A Riemannian Network for SPD Matrix Learning

Zhiwu Huang and Luc Van Gool
Computer Vision Lab, ETH Zurich, Switzerland
{zhiwu.huang, vangool}@vision.ee.ethz.ch

Abstract

Symmetric Positive Definite (SPD) matrix learning methods have become popular in many image and video processing tasks, thanks to their ability to learn appropriate statistical representations while respecting the Riemannian geometry of the underlying SPD manifold. In this paper we build a Riemannian network to open up a new direction of SPD matrix non-linear learning in a deep architecture. The built network generalizes the Euclidean network paradigm to non-Euclidean SPD manifolds. In particular, we devise bilinear mapping layers to transform input SPD matrices into more desirable SPD matrices, exploit eigenvalue rectification layers to introduce the non-linearity with a non-linear function on the new SPD matrices, and design eigenvalue logarithm layers to perform Log-Euclidean Riemannian computing on the resulting SPD matrices for regular output layers. For training the deep network, we propose a Riemannian matrix backpropagation by exploiting a variant of stochastic gradient descent on Stiefel manifolds where the network weights reside on. We show through experiments that the proposed SPD network can be simply trained and outperform existing SPD matrix learning and state-of-the-art methods in three typical visual classification tasks.

1. Introduction

Symmetric Positive Definite (SPD) matrices are often encountered in a variety of areas. In medical imaging, they are commonly used in diffusion tensor magnetic resonance imaging [1]. In visual recognition, due to the effectiveness of encoding data variations, SPD matrix-valued data have been shown to provide powerful statistical representations for images and videos. Examples include region covariance matrices for pedestrian detection [33], joint covariance descriptors for action recognition [15] and image set covariance matrices for face recognition [36].

As a consequence, there has been a growing need to carry out effective computations to interpolate, restore, and classify SPD matrix representations. However, the computations on SPD matrices often accompany with the challenge of their non-Euclidean data structure which underlies a specific Riemannian manifold [1, 2]. Applying the Euclidean geometry directly to SPD matrices often results in poor performances and undesirable effects, such as the swelling of diffusion tensors in the case of SPD matrices [1]. To address this problem, [1, 2, 31] introduced Riemannian metrics, e.g., affine-invariant metric [1] and Log-Euclidean metric [2], to encode the Riemannian geometry of SPD manifolds properly.

By employing these well-studied Riemannian metrics, existing SPD matrix learning approaches typically extend Euclidean algorithms to work on the underlying Riemannian manifold by either flattening it via tangent space approximation [34, 32, 10, 30] or mapping it into a high dimensional reproducing kernel Hilbert space [16, 36, 26]. To more faithfully respect the original Riemannian geometry, recently proposed SPD matrix discriminant learning methods [15, 17] pursue a manifold-to-manifold transformation mapping the original SPD manifold to another discriminative SPD manifold with the same geometry. Although these SPD matrix learning methods have reached some success, most of them adopt linear learning scheme on SPD matrices, which however reside on non-linear manifolds rather than vector spaces with usual additive structure. Therefore, this would inevitably lead to sub-optimal solutions for the problem of SPD matrix learning.

Deep neural networks have greatly surpassed traditional shallow linear learning architectures in many contexts in artificial intelligence and visual recognition. The power of deep networks stems both from their ability to perform non-linear computations, and from the effectiveness of the gradient-descent training procedure based on backpropagation. This motivates us to devise a deep neural network architecture for SPD matrix non-linear learning. Differing from most existing deep networks (e.g., [22, 21]) that deal with unstructured matrices (usually handle their vector forms), our designed network architecture takes SPD matrices directly as input, and yields new SPD matrices as output. In other words, this new network deeply learns SPD matrices on their underlying Riemannian manifolds in an end-to-end learning architecture. Accordingly, the key is-

sue of the proposed Riemannian network is how to generate valid SPD matrices through multiple hierarchical layers in a deep architecture. In the light of the recent works [15, 17] that advocate geometry-aware SPD matrix learning, we propose a convolution-like layer to transform the input SPD matrices by a bilinear mapping strategy with requiring the transformation matrices to be orthogonal and thus reside on compact Stiefel manifolds. In summary, this paper mainly brings the following three innovations:

- A novel Riemannian network architecture is introduced to deeply learn more desirable SPD matrices in a non-linear learning scheme. This opens a new direction of SPD matrix non-linear learning on Riemannian manifolds.
- The proposed network offers a generalization of the traditional neural network paradigm to non-Euclidean SPD manifolds. In other words, we provide an interesting paradigm of incorporating the well-established Riemannian structures into deep network architectures for compressing both of the data space and the weight parameter space.
- To optimize the weights performing transformations on SPD matrices during training the proposed network, a Riemannian matrix backpropagation is proposed by exploiting a stochastic gradient descent optimization algorithm on Stiefel manifolds.

2. Related Work

Deep neural networks have exhibited their great powers in machine learning problems where the processed data own a Euclidean data structure. In many contexts, however, one may be faced with data defined over coordinates which adhere to non-Euclidean geometries. To tackle manifold-like graph data other than regular Euclidean data, [9] presented a spectral formulation of convolutional networks by exploiting the notion of generalized (non shift-invariant) convolution that depends on the analogy between the classical Fourier transform and the Laplace-Beltrami eigenbasis. Following [9], a localized spectral network was proposed in [7] to non-Euclidean domains for the analysis of deformable shapes. In particular, [7] generalized the windowed Fourier transform to manifolds to extract the local behavior of some dense intrinsic descriptor. This is roughly acted as an analogy to patches in images. Similarly, [25] proposed a notion of geodesic convolution on non-Euclidean domains based on local geodesic system of coordinates to extract local patches on the shape manifold. This approach used a natural way of generalizing Euclidean networks to Riemannian manifolds, where convolutions are performed by sliding a window over the manifold, and local geodesic coordinates are used in place of image patches.

