicity, we use $\mathrm{Sp}(\boldsymbol{\alpha}) = \lambda\|\boldsymbol{\alpha}\|_1$, where $\lambda > 0$ is a regularization parameter. Since we are working with $\boldsymbol{\alpha} \geq 0$, we replace this penalty by $\lambda \sum_i \alpha_i$, which is differentiable.

Problem (30) measures reconstruction quality offered by a sparse non-negative linear combination of the atoms to a given input point $\boldsymbol{X}$. It will turn out (see experiments in Section 3) that the reconstructions obtained via this model actually lead to significant improvements in performance over sparse coding models that ignore SPD geometry. But this gain comes at a price: objective (30) is difficult to optimize, and remains difficult even if we take into account geodesic convexity of $\mathrm{d_R}$.

While in practice this nonconvexity does not seem to hurt our model, we digress below to show a surprising but intuitive constraint under which Problem (30) actually becomes convex. Although we do not exploit this observation to save on computation, we highlight it here due to its theoretical appeal.

Theorem 1 ([14]). *The function $\phi(\alpha) := \mathrm{d_R}^2(\sum_i \alpha_i B_i, X)$ is convex on the set*

$$\mathscr{A} := \{\alpha \mid \textstyle\sum_i \alpha_i B_i \preceq X, \text{ and } \alpha \geq 0\}. \tag{31}$$

2.2.3 Optimizing Sparse Codes

We minimize (30) using a projected gradient method. Specifically, we run the iteration

$$\alpha^{k+1} \leftarrow \mathscr{P}[\alpha^k - \eta_k \nabla\phi(\alpha^k)], \qquad k = 0, 1, \ldots, \tag{32}$$

where $\mathscr{P}[\cdot]$ denotes the projection operator defined as

$$\mathscr{P}[\alpha] \equiv \alpha \mapsto \mathrm{argmin}_{\alpha'}\ \tfrac{1}{2}\|\alpha' - \alpha\|_2^2, \quad \text{s.t. },\ \alpha' \in \mathscr{A}. \tag{33}$$

To implement iteration (32) we need to specify three components: (i) the stepsize η_k; (ii) the gradient $\nabla\phi(\alpha^k)$; and (iii) the projection (33). Lemma 2 shows how to compute the gradient. The projection task (33) is a special least-squares (dual) semidefinite program (SDP), which can be solved using any SDP solver. However, in the interest of speed, we avoid the heavy computational burden imposed by an SDP, and drop the constraint $\alpha \in \mathscr{A}$. Although this sacrifices convexity, the resulting computation is vastly easier, and work well empirically. With this change, we simply have $\mathscr{P}[\alpha] = \max(0, \alpha)$.

It remains to specify how to obtain the stepsize η_k. There are several choices in the nonlinear programming literature [6], but most of them can be expensive in our setting. We wish to avoid expensive iterative algorithms for computing η_k, and thus choose to use Barzilai-Borwein stepsizes [5] that have closed forms and that often work remarkably well in practice [5, 43]. In particular, we use the Spectral Projected Gradient (SPG) method [9] by adapting a simplified implementation of [43].

Lemma 2. *Let $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{X}$ be fixed SPD matrices. Consider the function $f(x) := \mathrm{d_R}^2(x\boldsymbol{B}+\boldsymbol{C},\boldsymbol{X})$. The derivative $f'(x)$ is given by*

$$f'(x) = 2\,\mathrm{Tr}(\log(\boldsymbol{S}(x\boldsymbol{B}+\boldsymbol{C})\boldsymbol{S})\boldsymbol{S}^{-1}(x\boldsymbol{B}+\boldsymbol{C})^{-1}\boldsymbol{B}\boldsymbol{S}), \qquad \textit{where } \boldsymbol{S} = \boldsymbol{X}^{-1/2}. \quad (34)$$

Proof. Introduce the shorthand $\boldsymbol{M}(x) \equiv x\boldsymbol{B}+\boldsymbol{C}$. Using (5) we have

$$f(x) = \mathrm{Tr}([\log(\boldsymbol{S}\boldsymbol{M}(x)\boldsymbol{S})]^T[\log(\boldsymbol{S}\boldsymbol{M}(x)\boldsymbol{S})]),$$

