Toward Faster and Simpler Matrix Normalization via Rank-1 Update

Tan Yu, Yunfeng Cai, and Ping Li

Cognitive Computing Lab
Baidu Research
10900 NE 8th St, Bellevue, WA 98004, USA
No. 10 Xibeiwang East Road, Beijing 100085, China
{tanyu01,caiyunfeng,liping11}@baidu.com

Abstract. Bilinear pooling has been used in many computer vision tasks and recent studies discover that matrix normalization is a vital step for achieving impressive performance of bilinear pooling. The standard matrix normalization, however, needs singular value decomposition (SVD), which is not well suited in the GPU platform, limiting its efficiency in training and inference. To resolve this issue, the Newton-Schulz (NS) iteration method has been proposed to approximate the matrix square-root. Although it is GPU-friendly, the NS iteration still takes several (expensive) iterations of matrix-matrix multiplications. Furthermore, the NS iteration is incompatible with the compact bilinear features obtained from Tensor Sketch (TS) or Random Maclaurin (RM). To overcome those known limitations, in this paper we propose a "rank-1 update normalization" (RUN), which only needs matrix-vector multiplications and is hence substantially more efficient than the NS iteration using matrix-matrix multiplications. Moreover, RUN readily supports the normalization on compact bilinear features from TS or RM. Besides, RUN is simpler than the NS iteration and easier for implementation in practice. As RUN is a differentiable procedure, we can plug it in a CNN-based an end-to-end training setting. Extensive experiments on four public benchmarks demonstrates that, for the full bilinear pooling, RUN achieves comparable accuracy with a substantial speedup over the NS iteration. For the compact bilinear pooling, RUN achieves comparable accuracy with a significant speedup over SVD-based normalization.

1 Introduction

Bilinear pooling has achieved excellent performance in many computer vision tasks, such as fine-grained recognition [22, 30, 20, 29, 19], generic image recognition [18], visual question answering [7, 36] and action classification [14, 31, 3, 28]. Recent studies [11, 15, 19, 20, 18] show that, the normalization on singular values of the bilinear matrix is vital for achieving high recognition performance. To be specific, DeepO^{2}P [11] adopts the logarithm normalization on the singular values to approximate the Log-Euclidean metric [1] for exploiting geometry of covariance spaces. MPN-COV [19] explains the advantage of the power normalization on singular values from the perspective of robust covariance estimation. In parallel, HoK [15], Improved BCNN [20], MHBN [35], Second-order Democratic Aggregation [21] and Power Normalization [16] demonstrate the importance of the normalization on singular values in remedying the burstiness phenomenon and equalizing contributions of singular values into the final image descriptor.

To conduct normalization on the singular values of the bilinear matrix, traditional methods such as matrix square-root normalization [19] and matrix logarithm normalization [11] rely on the singular value decomposition (SVD) to explicitly obtain the singular values. But SVD is not easily parallelizable and hence not well suited in the GPU platform, limiting its efficiency. To boost the efficiency in the GPU platform, improved B-CNN [20] attempts to approximate the matrix square root via the Newton-Schulz (NS) iteration [10] in the forward propagation. Since NS iteration only needs matrix-matrix product, it is easily parallelizable and well suited in the GPU platform in the inference time. But in the backward propagation of, the improved B-CNN uses Lyapunov equation, which is still expensive in computation. iSQRT [18] further makes the NS iteration differentiable and thus makes it feasible in the backward propagation. In each iteration, the NS method calculates several times of matrix-matrix multiplications. We denote the iteration number by K and denote the bilinear matrix by $\mathbf{B} \in \mathbb{R}^{D\times D}$, and NS method has a computation complexity of $\mathcal{O}(KD^3)$, which is expensive. In parallel, power-normalization methods are proposed in [16], which also rely on a series of matrix-matrix multiplications, taking a $\mathcal{O}(\log(\eta)D^3)$ computation complexity where η is the level of power normalization. In addition, NS iteration and power normalization only support normalization on the full bilinear matrix, and cannot support the normalization on the compact bilinear features [8] from Tensor Sketch [25] or Random Maclaurin [12]. This limits their usefulness in cases when low-dimension features are required.

To further improve the efficiency and overcome the limitation of the NS iteration, we propose a rank-1 update normalization (RUN). The proposed RUN is an iterative algorithm inspired by a classical numerical algorithm, power method [2]. In each iteration, it only needs twice matrix-vector multiplications, taking low computation cost. Meanwhile, it is easily parallelizable and well suited in the GPU platform. In each iteration, the computation complexity of the proposed RUN is $\mathcal{O}(DN)$. Here, N is the number of local features per image, which is in a comparable scale with D. In this case, the per-iteration cost of RUN, $\mathcal{O}(DN)$, is much lower than NS's per-iteration cost $\mathcal{O}(D^3)$. Moreover, in practice, our RUN needs less iterations to achieve the optimal recognition accuracy than NS method, making its efficiency advantage over the NS iteration more significant. Besides, the proposed RUN is much simpler than NS iteration and thus much easier for implementation in practice. In addition, the proposed RUN readily supports the normalization on a compact bilinear features generated from Tensor Sketch or Random Maclaurin. Thus, our RUN is especially useful for the cases when compact features are necessary. Power method is also used in obtaining the largest singular value for spectral normalization [24]. The spectral normalization is conducted on the weight matrix whereas ours is conducted on