Stochastic gradient descent (SGD) has been the workhorse for optimizing deep neural networks. One of the most well-known SGD algorithms uses Euclidean gradients with a varying learning rate to optimize the weights of a deep network. Backpropagation is an algorithm for efficiently computing the gradient of the loss with respect to the parameters. In addition to standard backpropagation algorithms working scalar inputs, the two works [19, 14] also developed backpropagation algorithms directly on matrices. For example, [19] formulated matrix backpropagation as a generalized chain rule mechanism for computing derivatives of composed matrix functions with respect to matrix inputs by relying on the calculus of adjoint matrix variations. Besides, the other family of network optimization algorithms exploits Riemannian gradients to handle weight space symmetries in neural networks. Optimizing over a Riemannian manifold has been a topic of much research and provides guide to compute non-Euclidean gradients for parameter computations [3]. This kind of gradient pursuing methods commonly computes the steepest descent for weight update on a specific type of Riemannian manifold. For instance, recent works [8, 6, 29, 24] developed several network optimization algorithms by building Riemannian metrics on the activity and parameter space of neural networks, treated as Riemannian manifolds.

3. Riemannian SPD Matrix Network

The proposed SPD network (SPDNet) architecture performs deep learning on input SPD matrices, and outputs more appropriate SPD matrices. Taking the well-known convolutional network (ConvNet) as an analogy, the proposed network also designs convolution-like layers and rectified linear units (ReLU)-like layers, named bilinear mapping (BiMap) layers and eigenvalue rectification (ReEig) layers respectively. Analogously to the convolution layer in the ConvNet, the proposed BiMap layers perform transformations on input SPD matrices to generate new SPD matrices by adopting a bilinear mapping scheme. As the classical ReLU layers, the designed ReEig layers introduce the non-linearity to the SPDNet by rectifying the resulting SPD matrices with a non-linear function. Since SPD matrices reside on non-Euclidean manifolds, directly applying regular Euclidean output layers such as softmax layer on them would yield sub-optimal solutions. To address this issue, we devise eigenvalue logarithm (LogEig) layers to carry out Log-Euclidean Riemannian computing on the resulting SPD matrices. By exploiting the Riemannian computation, the SPD matrices are transformed into usual symmetric matrices, which lie in Euclidean space and thus can be fed into any Euclidean output layers. The proposed Riemannian network is illustrated in Fig.1.

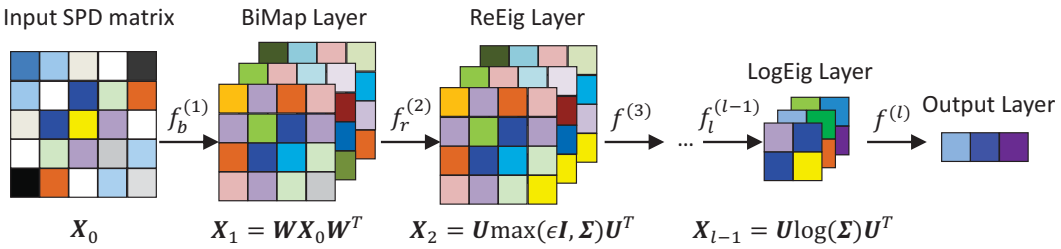

Figure 1. Conceptual illustration of the proposed SPDNet architecture.

3.1. BiMap Layer

The most primary function of the SPDNet is to generate more desirable SPD matrices on Riemannian manifolds. For this purpose, we design the BiMap layer (a convolution-like fully connected layer) to transform the input SPD matrices into new SPD matrices by a bilinear mapping f_b as

$$\boldsymbol{X}_k = f_b^{(k)}(\boldsymbol{X}_{k-1}; \boldsymbol{W}_k) = \boldsymbol{W}_k \boldsymbol{X}_{k-1} \boldsymbol{W}_k^T, \qquad (1)$$

where $\boldsymbol{X}_{k-1} \in Sym_{d_{k-1}}^+$ is the input SPD matrix of the k-th layer, $\boldsymbol{W}_k \in \mathbb{R}_*^{d_k \times d_{k-1}}, (d_k \leq d_{k-1})$ is the transformation matrix (connection weights), $\boldsymbol{X}_k \in \mathbb{R}^{d_k \times d_k}$ is the resulting matrix. To ensure the matrix $\boldsymbol{X}_k$ becomes a valid SPD matrix residing on another SPD manifold $Sym_{d_k}^+$, the transformation matrix $\boldsymbol{W}_k$ is basically required to be a row full-rank matrix. By applying the BiMap layer, the input matrices on the original SPD manifold $Sym_{d_{k-1}}^+$ are transformed into lower-dimensional SPD matrices, which also form a valid SPD manifold $Sym_{d_k}^+$. In other words, the data space on each BiMap layer corresponds to one SPD manifold.

Since the weight space $\mathbb{R}_*^{d_k \times d_{k-1}}$ of full-rank matrices on each BiMap layer is a non-compact Stiefel manifold where the distance function has no upper bound, directly optimizing on the manifold is infeasible. To handle this problem, we additionally assume the transformation matrix $\boldsymbol{W}_k$ to be orthogonal so that they reside on a compact Stiefel manifold $St(d_k, d_{k-1})$ [1]. As a result, optimizing over the compact Stiefel manifolds can achieve the optimal solution of the transformation matrices.