The chain-rule of calculus then immediately yields the desired result

$$f'(x) = 2\,\mathrm{Tr}(\log(\boldsymbol{S}\boldsymbol{M}(x)\boldsymbol{S})(\boldsymbol{S}\boldsymbol{M}(x)\boldsymbol{S})^{-1}\boldsymbol{S}\boldsymbol{M}'(x)\boldsymbol{S}).$$

Writing $\boldsymbol{M}(\alpha_p) = \alpha_p\boldsymbol{B}_p + \sum_{i\neq p}\alpha_i\boldsymbol{B}_i$ and using Lemma 2 we obtain

$$\frac{\partial\phi(\alpha)}{\partial\alpha_p} = \mathrm{Tr}\big(\log\big(\boldsymbol{S}\boldsymbol{M}(\alpha_p)\boldsymbol{S}\big)\big(\boldsymbol{S}\boldsymbol{M}(\alpha_p)\boldsymbol{S}\big)^{-1}\boldsymbol{S}\boldsymbol{B}_p\boldsymbol{S}\big) + \lambda. \quad (35)$$

Computing (35) for all α is the dominant when running SPG. A a naïve implementation of (35) costs $O(nd^3)$, but with slight care this cost can be reduced $O(nd^2)+O(d^3)$ [14].

3 Applications of Sparse Coding

In this section, we describe an application of sparse coding for covariances, namely nearest neighbor (NN) retrieval. This is a fundamental task in several computer vision and machine learning applications in which the goal is to find a data point closest to a given query point within a large database.

3.1 Nearest Neighbors on Covariance Descriptors

Suppose we have obtained a sparse code matrix $\boldsymbol{A}$ using either GDL or Riemannian DLSC for an input matrix $\boldsymbol{X}$. Since, we use an overcomplete dictionary typically only a few dictionary atoms participate in the reconstruction of $\boldsymbol{X}$. Thus, with high probability dissimilar input points will obtain different sparse codes. In other words, suppose that we use a dictionary with n rank-one atoms and that only r of these

matrices are used to obtain a reconstruction for a given input. Then, there are $\binom{n}{r}$ unique basis combinations possible. With appropriate choices of n and r, we will likely obtain a unique set of rank-one matrices that encode $\boldsymbol{X}$.

Using this intuition, we propose a sorted integer tuple representation to encode an input covariance matrix; the integers simply index the dictionary atoms used in the encoding. Formally, let $\boldsymbol{X} \in \mathbb{S}_+^d$ be the input, B an overcomplete dictionary, and $\boldsymbol{u}_i$ ($i \in [n]$) a unique identifier for the i-th atom of B. If $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)^T$ is the coefficient vector corresponding to $\boldsymbol{X} \approx \mathsf{B}(\boldsymbol{\alpha})$, then the tuple $h(\boldsymbol{X}) = \langle \boldsymbol{u}_i, \cdots, \boldsymbol{u}_k \rangle$ is the *hash code* of $\boldsymbol{X}$. We choose only those identifiers for which the code α_j is larger than a threshold $\varepsilon \geq 0$.

In our case, we assume that the u_i's are just integers in $\{1, \ldots, n\}$ and that the hash code is a sorted tuple of these indices. The threshold ε helps select significant coefficients from the sparse coding, and makes the chosen code robust to noise. This coded representation enables the use of hash tables for fast locality sensitive hashing. Let us see how. Each column of the dictionary is identified by its index number; so each hash-key is a set of integers encoded as a character string. To tackle collisions in the hash buckets, the colliding input matrices are organized as a linked list. If the linked list gets too long, the data within a hash bucket can be further organized using a metric tree or any other efficient data structure. This idea of hashing is a direct adaptation of the scheme proposed in [12].

Given a query SPD matrix, we solve the sparse coding problem to first obtain the corresponding sparse coefficients. From these coefficients we compute the above hash code query the hash table. If there are several entries in a matching bucket, we run a linear scan using the AIRM distance (5) to find the best matches (the bucket can also be organized for faster than linear scans, if desired).