Table 1. Differences between the proposed RUN and other bilinear pooling methods, O^2P [11], G^2DeNet [29], MPN-COV [19], Improved B-CNN [20], iSQRT [18], Power-Norm [16] and MoNet [9]. Here, K_1 is the number of iterations used in Newton-Schulz (NS) method, and K_2 is that in the proposed RUN, η is the level of power normalization, D is the local feature dimension and N is the number of local features. In practice, N is in a comparable scale with D, $K_2 < K_1$. Thus, the complexity $\mathcal{O}(K_2DN)$ of ours is significantly lower than the complexity $\mathcal{O}(K_1D^3)$ of the NS method. Meanwhile, our RUN is well suited in the GPU and readily supports compact bilinear pooling (CBP) based on Tensor Sketch or Random Maclaurin.

Method	Algorithm	Complexity	GPU Support	CBP Support
O^2P	SVD	$\mathcal{O}(D^3)$	limited	No
G^2DeNet	SVD	$\mathcal{O}(D^3)$	limited	No
MPN-COV	Eigen Decomp	$\mathcal{O}(D^3)$	limited	No
Improved B-CNN	Newton-Schulz in FP	$\mathcal{O}(D^3)$	good in FP	No
iSQRT	Newton-Schulz	$\mathcal{O}(K_1D^3)$	good	No
Power Normalization	MaxExp	$\mathcal{O}(\log(\eta)D^3)$	good	No
MoNet	SVD	$\mathcal{O}(D^3)$	limited	Yes
RUN (Ours)	Power Method	$\mathcal{O}(K_2DN)$	good	Yes

the feature. Table 1 compares our RUN with several bilinear pooling methods. In summary, our contributions in this paper are three-fold:

- We propose "rank-1 update normalization" (RUN), which only needs several matrix-vector multiplications and is considerably more efficient than existing GPU-friendly normalization methods based on the NS iteration.
- The proposed RUN readily supports normalization on compact bilinear features from Tensor Sketch or Random Maclaurin, which cannot be achieved by existing matrix normalization methods based on the NS iteration.
- The proposed RUN is a differentiable procedure. We plug it into a neural network and achieve an end-to-end training. The systematic experiments conducted on four public benchmarks demonstrate its excellence.

2 Background

We denote the feature map from a convolutional layer by $\mathcal{F} \in \mathbb{R}^{W\times H\times D}$, where W is the width, H is the height and D is the depth. We reshape $\mathcal{F}$ into a matrix $\mathbf{F} \in \mathbb{R}^{WH\times D}$. Bilinear pooling obtains the bilinear matrix by $\mathbf{B} = \mathbf{F}^\top\mathbf{F} \in \mathbb{R}^{D\times D}$. A pioneering work, B-CNN [22], implements the bilinear pooling as a layer of a convolutional neural network to support an end-to-end training. It achieves a better performance on fine-grained classification than standard AlexNet. The following research on bilinear pooling in deep neural network proceeds along two main directions: 1) improve the effectiveness of bilinear features [17, 29, 38, 20, 19]; 2) reduce the dimension of bilinear features [8, 5, 13]. Our work is related with both directions since we propose a fast matrix normalization method to boost its effectiveness, and make it compatible with compact bilinear pooling to obtain a compact and normalized bilinear feature. Below we review existing matrix normalization methods and compact bilinear pooling methods, respectively.

2.1 Matrix Normalization

We first review two traditional methods, matrix square-root normalization in improved B-CNN [20] and matrix logarithm normalization in O^{2}P [11]. They first conduct singular value decomposition (SVD) on the bilinear matrix $\mathbf{B}$ by

$$\mathbf{B} \rightarrow \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top.$$

Normalization is conducted on singular values and the normalized feature is

$$\hat{\mathbf{B}} \leftarrow \mathbf{U}\mathrm{g}(\mathbf{\Sigma})\mathbf{U}^\top,$$

where $\mathrm{g}(\mathbf{\Sigma})$ is conducted on singular values in an element-wise manner. Matrix square-root normalization adopts $\mathrm{g}(\mathbf{\Sigma}) = \mathbf{\Sigma}^{1/2}$ and matrix logarithm normalization adopts $\mathrm{g}(\mathbf{\Sigma}) = \log(\mathbf{\Sigma})$. However, as mentioned before, SVD is not easily parallelizable and not well supported in the GPU platform, limiting its efficiency in training and inference. Improved B-CNN [20] utilizes Newton-Schulz (NS) iteration to approximate the matrix square root in the forward propagation. iSQRT [18] makes the NS iteration differentiable and makes it support backward propagation as well. Given a bilinear matrix $\mathbf{B}$, the NS method initializes $\mathbf{Y}_0 = \mathbf{B}$ and $\mathbf{Z}_0 = \mathbf{I}$. In each iteration, the NS method updates $\mathbf{Z}_k$ and $\mathbf{Y}_k$ by