3.2. ReEig Layer

In the context of ConvNets, [20, 28] have presented various rectified linear units (ReLU) (including the $\max(0, x)$ non-linearity) to improve discriminative performance. Hence, exploiting ReLU-like layers to introduce the non-linearity to the domain of SPDNet is also necessary. Inspired by the idea of the $\max(0, x)$ non-linearity, we devise the ReEig layer to rectify the SPD matrices by tuning up their small positive eigenvalues. Intuitively, this rectification would prevent the SPD matrices from being non-positive as well. Nevertheless, it is not originally designed for regularization because the input matrices for ReEig layers are already non-singular after the operation of BiMap layers. Accordingly, we formulate the rectification function f_r on a SPD matrix in the k-th layer as

$$\boldsymbol{X}_k = f_r^{(k)}(\boldsymbol{X}_{k-1}) = \boldsymbol{U} \max(\epsilon \boldsymbol{I}, \boldsymbol{\Sigma}) \boldsymbol{U}^T, \qquad (2)$$

where $\boldsymbol{U}, \boldsymbol{\Sigma}$ are the eigenvector and eigenvalue matrices achieved by singular value decomposition (SVD) on the input SPD matrix as $\boldsymbol{X}_{k-1} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^T$, ϵ is the small rectification threshold, $\max(\epsilon \boldsymbol{I}, \boldsymbol{\Sigma})$ is the diagonal matrix of the max operations defined as

$$\max(\epsilon \boldsymbol{I}, \boldsymbol{\Sigma})_{ii} = \begin{cases} \boldsymbol{\Sigma_{ii}}, & \boldsymbol{\Sigma}_{ii} > \epsilon, \\ \epsilon, & \boldsymbol{\Sigma}_{ii} \leq \epsilon. \end{cases} \qquad (3)$$

Besides, there also exist other feasible strategies to derive a rectifying non-linearity on the input SPD matrices. For example, the noisy version of Eqn.2 can be applied by $\boldsymbol{X}_k = \boldsymbol{U} \max(\epsilon \boldsymbol{I}, \boldsymbol{\Sigma} + N(0, \sigma(\boldsymbol{\Sigma})))\boldsymbol{U}^T$, where $N(0, \sigma(\boldsymbol{\Sigma}))$ is Gaussian noise with zero mean and variance of $\boldsymbol{\Sigma}$. Due to the space limitation, we do not discuss this any further in this paper.

3.3. LogEig Layer

The LogEig layer is designed to perform Riemannian computing on the resulting SPD matrices for output layers with objective functions. As studied in [2], the Log-Euclidean Riemannian metric is able to endow the Riemannian manifold of SPD matrices with a Lie group structure so that the manifold is reduced to a flat space with the matrix logarithm operation $\log(\cdot)$ on the SPD matrices. In the flat space, classical Euclidean computations can be applied to the domain of SPD matrix logarithms. Formally, we apply the Log-Euclidean Riemannian computation [2] on a SPD matrix in the k-th layer with the function f_l defined as

$$\boldsymbol{X}_k = f_l^{(k)}(\boldsymbol{X}_{k-1}) = \log(\boldsymbol{X}_{k-1}) = \boldsymbol{U} \log(\boldsymbol{\Sigma}) \boldsymbol{U}^T, \qquad (4)$$

where $\boldsymbol{X}_{k-1} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^T$, $\log(\boldsymbol{\Sigma})$ is the diagonal matrix of the eigenvalue logarithms.

[1] A compact Stiefel manifold $St(d_k, d_{k-1})$ is the set of d_k-dimensional orthonormal matrices of the $\mathbb{R}^{d_{k-1}}$.

For SPD manifolds, the Log-Euclidean Riemannian computation is particularly simple to use and avoids the high expense of other Riemannian computations [1, 31], while preserving favorable theoretical properties. As for other Riemannian computations on SPD manifolds, please refer to [1, 31] for more detailed discussion on their properties.

3.4. Other Layers

After applying the LogEig layer, the outputs (i.e., SPD matrix logarithms) lie in Euclidean space and thus can be converted into vector forms. Therefore, on the top of the LogEig layer, classical Euclidean network layers can be applied. For example, the Euclidean fully connected (FC) layer could be built after the LogEig layer. The dimensionality of the filters in the FC layer can be set to $\mathbb{R}^{d_k \times d_{k-1}}$, where d_k and d_{k-1} are the class number and the dimensionalities of the vector forms of the input symmetric matrices respectively. The final output layer for visual recognition tasks could be softmax layer or softmax log-loss layer used in the context of Euclidean networks.

In addition, the pooling layer and normalization layer are also important to improve Euclidean ConvNets. For the SPDNet, the pooling on SPD matrices can be first carried out on their matrix logarithms, and then transform them back to SPD matrices by employing the matrix exponential map $\exp(\cdot)$ in the Riemannian framework [1, 2]. Similarly, the normalization procedure on SPD matrices could be first to calculate the mean and variance of their matrix logarithms, and then normalize them with their mean and variance as done in [18]. The normalized matrices are finally mapped back by using the operation $\exp(\cdot)$ to the SPD manifold to get the normalized SPD matrices.

4. Riemannian Matrix Backpropagation

Similar to traditional Euclidean networks, the model of the proposed SPDNet can be written as a series of successive function compositions $f = f^{(l)} \circ f^{(l-1)} \circ f^{(l-2)} \ldots \circ f^{(2)} \circ f^{(1)}$ with parameter tuple $\boldsymbol{W} = (\boldsymbol{W}_l, \boldsymbol{W}_{l-1}, \ldots, \boldsymbol{W}_1)$, where $f^{(k)}$ is the function for the k-th layer, $\boldsymbol{W}_k$ is the weight parameter of the k-th layer and l is the number of layers. The loss of the k-th layer could be denoted by a function as $L^{(k)} = \ell \circ f^{(l)} \circ \ldots f^{(k)}$, where ℓ is the loss function for the final output layer.

Training deep networks often uses stochastic gradient descent (SGD) algorithms. The key operation of one classical SGD algorithm is to compute the gradient of the objective function, which is obtained by an application of the chain rule known as backpropagation (backprop). For the k-th layer, the gradients of the weight $\boldsymbol{W}_k$ and the data $\boldsymbol{X}_{k-1}$ can be respectively computed by backprop as

$$\frac{\partial L^{(k)}(\boldsymbol{X}_{k-1}, y)}{\partial \boldsymbol{W}_k} = \frac{\partial L^{(k+1)}(\boldsymbol{X}_k, y)}{\partial \boldsymbol{X}_k} \frac{\partial f^{(k)}(\boldsymbol{X}_{k-1})}{\partial \boldsymbol{W}_k}, \quad (5)$$

$$\frac{\partial L^{(k)}(\boldsymbol{X}_{k-1}, y)}{\partial \boldsymbol{X}_{k-1}} = \frac{\partial L^{(k+1)}(\boldsymbol{X}_k, y)}{\partial \boldsymbol{X}_k} \frac{\partial f^{(k)}(\boldsymbol{X}_{k-1})}{\partial \boldsymbol{X}_{k-1}}, \quad (6)$$

where y is the desired output, $\boldsymbol{X}_k = f^{(k)}(\boldsymbol{X}_{k-1})$. Eqn.5 is the gradient for updating $\boldsymbol{W}_k$, while Eqn.6 is necessary for computing the gradients in the layers below to update the involved parameters.