3.2 GDL Experiments

This section illustrates nearest neighbor search based upon our dictionary learning examples. We use the following datasets:

- **Face recognition.** The *FERET face dataset* [42, 41] contains facial appearances segregated into multiple classes. Each class has different views of the face of the same person for varying poses. We selected six images from each class. Inspired by the success of covariances created from Gabor filters for face recognition [31], we applied 40 Gabor filters on each image, later combining the filters into a covariance of size 40×40. We created a covariance dataset of approximately 10K descriptors using this approach.
- **Texture classification.** Texture is an essential cue in many data mining applications like satellite imagery, industry inspection systems, etc. Thus, we used a combination of the *Brodatz dataset* [11] and the *Curret dataset* [16] for creating a texture covariance dataset. Brodatz dataset contains approximately 111 texture classes, while Curret dataset contains 60 classes. To create the covariances data, we used the feature vector $F = [x, y, I, I_x, I_y]$, where the first two dimensions are

the relative location of the pixel with respect to the texture patch, the third dimension encodes the grayscale intensity, and the last two dimensions capture the pixel gradients. Thus, each covariance is 5×5, and we created approximately 40K such covariances.

Methods Compared. We compare against locality sensitive hashing (LSH) of vectorized covariances (VEC), hashing after log-Euclidean embedding (L2LSH), and kernelized LSH [28] using an RBF kernel using the AIRM distance.

3.2.1 GDL Experimental Setup

We implemented GDL in Matlab. For L2LSH, VEC, and HAM we used the C-implementation from the Caltech Toolbox.[1] Since the programs have different computational baselines, we cannot compare their retrieval speed. Rather, we show in Table 1 the average portion of each of the datasets scanned by GDL to find the nearest neighbor. The geodesic distance was used to resolve hash table collisions. As is seen from the table, the percentage coverage is low, which is exactly as desired.

Dataset	Faces	Texture
Avg. coverage (%)	3.54	6.26

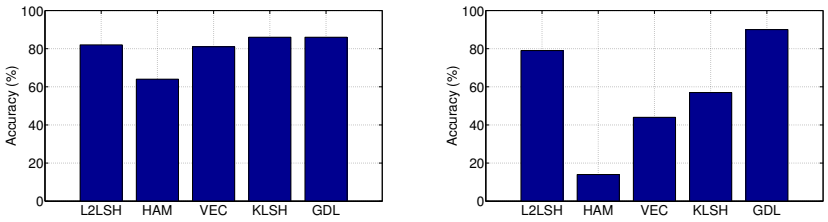
Table 1 Percentage of the database searched to find the nearest neighbor using sparse codes generated by GDL.

Next, we substantiate effectiveness of our NN retrieval scheme. To this end, we split each of the datasets into database and query sets (approximately 5% of the data). To compute the ground truth we use a linear scan over the database using geodesic distances to measure nearness. Since ensuring exact NN is hard, we restrict our search to Approximate Nearest Neighbors (ANN). Assume $\boldsymbol{Q}$ is a query point, $\boldsymbol{X}_{ls}$ is the exact NN found by a linear scan and $\boldsymbol{X}_{algo}$ is the neighbor returned by an NN algorithm. We classify an NN as correct if $\frac{d_{AIRM}(\boldsymbol{Q},\boldsymbol{X}_{ls})}{d_{AIRM}(\boldsymbol{Q},\boldsymbol{X}_{AIRM})} > \varepsilon$; we use $\varepsilon = 0.75$ below. Fig. 2 shows the accuracy of different methods, where

$$\text{Accuracy} := \frac{\text{\#correct matches}}{\text{\#query size}}. \tag{36}$$

The plots indicate that GDL performs well across the datasets, while performance of the other methods varies. Vectorizing input matrices fails on all datasets, while KLSH performs reasonably well. We note, however, that KLSH needs to compute the kernel matrix for the query point against the *entire* dataset—this can drastically slow it down. On the face dataset, all methods had high accuracy, most probably because this dataset is noise free.