There exist two key issues for generalizing backprop to the context of the proposed Riemannian network for SPD matrices. The first one is updating the weights in the BiMap layers. As we force the weights to be on the Stiefel manifold, merely using Eqn.5 to compute their Euclidean gradients rather than Riemannian gradients in the procedure of backprop cannot yield valid orthogonal weights. While the gradients of the SPD matrices in the BiMap layers can be calculated by Eqn.6 as usual, computing those with SVD operations in the layers of ReEig and LogEig has not been well-solved by the traditional backprop. Thus, it is the second key issue for training the proposed network.

To tackle the first issue, we propose a new way of updating the weights occurred in Eqn.1 for the BiMap layers by exploiting a stochastic gradient descent setting on Stiefel manifolds. The steepest descent direction for the corresponding loss function $L^{(k)}(\boldsymbol{X}_{k-1}, y)$ with respect to $\boldsymbol{W}_k$ on the Stiefel manifold is the Riemannian gradient $\tilde{\nabla} L^{(k)}_{\boldsymbol{W}_k}$. To obtain it, the normal component of the Euclidean gradient $\nabla L^{(k)}_{\boldsymbol{W}_k}$ is subtracted to generate the tangential component (to the Stiefel manifold). Searching along the tangential direction takes the update in the tangent space of the Stiefel manifold. Then, such the update is mapped back to the Stiefel manifold with a retraction operation. For more details about the geometry of Stiefel manifolds and its retraction operation, the readers are referred to the works [12] and [3] (Page 45-48, 59). Consequently, an update of the weight $\boldsymbol{W}_k$ on the Stiefel manifold is of the following form

$$\tilde{\nabla} L^{(k)}_{\boldsymbol{W}_k} = \nabla L^{(k)}_{\boldsymbol{W}_k} - \nabla L^{(k)}_{\boldsymbol{W}_k} \boldsymbol{W}_k^T \boldsymbol{W}_k, \quad (7)$$

$$\boldsymbol{W}_k^{t+1} = \Gamma(\boldsymbol{W}_k^t - \lambda \tilde{\nabla} L^{(k)}_{\boldsymbol{W}_k}), \quad (8)$$

where Γ is the retraction operation, $\boldsymbol{W}_k^t$ is the current weight, λ is the learning rate, $\nabla L^{(k)}_{\boldsymbol{W}_k} \boldsymbol{W}_k^T \boldsymbol{W}_k$ is the normal component of the Euclidean gradient $\nabla L^{(k)}_{\boldsymbol{W}_k}$, which is computed by using Eqn.5 as

$$\nabla L^{(k)}_{\boldsymbol{W}_k} = 2 \frac{\partial L^{(k+1)}}{\partial \boldsymbol{X}_k} \boldsymbol{W}_k \boldsymbol{X}_{k-1}. \quad (9)$$

As for the second issue, we exploit the matrix generalization of backprop studied in [19] to compute the gradients of

the involved SPD matrices in the ReEig and LogEig layers. In particular, let $\mathcal{F}$ be a function describing the variations of the upper layer variables with respect to the lower layer variables, i.e., $d\boldsymbol{X}_k = \mathcal{F}(d\boldsymbol{X}_{k-1})$. By defining the function $\mathcal{F}$, a new version of the chain rule Eqn.6 for the matrix backprop is defined as

$$\frac{\partial L^{(k)}(\boldsymbol{X}_{k-1}, y)}{\partial \boldsymbol{X}_{k-1}} = \mathcal{F}^* \left(\frac{\partial L^{(k+1)}(\boldsymbol{X}_k, y)}{\partial \boldsymbol{X}_k} \right), \tag{10}$$

where $\mathcal{F}^*$ is a non-linear adjoint operator of $\mathcal{F}$, i.e., $a : \mathcal{F}(b) = \mathcal{F}^*(a) : b$, the matrix inner product $\boldsymbol{A} : \boldsymbol{B} = Tr(\boldsymbol{A}^T\boldsymbol{B})$.

Actually, both of the two functions Eqn.2 and Eqn.4 for the ReEig and LogEig layers involve the SVD operation $\boldsymbol{X}_{k-1} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^T$. Hence, we introduce a virtual spectral layer ($k^{'}$ layer) for the SVD operation. Applying the new chain rule Eqn.10, we can achieve

$$\begin{aligned}
&\frac{\partial L^{(k)}}{\partial \boldsymbol{X}_{k-1}} : d\boldsymbol{X}_{k-1} \\
&= \mathcal{F}^* \left(\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{U}} \right) : d\boldsymbol{X}_{k-1} + \mathcal{F}^* \left(\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{\Sigma}} \right) : d\boldsymbol{X}_{k-1} \\
&= \frac{\partial L^{(k^{'})}}{\partial \boldsymbol{U}} : \mathcal{F}\left(d\boldsymbol{X}_{k-1}\right) + \frac{\partial L^{(k^{'})}}{\partial \boldsymbol{\Sigma}} : \mathcal{F}\left(d\boldsymbol{X}_{k-1}\right) \\
&= \frac{\partial L^{(k^{'})}}{\partial \boldsymbol{U}} : d\boldsymbol{U} + \frac{\partial L^{(k^{'})}}{\partial \boldsymbol{\Sigma}} : d\boldsymbol{\Sigma},
\end{aligned} \tag{11}$$

where the two variations $d\boldsymbol{U}$ and $d\boldsymbol{\Sigma}$ are derived by the variation of the SVD operation $d\boldsymbol{X}_{k-1} = d\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^T + \boldsymbol{U}d\boldsymbol{\Sigma}\boldsymbol{U}^T + \boldsymbol{U}\boldsymbol{\Sigma}d\boldsymbol{U}^T$ as:

$$d\boldsymbol{U} = 2\boldsymbol{U}(\boldsymbol{P}^T \circ (\boldsymbol{\Sigma}^T\boldsymbol{U}^T d\boldsymbol{X}_{k-1}\boldsymbol{U})_{sym}), \tag{12}$$

$$d\boldsymbol{\Sigma} = (\boldsymbol{U}^T d\boldsymbol{X}_{k-1}\boldsymbol{U})_{diag}, \tag{13}$$

where $\circ$ is the Hadamard product, $\boldsymbol{A}_{sym} = \frac{1}{2}(\boldsymbol{A} + \boldsymbol{A}^T)$, $\boldsymbol{A}_{diag}$ is $\boldsymbol{A}$ with all off-diagonal elements being 0 (note that we also use these two denotations in the following), $\boldsymbol{P}$ is calculated by operating on the eigenvalues σ in $\boldsymbol{\Sigma}$:

$$\boldsymbol{P}_{ij} = \begin{cases} \frac{1}{\sigma_i^2 - \sigma_j^2}, & i \neq j, \\ 0, & i = j. \end{cases} \tag{14}$$

For more details to derive Eqn.12 and Eqn.13, the readers are referred to [19]. Plugging Eqn.12 and Eqn.13 into Eqn.11 gives the partial derivatives of the loss functions for the ReEig and LogEig layers:

$$\begin{aligned}
\frac{\partial L^{(k)}}{\partial \boldsymbol{X}_{k-1}} = &2\boldsymbol{U}\boldsymbol{\Sigma} \left(\boldsymbol{P}^T \circ \left(\boldsymbol{U}^T \frac{\partial L^{(k^{'})}}{\partial \boldsymbol{U}} \right) \right)_{sym} \boldsymbol{U}^T \\
&+ \boldsymbol{U} \left(\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{\Sigma}} \right)_{diag} \boldsymbol{U}^T,
\end{aligned} \tag{15}$$

where $\frac{\partial L^{(k^{'})}(\boldsymbol{X}_{k^{'}}, y)}{\partial \boldsymbol{U}}$ and $\frac{\partial L^{(k^{'})}(\boldsymbol{X}_{k^{'}}, y)}{\partial \boldsymbol{\Sigma}}$ can be obtained by the same derivation strategy used in Eqn.11. For the function Eqn.2 employed in the ReEig layers, its variation becomes $d\boldsymbol{X}_k = 2(d\boldsymbol{U}\max(\epsilon\boldsymbol{I}, \boldsymbol{\Sigma})\boldsymbol{U}^T)_{sym} + (\boldsymbol{U}\boldsymbol{Q}d\boldsymbol{\Sigma}\boldsymbol{U}^T)_{sym}$, and these two partial derivatives are computed by

$$\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{U}} = 2 \left(\frac{\partial L^{(k+1)}}{\partial \boldsymbol{X}_k} \right)_{sym} \boldsymbol{U}\max(\epsilon\boldsymbol{I}, \boldsymbol{\Sigma}), \tag{16}$$

$$\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{\Sigma}} = \boldsymbol{Q}\boldsymbol{U}^T \left(\frac{\partial L^{(k+1)}}{\partial \boldsymbol{X}_k} \right)_{sym} \boldsymbol{U}, \tag{17}$$

where $\max(\epsilon\boldsymbol{I}, \boldsymbol{\Sigma})$ is defined in Eqn.3, and $\boldsymbol{Q}$ is the gradient of $\max(\epsilon\boldsymbol{I}, \boldsymbol{\Sigma})$:

$$\boldsymbol{Q}_{ii} = \begin{cases} 1, & \boldsymbol{\Sigma}_{ii} > \epsilon, \\ 0, & \boldsymbol{\Sigma}_{ii} \leq \epsilon. \end{cases} \tag{18}$$

For the function Eqn.4 used in the LogEig layers, its variation is $d\boldsymbol{X}_k = 2(d\boldsymbol{U}\log(\boldsymbol{\Sigma})\boldsymbol{U}^T)_{sym} + (\boldsymbol{U}\boldsymbol{\Sigma}^{-1}d\boldsymbol{\Sigma}\boldsymbol{U}^T)_{sym}$. Then we calculate the following two partial derivatives:

$$\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{U}} = 2 \left(\frac{\partial L^{(k+1)}}{\partial \boldsymbol{X}_k} \right)_{sym} \boldsymbol{U}\log(\boldsymbol{\Sigma}). \tag{19}$$

$$\frac{\partial L^{(k^{'})}}{\partial \boldsymbol{\Sigma}} = \boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T \left(\frac{\partial L^{(k+1)}}{\partial \boldsymbol{X}_k} \right)_{sym} \boldsymbol{U}, \tag{20}$$

By mainly employing Eqn.7–Eqn.9 and Eqn.15–Eqn.20, the Riemannian matrix backprop for training the SPDNet can be realized. The convergence analysis of the used SGD algorithm on Riemannian manifolds follows the developments in [8, 6].

5. Discussion about Related Works

As claimed before, building the SPDNet structure is inspired by the geometry-aware SPD matrix learning idea (i.e. preserving SPD property) of two recent works [15, 17] and the paradigm of typical deep networks. Thus, the goal of the SPDNet, that mainly uses multiple BiMap layers with a softmax layer, is to deeply learn lower-dimensional and more discriminative (desirable) SPD matrices, which benefit the classification tasks. In addition, our SPDNet has designed ReEig layers to introduce the non-linearity during learning desirable SPD matrices. Consequently, the main advantages of the proposed SPDNet over the two works [15, 17] lie in the multiple-layer learning and non-linear learning schemes, both of which has shown great power in the field of deep learning on Euclidean data.