[1] http://www.vision.caltech.edu/malaa/software/research/image-search/



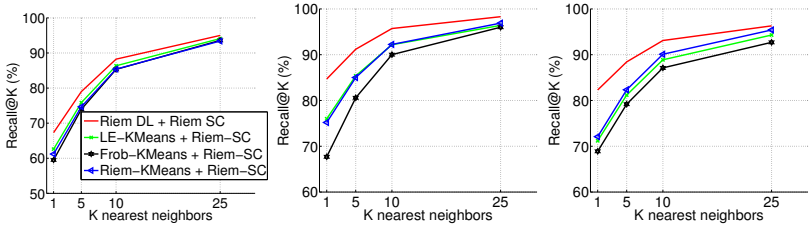
(a) (b)

Fig. 2 Plots demonstrating nearest neighbor classification accuracy of GDL generated sparse codes compared to various standard techniques; (a) on faces dataset and (b) on texture dataset.

3.3 Riemannian Dictionary Learning Experiments

Next, we evaluate the Riemannian DLSC setup, and denote below dictionary learning by DL and sparse coding by SC. We compare our Riemannian (Riem) formulation against combinations of several state-of-the-art DLSC methods on SPD matrices, namely (i) log-Euclidean (LE) metric for DLSC [25], (ii) Frobenius norm (Frob) which discards the manifold structure, (iii) kernel methods such as the Stein-Kernel [48] proposed in [26], and the log-Euclidean kernel [30].

We experiment on data available from three standard computer vision applications: (i) 3D object recognition on the RGBD objects dataset [29]; (ii) texture recognition on the standard Brodatz dataset [38]; and (iii) person re-identification on the ETHZ people dataset [19]. We describe (i) and (iii) below.



(a) Brodatz textures (b) RGB 3D objects (c) ETHZ people

Fig. 3 Results of Nearest neighbor recall@K accuracy against increasing number of retrieved points (K). Comparisons of Riem-DL and Riem-SC against other DL learning schemes based on clustering, while using our Riem-SC for sparse coding.

– **Person re-identification task.** We use the benchmark ETHZ dataset [44] for evaluating people re-identification. This dataset consists of low-resolution images of tracked people from a real-world surveillance setup. The images are from 146 different individuals. There are about 5–356 images per person. There are a total

of 8580 images in this dataset. Rather than detailing the results on several feature combinations, we describe here the feature combination that worked the best in our experiments. For this purpose, we used a validation set of 500 covariances and 10 true clusters from this dataset. The performance was evaluated using the Log-Euclidean SC setup with a dictionary learning via Log-Euclidean K-Means. We used a combination of nine features for each image as described below:

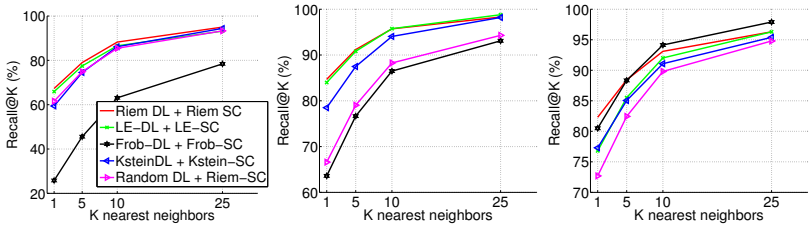
$$F_{ETHZ} = \left[x,\, I_r,\, I_g,\, I_b,\, Y_i,\, |I_x|,\, \left|I_y\right|, |\sin(\theta)+\cos(\theta)|,\, \left|H_y\right|\right],$$

where x is the x-coordinate of a pixel location, I_r, I_g, I_b are the RGB color of a pixel, Y_i is the pixel intensity in the YCbCr color space, I_x, I_y are the gray scale pixel gradients, and H_y is the y-gradient of pixel hue. We also use the gradient angle $\theta = \tan^{-1}(I_y/I_x)$ in our feature set. Each image is resized to a fixed size 300×100, and divided into upper and lower parts. We compute two different region covariances for each part, which are combined as two block diagonal matrices to form a single covariance of size 18×18 for each appearance image.