The work [19] proposed a deep network to introduce a LogEig-like map for second-order pooling in image based

visual tasks. However, [19] merely plugs one layer of covariance Log-Euclidean map into ConvNets starting from images. In contrast, our SPDNet exploits multiple layers for the operations on SPD matrices, including BiMap, ReEig and LogEig layers, to deeply learn the SPD matrices in the context of Riemannian manifolds. From another point of the view, the proposed work could be built on the top of the network [19] for deeper SPD matrix learning architecture that starts from images. The superiority of the proposed SPDNet will be further studied in the following experiments. Moreover, we must claim that our SPDNet with bilinear maps is still useful while the work [19] will totally break down when the processing data are not covariance matrices for images.

6. Experiments

We study the effectiveness of the proposed SPD network (SPDNet) by conducting evaluations for three popular visual classification tasks including emotion recognition, action recognition and face verification where SPD matrix representations have achieved successes. For the evaluations, the state-of-the-art SPD matrix learning methods are compared. They are Covariance Discriminative Learning (CDL) [36], CDL on approximate infinite-dimensional region covariance descriptors obtained by random Fourier feature (CDL-F) [13], Log-Euclidean Metric Learning (LEML) [17] and SPD Manifold Learning (SPDML) [15] that uses affine-invariant metric (AIM) [1] and stein divergence [31]. The Riemannian Sparse Representation (RSR) [16] for SPD matrices is also evaluated. In addition, we also evaluate the deep second-order pooling (DeepO2P) network [19] which merely plugs a covariance Log-Euclidean map layer into ConvNets starting from images. For all of them, we use their source codes from authors with tuning their parameters according to the original works. For our SPDNet, we build its architecture with 8 layers: $\boldsymbol{X}_0 \rightarrow f_b^{(1)} \rightarrow f_r^{(2)} \rightarrow f_b^{(3)} \rightarrow f_r^{(4)} \rightarrow f_b^{(5)} \rightarrow f_l^{(6)} \rightarrow f_f^{(7)} \rightarrow f_s^{(8)}$, where f_b, f_r, f_l, f_f, f_s indicate the BiMap, ReEig, LogEig, Euclidean fully connected and softmax log-loss layers respectively. The learning rate λ is fixed as $1e^{-2}$, the batch size is set to 30, the weights in BiMap layers are initialized as random semi-orthogonal matrices, and the rectification threshold ϵ is set to $1e^{-4}$. For training the SPDNet, we just use an i7-2600K (3.40GHz) PC without any GPUs.

6.1. Emotion Recognition

We use the Acted Facial Expression in Wild (AFEW) [11] dataset for the task of emotion recognition. To simulate natural expressions in unconstrained environments, the AFEW database collects 1,345 videos of facial expressions of actors in movies with close-to-real-world scenarios.

Method	Accuracy
STM-ExpLet [23]	31.73%
RSR-SPDML [15]	30.12%
DeepO2P [19]	28.54%
CDL [36]	31.81%
CDL-F [13]	30.46%
LEML [17]	25.13%
SPDML-AIM [15]	26.72%
SPDML-Stein [15]	24.55%
RSR [16]	27.49%
SPDNet-0-BiReLayer	26.32%
SPDNet-1-BiReLayer	29.12%
SPDNet-2-BiReLayers	31.54%
SPDNet-3-BiReLayers	**34.23%**

Table 1. Emotion recognition accuracies for the AFEW database.

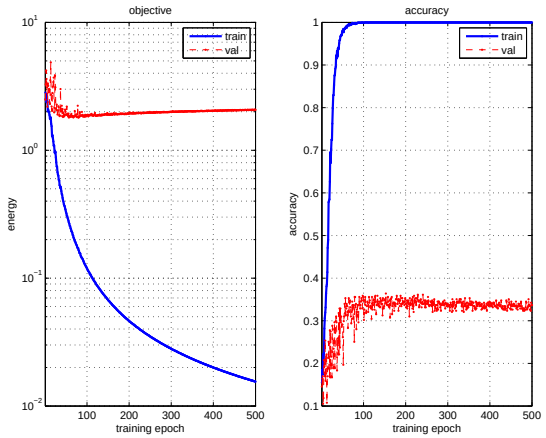

Figure 2. The convergence behavior (left) and accuracy curve (right) of the proposed SPDNet with ϵ being $1e^{-4}$ for the AFEW dataset.

According to the protocols of the Emotion Recognition in the Wild Challenge (EmotiW2014) [11], the database is divided into three data sets: training, validation and test. The task of this challenge is to classify each video sequence into one of seven expression categories. Since the ground truth of the test set has not been released, we here just report the results on the validation set for comparison. To augment the training data, we segment the training videos into 1,747 small video clips. For the evaluation, each facial frame is normalized to a gray-scale image of size 20×20. Then, following [36], we compute the image set covariance matrix of size 400×400 to represent each video sequence.

On the AFEW database, the dimensionalities of the SPDNet transformation matrices are set to $\mathbb{R}_*^{400\times200}, \mathbb{R}_*^{200\times100}, \mathbb{R}_*^{100\times50}$ respectively. Training the SPDNet per epoch (500 epoches in total) costs around 2 minutes(m) on this dataset by using the i7-

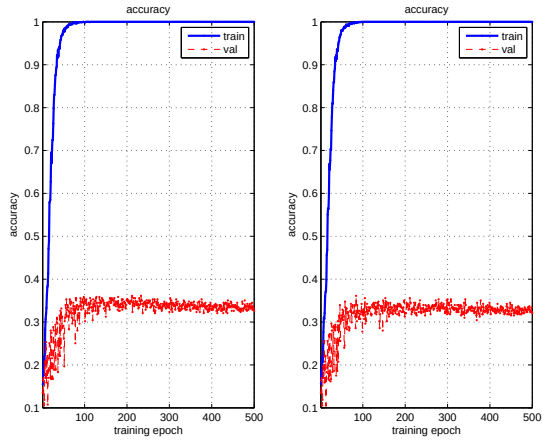

Figure 3. The accuracy curves of the proposed SPDNet with ϵ being $5e^{-5}$ (left) and 0 (right) respectively for the AFEW dataset.