– **3D Object Recognition.** The goal of this experiment is to recognize objects in 3D point clouds. We use the public RGB-D Object dataset [29], which consists of about 300 objects belonging to 51 categories and spread across ∼250K frames. We used approximately 15K frames for our evaluation with approximately 250–350 frames devoted to every object seen from three different viewpoints (30, 45, and 60 degrees above the horizon). Following the procedure suggested in [23][Chap. 5], for every frame, the object was segmented out and 18 dimensional feature vectors generated for every 3D point in the cloud (and thus 18×18 covariance descriptors); the features we used are as follows:

$$F_{RGBD} = [x, y, z, I_r, I_g, I_b, I_x, I_y, I_{xx}, I_{yy}, I_{xy}, I_m, \delta_x, \delta_y, \delta_m, \nu_x, \nu_y, \nu_z], \qquad (37)$$

where the first three dimensions are the spatial coordinates, I_m is the magnitude of the intensity gradient, δ's represent gradients over the depth-maps, and ν represents the surface normal at the given 3D point.



(a) Brodatz textures (b) RGB 3D objects (c) ETHZ people

Fig. 4 Results of k-nearest neighbor retrieval accuracy against an increasing number of retrieved points (K). Comparisons of Riem-DL and Riem-SC against other dictionary learning and sparse coding combinations.

3.3.1 Evaluation Techniques

We evaluate our algorithms for nearest neighbor (NN) retrieval against a gallery set via computing the Euclidean distances between sparse codes. We use the standard Recall@K accuracy defined as follows: Given a gallery $\mathscr{X}$ and a query set $\mathscr{Q}$. Recall@K computes the average accuracy when retrieving K nearest neighbors from $\mathscr{X}$ for each instance in $\mathscr{Q}$. Suppose G_K^q is the set of ground truth class labels associated with the qth query, and S_K^q is the set of labels associated with the K neighbors found by some algorithm, then

$$\text{Recall@K} := \frac{1}{|Q|} \sum_{q \in Q} \frac{|G_K^q \cap S_K^q|}{|G_K^q|}. \tag{38}$$

All the experiments used 5-fold cross-validation in which 80% of the datasets were used for training the dictionary, 10% for generating the gallery set, and the rest as queries. The size of the dictionary was considered to be twice the number of classes in the respective dataset. This scheme was considered for all the comparison methods as well. Our DLSC scheme was implemented in MATLAB. We used the Manopt optimization toolbox [10] for implementing the CG method for our DL subproblem. We found that initializing the dictionary learning setup using K-Means clustering (using the Karcher mean algorithm [40]) led to faster convergence of CG.

3.3.2 Results

We compare below performance of our Riem-DL and Riem-SC against several prior DLSC schemes on the three datasets described above. In particular, we compare (i) Riemannian geometric methods such as log-Euclidean (LE-DL + LE-SC), (ii) Kernelized methods using the Stein kernel (Kernel-Stein-DL and kernel-Stein-SC), (iii) Euclidean DLSC (Frob-DL + Frob-SC), and using a dictionary generated by random sampling the dataset followed by sparse coding using our Riemannian method (Random-DL + Riem-SC). In Figure 4, we show the performance comparison for the task of K-NN where K is increased from 1 to 25.

A commonly adopted alternative to dictionary learning is to approximate the dictionary using centroids of a K-Means clustering of the dataset. Such a method is faster than Riemannian DL, and also demonstrate reasonable performance [13, 46]. Thus, an important experiment is to ensure that learning the dictionary actually provides superior performance compared to the *ad hoc* clustering setup. In Figure 3, we plot the K-NN retrieval when we use a clustering scheme to generate the dictionary.

3.3.3 Discussion of Results

With regard to Figure 4, we found that the performance of different methods is diverse across datasets. For example, the log-euclidean DLSC variant (LE-DL+LE-

SC) is generally seen to show good performance, but its performance is inferior when the number of data instances per class is small (as in the ETHZ people dataset). The kernelized DLSC method (Kernel-Stein-DL) performs favorably on most datasets. The most surprising of the results that we found was for Frob-DL. It is generally assumed that using Frobenius distance for comparing SPD matrices leads to poor accuracy, a view echoed by Figures 4(a) and 4(b). However, when the matrices are ill-conditioned, taking the logarithm (as in the LE-DL scheme) of these matrices results in amplifying the influence of the smaller eigenvalues, which is essentially noise. When learning a dictionary, the atoms will be learned to reconstruct this noise against the signal, thus leading to inferior performance than for FrobDL which do not use the logarithm. In comparison to all the compared methods, Riem-DL+Riem-SC was found to produce consistent and competitive performance, substantiating the usefulness of our model. While running the experiments, we found that the initialization of our DL sub-problem (using Riemannian K-Means) played an important role in achieving this superior performance.

We further compare Riem-DL against alternative DL schemes via clustering in Figure 3. We see that learning the dictionary using Riem-DL demonstrates the best performance against the next best and efficient alternative of using LE-KMeans as was done in [13]. Using Frob-KMeans or using a random dictionary are generally seen to have inferior performance compared to other learning methods.

3.4 GDL versus Riemannian Sparse Coding

Finally, we compare sparse coding via our GDL model and the Riem-SC setup. We use the Brodatz and RGB-D Object recognition datasets. Tables 2 and 3 show the results. While GDL is significantly faster in sparse coding, as is clear from the table, the Riemannian approach leads to much higher accuracy.

Method	Accuracy (%)
Frob-SC	32.3 (4.4)
TSC [46]	35.6 (7.1)
GDL	43.7 (6.3)
Riem-SC	**53.9** (3.4)

Table 2 Brodatz texture dataset

Method	Accuracy (%)
Frob-SC	80.3 (1.1)
TSC [46]	72.8 (2.1)
GDL	61.9 (0.4)
Riem-SC	**84.0** (0.6)

Table 3 RGB-D Object Recognition

Comparison of the average classification accuracy using a linear SVM.

4 Conclusion and Future Work

In this chapter, we reviewed the steps for constructing covariance descriptors, followed by a brief exposition of SPD matrix geometry motivated by the design of

novel machine learning models on these descriptors. We covered the concrete problems of dictionary learning and sparse coding, and noted two approaches: (i) a framework that uses Euclidean embedding of SPD matrices for sparse coding; and (ii) a Riemannian geometric approach. Our experiments demonstrated that designing machine learning algorithms for SPD matrices that respect the Riemannian geometry fares significantly better than using Euclidean embedding.

That said, Riemannian optimization algorithms are usually computationally more expensive. This is mainly due to the need for operating in the tangent space of the SPD manifold, which involves matrix exponentials and logarithms that require $O(d^3)$ flops. Designing faster Riemannian machine learning algorithms is a challenge that needs to be addressed for these algorithms to be more widely accepted.

Acknowledgements AC is funded by the Australian Research Council Centre of Excellence for Robotic Vision (number CE140100016). SS acknowledges support from NSF grant IIS-1409802.

References

1. Absil, P.A., Baker, C.G., Gallivan, K.A.: Trust-region methods on riemannian manifolds. Foundations of Computational Mathematics **7**(3), 303–330 (2007)
2. Absil, P.A., Mahony, R., Sepulchre, R.: Optimization algorithms on matrix manifolds. Princeton University Press (2009)
3. Alexander, D.C., Pierpaoli, C., Basser, P.J., Gee, J.C.: Spatial transformations of diffusion tensor magnetic resonance images. IEEE Transactions on Medical Imaging **20**(11), 1131–1139 (2001)
4. Arsigny, V., Fillard, P., Pennec, X., Ayache, N.: Log-Euclidean metrics for fast and simple calculus on diffusion tensors. Magnetic Resonance in Medicine **56**(2), 411–421 (2006)
5. Barzilai, J., Borwein, J.M.: Two-Point Step Size Gradient Methods. IMA J. Num. Analy. **8**(1) (1988)
6. Bertsekas, D.P., Bertsekas, D.P.: Nonlinear Programming, second edn. Athena Scientific (1999)
7. Bhatia, R.: Positive Definite Matrices. Princeton University Press (2007)
8. Birgin, E., Martínez, J., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM Journal on Optimization **10**(4), 1196–1211 (2000)
9. Birgin, E.G., Martínez, J.M., Raydan, M.: Algorithm 813: SPG - Software for Convex-constrained Optimization. ACM Transactions on Mathematical Software **27**, 340–349 (2001)
10. Boumal, N., Mishra, B., Absil, P.A., Sepulchre, R.: Manopt, a matlab toolbox for optimization on manifolds. The Journal of Machine Learning Research **15**(1), 1455–1459 (2014)
11. Brodatz, P.: Textures: a photographic album for artists and designers, vol. 66. Dover New York (1966)
12. Cherian, A.: Nearest neighbors using compact sparse codes. In: International Conference on Machine Learning, pp. 1053–1061 (2014)
13. Cherian, A., Sra, S.: Riemannian sparse coding for positive definite matrices. In: European Conference on Computer Vision. Springer (2014)
14. Cherian, A., Sra, S.: Riemannian dictionary learning and sparse coding for positive definite matrices. arXiv preprint arXiv:1507.02772 (2015)
15. Cherian, A., Sra, S., Banerjee, A., Papanikolopoulos, N.: Jensen-bregman logdet divergence with application to efficient similarity search for covariance matrices. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(9), 2161–2174 (2013)