2600K (3.40GHz) PC. As show in Table.1, we report the performances of the competing methods including the state-of-the-art method (STM-ExpLet [23]) on this database. It can be observed that our proposed SPDNet achieves several improvements over the state-of-the-art methods although the training data is small.

In addition, we study the performance of different configurations of the proposed SPDNet. First, we evaluate our SPDNet without using LogEig layers. For the AFEW database, its result 21.49% is extremely low. This shows the Riemannian computing is necessary as well studied in many works (e.g. [36, 16, 15, 17]). Second, we study the case of learning directly on Log-Euclidean forms of original SPD matrices by only using the tail of our SPDNet that begins from LogEig layer, FC layer to softmax layer. We denote the case as SPDNet-0-BiReLayer. The performance of the case is 26.32%, which is clearly outperformed by the whole SPDNet. This demonstrates the importance of using the SPD layers. Third, to study the effectiveness of using multiple BiMap and ReEig layers, we compute the results of using 1 or 2 BiMap/ReEig layer(s) (i.e., SPDNet-1-BiReLayer and SPDNet-2-BiReLayes) that feeds the LogEig layer with SPD matrices of the same size as set above. The performances of using 1 BiMap layer and 2 BiMap layers are 29.12% and 31.54% respectively, which is still surpassed by using 3 BiMap layers reported in Table.1. Fourth, the power of the designed rectification layer (ReEig) is also studied in Fig.3. As can be seen in Fig.2 (right) and Fig.3, the performances of the three different rectification threshold settings $\epsilon = 1e^{-4}, 5e^{-5}, 0$ are $34.23\%, 33.15\%, 32.35\%$ respectively, which demonstrates the non-linearity introduced in the ReEig layers is necessary to improve the discriminative power.

In the end, we also show the convergence behavior and accuracy curve of our SPDNet for the AFEW in Fig.2. This suggests that the proposed SPDNet is able to converge to a favorable solution after hundreds of iterations.

Method	Accuracy
RSR-SPDML [15]	48.01%±3.38
CDL [36]	41.74%±1.92
CDL-F [13]	46.25%±2.71
LEML [17]	46.87%±2.19
SPDML-AIM [15]	47.25%±2.78
SPDML-Stein [15]	46.21%±2.65
RSR [16]	41.12%±2.53
SPDNet-0-BiReLayer	48.12%±3.15
SPDNet-1-BiReLayer	55.26%±2.37
SPDNet-2-BiReLayers	59.13%±1.78
SPDNet-3-BiReLayers	**61.45%±1.12**

Table 2. Action recognition accuracies for the HDM05 database.

6.2. Action Recognition

We handle the problem of human action recognition from skeleton-based motion capture sequences using the HDM05 database [27]. This dataset contains 2,273 sequences of 130 motion classes in 10 to 50 realizations executed by various actors. The 3D locations of 31 joints of the subjects are provided over time acquired at the speed of 120 frames per second.

On the HDM05 dataset, we conduct 10 random evaluations, in each of which half of sequences are randomly selected for training, and the rest are used for testing. Note that the work [15] only recognizes 14 motion classes while the protocol designed by us is to identify 130 action classes and thus be more challenging. For data augmentation, the training sequences are divided into around 18,000 small sequences in each random evaluation. As done in [15], we represent each sequence by a joint covariance descriptor of size 93×93, which is computed by the second order statistics of 93-dimensional vectors concatenating the 3D coordinates of the 31 joints in each frame.

For our SPDNet, the sizes of the transformation matrices are set to $\mathbb{R}_*^{93\times70}, \mathbb{R}_*^{70\times50}, \mathbb{R}_*^{50\times30}$ respectively, and its training time at each of 500 epoches is about 4m on average. Table.2 summarizes the performances of the comparative algorithms and of the state-of-the-art method (RSR-SPDML) [15] on this dataset. As DeepO2P [19] is merely for image based visual classification tasks, we do not evaluate it in the 3D skeleton based action recognition task. We can see that our SPDNet outperforms the state-of-the-art shallow SPD matrix learning methods by a large margin (more than 13%). This shows that the proposed non-linear deep learning scheme on SPD matrices leads to great improvements when the training data is large enough. As for the studies of without using LogEig layers and different

Method	Accuracy-Con	Accuracy-Han
HERML-DeLF [5]	58.0%	59.0%
VGGDeepFace [35]	78.82%	68.24%
DeepO2P [19]	68.76%	60.14%
CDL [36]	78.29%	70.41%
CDL-F [13]	75.54%	67.23%
LEML [17]	66.53%	58.34%
SPDML-AIM [15]	65.47%	59.03%
SPDML-Stein [15]	61.63%	56.67%
SPDNet-0-BiReLayer	68.52%	63.92%
SPDNet-1-BiReLayer	71.75%	65.81%
SPDNet-2-BiReLayers	76.23%	69.64%
SPDNet-3-BiReLayers	**80.12%**	**72.83%**

Table 3. Face verification accuracies for the two experiments (i.e., control and handheld) of the PaSC database.

numbers of BiMap/ReEig layers are executed as the way of the last evaluation. As seen from Table.2, the performance of the case of without using LogEig layers is 4.89%, again validating the importance of using this kind of Riemannian computing layers. Besides, the same conclusions as the last evaluation for different settings of BiMap/ReEig layers can be drew.

6.3. Face Verification

For face verification, we employ the Point-and-Shoot Challenge (PaSC) database [4], which is a challenge for verifying faces in videos. It includes 1,401 videos taken by control cameras and 1,401 videos captured by handheld cameras for 265 people. Besides, it also contains 280 videos for training.

On the PaSC database, there are control-to-control and handheld-to-handheld face verification tasks, both of which are to verify a claimed identity in the query video by comparing with the associated target video. As done in [5], we also use the external training data (COX) with 900 videos. Similar to the last two experiments, the whole training data is also augmented to 12,529 small video clips. For evaluation, we use the approach of [35] to extract state-of-the-art deep face features from the normalized face images of size 224×224. Then we employ PCA to reduce the deep features to 400-dimensional features. Following [17], we compute a SPD matrix of size 401×401, which fuses the data covariance matrix and mean, for each video sequence of frames on this dataset.

For the evaluation of face verification, we configure the sizes of the SPDNet weights in BiMap layers to $\mathbb{R}_*^{401\times200}, \mathbb{R}_*^{200\times100}, \mathbb{R}_*^{100\times50}$ respectively. The time for training the SPDNet at each of 100 epoches is around 15m. Table.3 compares the accuracies of the different methods including the state-of-the-art methods (HERML-DeLF [5] and VGGDeepFace [35]) on the PaSC. Since the RSR method is designed for recognition tasks rather than verification tasks, we do not report its results. Although the used sofmax output layer in our SPDNet is not favorable for the verification tasks, we find that it still achieves state-of-the-art results. In the end, we can also obtain the same validations for the studies of without using LogEig layers (i.e., SPDNet-0-BiReLayer) and different numbers of BiMap/ReEig layers (i.e., SPDNet-1-BiReLayer and SPDNet-2-BiReLayers) as observed from Table.3.

7. Conclusion

We introduced a novel deep Riemannian network for opening up a possibility of SPD matrix non-linear learning. In the proposed deep network architecture, we mainly built BiMap layers, ReEig layers and LogEig layers in the context of SPD matrix deep learning on the underlying Riemannian manifolds. To train the proposed SPD network, we exploited a Riemannian matrix backpropagation by generalizing the SGD optimization algorithm to Stiefel manifolds. The evaluations on three typical visual classification tasks studied the effectiveness of the proposed network for SPD matrix learning.

As future work, it would be interesting to explore more effective layers, e.g., BiMap layers with multiple projections, pooling layer and normalization layer, to equip the new Riemannian network. Another interesting direction is to extend this work to a framework for a general Riemannian manifold by adopting the constrained transformations and the proposed Riemannian matrix backprop. In addition, we would also build the proposed SPD network on the top of existing convolution networks such as [19] for deeper architecture starting from images in an end-to-end deep learning manner.

References

[1] X. Pennec, P. Fillard, and N. Ayache. A Riemannian framework for tensor computing. *IJCV*, 2006.

[2] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM J. Matrix Analysis and Applications*, 29(1):328–347, 2007.

[3] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.

[4] J. Beveridge, J. Phillips, D. Bolme, B. Draper, G. Givens, Y. Lui, M. Teli, H. Zhang, W. Scruggs, K. Bowyer, P. Flynn, and S. Cheng. The challenge of face recognition from digital point-and-shoot cameras. In *BTAS*, 2013.

[5] J. Beveridge, H. Zhang, et al. Report on the FG 2015 video person recognition evaluation. In *FG*, 2015.

[6] S. Bonnabel. Stochastic gradient descent on Riemannian manifolds. *IEEE Trans. Auto. Control*, 2013.

[7] D. Boscaini, J. Masci, S. Melzi, M. Bronstein, U. Castellani, and P. Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. In *Computer Graphics Forum*, volume 34, pages 13–23. Wiley Online Library, 2015.

[8] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, 2010.

[9] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

[10] J. Carreira, R. Caseiro, J. Caseiro, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012.

[11] A. Dhall, R. Goecke, J. Joshi, K. Sikka, and T. Gedeon. Emotion recognition in the wild challenge 2014: Baseline, data and protocol. In *ICMI*, 2014.

[12] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[13] M. Faraki, M. T. Harandi, and F. Porikli. Approximate infinite-dimensional region covariance descriptors for image classification. In *ICASSP*, 2015.

[14] J. Gao, Y. Guo, and Z. Wang. Matrix neural networks. *arXiv:1601.03805v1*, 2016.

[15] M. Harandi, M. Salzmann, and R. Hartley. From manifold to manifold: Geometry-aware dimensionality reduction for SPD matrices. In *ECCV*, 2014.

[16] M. Harandi, C. Sanderson, R. Hartley, and B. C. Lovell. Sparse coding and dictionary learning for symmetric positive definite matrices: A kernel approach. In *ECCV*, 2012.

[17] Z. Huang, R. Wang, S. Shan, X. Li, and X. Chen. Log-Euclidean metric learning on symmetric positive definite manifold with application to image set classification. In *ICML*, 2015.

[18] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[19] C. Ionescu, O. Vantzos, and C. Sminchisescu. Training deep networks with structured layers by matrix backpropagation. *arXiv:1509.07838*, 2015.

[20] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, 2009.

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

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[23] M. Liu, S. Shan, R. Wang, and X. Chen. Learning expressionlets on spatio-temporal manifold for dynamic facial expression recognition. In *CVPR*, 2014.

[24] G. Marceau-Caron and Y. Ollivier. Practical Riemannian neural networks. *arXiv:1602.08007*, 2016.

[25] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *ICCV Workshops*, 2015.

[26] H. Minh, M. Biagio, and V. Murino. Log-Hilbert-Schmidt metric between positive definite operators on Hilbert spaces. In *NIPS*, 2014.

[27] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation: Mocap database HDM05. *Tech. Rep. CG-2007-2*, 2007.

[28] V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[29] Y. Ollivier. Riemannian metrics for neural networks I: Feedforward networks. *Infor. and Infer.*, 2015.

[30] A. Sanin, C. Sanderson, M. T. Harandi, and B. C. Lovell. Spatio-temporal covariance descriptors for action and gesture recognition. In *WACV Workshop*, 2013.

[31] S. Sra. Positive definite matrices and the s-divergence. *arXiv:1110.1773*, 2011.

[32] D. Tosato, M. Farenzena, M. Cristani, M. Spera, and V. Murino. Multi-class classification on Riemannian manifolds for video surveillance. In *ECCV*, 2010.

[33] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *ECCV*. 2006.

[34] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on Riemannian manifolds. *IEEE T-PAMI*, 30(10):1713–1727, 2008.

[35] O. P. A. Vedaldi and A. Zisserman. Deep face recognition. In *BMVC*, 2015.

[36] R. Wang, H. Guo, L. Davis, and Q. Dai. Covariance discriminative learning: A natural and efficient approach to image set classification. In *CVPR*, 2012.