16. Dana, K., Van Ginneken, B., Nayar, S., Koenderink, J.: Reflectance and texture of real-world surfaces. ACM Transactions on Graphics (TOG) **18**(1), 1–34 (1999)
17. Dodero, L., Minh, H.Q., Biagio, M.S., Murino, V., Sona, D.: Kernel-based classification for brain connectivity graphs on the Riemannian manifold of positive definite matrices. In: International Symposium on Biomedical Imaging, pp. 42–45. IEEE (2015)
18. Elad, M., Aharon, M.: Image denoising via learned dictionaries and sparse representation. In: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, vol. 1, pp. 895–900. IEEE (2006)
19. Ess, A., Leibe, B., Gool, L.V.: Depth and appearance for mobile scene analysis. In: International Conference on Computer Vision. IEEE (2007)
20. Fano, U.: Description of states in quantum mechanics by density matrix and operator techniques. Reviews of Modern Physics **29**(1), 74–93 (1957)
21. Faraki, M., Harandi, M.: Bag of riemannian words for virus classification. Case Studies in Intelligent Computing: Achievements and Trends pp. 271–284 (2014)
22. Fehr, D., Cherian, A., Sivalingam, R., Nickolay, S., Morellas, V., Papanikolopoulos, N.: Compact covariance descriptors in 3d point clouds for object recognition. In: IEEE International Conference on Robotics and Automation, pp. 1793–1798. IEEE (2012)
23. Fehr, D.A.: Covariance based point cloud descriptors for object detection and classification. University Of Minnesota (2013)
24. Ferro-Famil, L., Pottier, E., Lee, J.: Unsupervised classification of multifrequency and fully polarimetric SAR images based on the H/A/Alpha-Wishart classifier. IEEE Transactions on Geoscience and Remote Sensing **39**(11), 2332–2342 (2001)
25. Guo, K., Ishwar, P., Konrad, J.: Action recognition using sparse representation on covariance manifolds of optical flow. In: Advanced Video and Signal Based Surveillance. IEEE (2010)
26. Harandi, M.T., Sanderson, C., Hartley, R., Lovell, B.C.: Sparse coding and dictionary learning for symmetric positive definite matrices: A kernel approach. In: European Conference on Computer Vision. Springer (2012)
27. Jayasumana, S., Hartley, R., Salzmann, M., Li, H., Harandi, M.: Kernel methods on the Riemannian manifold of symmetric positive definite matrices. In: Computer Vision and Pattern Recognition. IEEE (2013)
28. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. ICCV, October **1**, 3 (2009)
29. Lai, K., Bo, L., Ren, X., Fox, D.: A large-scale hierarchical multi-view RGB-D object dataset. In: International Conference on Robotics and Automation (2011)
30. Li, P., Wang, Q., Zuo, W., Zhang, L.: Log-euclidean kernels for sparse representation and dictionary learning. In: International Conference on Computer Vision. IEEE (2013)
31. Liu, C.: Gabor-based kernel PCA with fractional power polynomial models for face recognition. Pattern Analysis and Machine Intelligence **26**(5), 572–581 (2004)
32. Liu, D., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Mathematical programming **45**(1), 503–528 (1989)
33. Ma, B., Su, Y., Jurie, F.: BiCov: a novel image representation for person re-identification and face verification. In: British Machine Vision Conference (2012)
34. Mairal, J., Bach, F., Ponce, J.: Sparse modeling for image and vision processing. arXiv preprint arXiv:1411.3230 (2014)
35. Marčelja, S.: Mathematical description of the responses of simple cortical cells*. JOSA **70**(11), 1297–1300 (1980)
36. O. Tuzel, F. Porikli, and P. Meer.: Covariance Tracking using Model Update Based on Lie Algebra. Computer Vision and Pattern Recognition (2006)
37. O. Tuzel, F. Porikli, and P. Meer.: Region Covariance: A Fast Descriptor for Detection and Classification. In: European Conference on Computer Vision (2006)
38. Ojala, T., Pietikäinen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern recognition **29**(1), 51–59 (1996)
39. Pang, Y., Yuan, Y., Li, X.: Gabor-based region covariance matrices for face recognition. IEEE Transactions on Circuits and Systems for Video Technology **18**(7), 989–993 (2008)

40. Pennec, X., Fillard, P., Ayache, N.: A Riemannian framework for tensor computing. International Journal of Computer Vision **66**(1), 41–66 (2006)
41. Phillips, P., Moon, H., Rizvi, S., Rauss, P.: The FERET evaluation methodology for face-recognition algorithms. Pattern Analysis and Machine Intelligence **22**(10), 1090–1104 (2000)
42. Phillips, P., Wechsler, H., Huang, J., Rauss, P.: The FERET database and evaluation procedure for face-recognition algorithms. Image and Vision Computing **16**(5), 295–306 (1998)
43. Schmidt, M., van den Berg, E., Friedlander, M., Murphy, K.: Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm. In: International Conference on Artificial Intelligence and Statistics (2009)
44. Schwartz, W., Davis, L.: Learning Discriminative Appearance-Based Models Using Partial Least Squares. In: Proceedings of the XXII Brazilian Symposium on Computer Graphics and Image Processing (2009)
45. Shinohara, Y., Masuko, T., Akamine, M.: Covariance clustering on Riemannian manifolds for acoustic model compression. In: International Conference on Acoustics, Speech and Signal Processing (2010)
46. Sivalingam, R., Boley, D., Morellas, V., Papanikolopoulos, N.: Tensor sparse coding for region covariances. In: European Conference onComputer Vision. Springer (2010)
47. Somasundaram, G., Cherian, A., Morellas, V., Papanikolopoulos, N.: Action recognition using global spatio-temporal features derived from sparse representations. Computer Vision and Image Understanding **123**, 1–13 (2014)
48. Sra, S.: Positive Definite Matrices and the S-Divergence. Proceedings of the American Mathematical Society (2015). ArXiv:1110.1773v4
49. Sra, S., Cherian, A.: Generalized dictionary learning for symmetric positive definite matrices with application to nearest neighbor retrieval. In: European Conference on Machine Learning. Springer (2011)
50. Su, J., Srivastava, A., de Souza, F., Sarkar, S.: Rate-invariant analysis of trajectories on riemannian manifolds with application in visual speech recognition. In: Computer Vision and Pattern Recognition, pp. 620–627. IEEE (2014)
51. Tosato, D., Farenzena, M., Spera, M., Murino, V., Cristani, M.: Multi-class classification on Riemannian manifolds for video surveillance. In: European Conference on Computer Vision (2010)
52. Wang, R., Guo, H., Davis, L.S., Dai, Q.: Covariance discriminative learning: A natural and efficient approach to image set classification. In: Computer Vision and Pattern Recognition. IEEE (2012)
53. Zheng, W., Tang, H., Lin, Z., Huang, T.S.: Emotion recognition from arbitrary view facial images. In: European Conference on Computer Vision, pp. 490–503. Springer (2010)