$$\mathbf{Y}_k = \frac{1}{2}\mathbf{Y}_{k-1}(3\mathbf{I} - \mathbf{Z}_{k-1}\mathbf{Y}_{k-1}), \quad \mathbf{Z}_k = \frac{1}{2}(3\mathbf{I} - \mathbf{Z}_{k-1}\mathbf{Y}_{k-1})\mathbf{Z}_{k-1},$$

where $\mathbf{Y}_k$ converges to $\mathbf{B}^{1/2}$. Since it involves only matrix-matrix product, it is easily parallelizable and well supported in the GPU platform. The computation complexity of each iteration is $\mathcal{O}(D^3)$, where D is the local feature dimension. Since D is large, computing Newton-Schulz (NS) iteration is still expensive. In contrast, our method is based on iterations of matrix-vector multiplications, which are computationally cheaper than the matrix-matrix multiplications used in the NS iteration. What's more, we will show in Section 2.2 that, the NS iteration is not compatible with compact bilinear pooling methods based on Random Maclaurin and Tensor Sketch, whereas ours readily supports normalization on compact bilinear pooling features.

2.2 Compact Bilinear Pooling

The dimension of a bilinear feature is $D \times D$, which is extremely high. On one hand, it is prone to over-fitting due to huge number of model parameters in the classifier, and thus requires a large number of training sample. On the other hand, it is extremely expensive in memory and computation when training the classifier. To overcome these drawbacks, CBP [8] is proposed. It treats the outer product used in bilinear pooling as a polynomial kernel embedding, and seeks to approximate the explicit kernel feature map. To be specific, by rearranging the feature map $\mathcal{F}$ to $\mathbf{F} = [\mathbf{f}_1, \cdots, \mathbf{f}_{WH}]^\top$, the bilinear matrix $\mathbf{B}$ is obtained by

$$\mathbf{B} = \mathbf{F}^\top \mathbf{F} = \sum_{i=1}^{WH} \mathbf{f}_i \mathbf{f}_i^\top = \sum_{i=1}^{WH} \mathrm{h}(\mathbf{f}_i),$$

where $\mathrm{h}(\mathbf{f}_i) = \mathbf{f}_i \mathbf{f}_i^\top \in \mathbb{R}^{D \times D}$ is the explicit feature map of the second-order polynomial kernel. CBP seeks for a low-dimensional projection function $\phi(\mathbf{f}_i) \in \mathbb{R}^d$ with $d \ll D^2$ such that

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \approx \langle \mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y})) \rangle,$$

where $\mathrm{vec}(\cdot)$ is the operation to unfold a 2D matrix to a 1D vector. In this case, the approximated low-dimensional bilinear feature is obtained by $\tilde{\mathbf{B}} = \sum_{i=1}^{WH} \phi(\mathbf{f}_i)$. The advantages of low-dimensional features are two-fold: 1) less prone to over-fitting, 2) faster in training the classifier.

CBP investigates two types of approximation methods: Random Maclaurin [12] and Tensor Sketch [25]. Since the compact bilinear feature $\tilde{\mathbf{B}}$ has broken the matrix structure, the matrix normalization methods conducted on the bilinear feature $\mathbf{B}$, such as Newton-Schulz iteration, is no longer feasible for normalizing $\tilde{\mathbf{B}}$. To tackle this challenge, MoNet [9] conducts SVD directly on the original local feature $\mathbf{F}$ instead of the bilinear matrix $\mathbf{B}$ and then conducts compact bilinear pooling. Nevertheless, as we mentioned, the SVD is not well supported on GPU platform, limiting the training and inference efficiency. In contrast, we will see in the next section that our method only relies on matrix-vector multiplications, and hence is easily parallelizable and well supported in the GPU platform. Meanwhile, the proposed RUN supports the normalization on a compact bilinear feature generated from Tensor Sketch or Random Maclaurin.

3 Rank-1 Update Normalization (RUN)

To overcome the limitations of previous methods, we propose a rank-1 update normalization (RUN). Below we give the details of the proposed RUN method and then summarize the method in Algorithm 1.

We first define some notations. Assuming that, through SVD, the bilinear feature $\mathbf{B}$ can be decomposed into $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$. $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_D]$ consists of singular vectors, which are orthogonal to each other. $\mathbf{\Sigma} = \mathrm{diag}([\sigma_1, \cdots, \sigma_D])$ is a diagonal matrix where $\{\sigma_d\}_{d=1}^D$ are singular values and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_D$. Next we introduce the process of the proposed RUN.

In the first step, we initialize a random vector $\mathbf{v}_0 = [v_1, ..., v_D] \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. That is, $\{v_i\}_{i=1}^D$ are *i.i.d.* random variables with standard normal distribution. Then, we perform K iterations of power method as follows:

$$\mathbf{v}_k = \mathbf{B}\mathbf{v}_{k-1}, \text{ for } k = 1, \ldots, K. \tag{1}$$

After that, the rank-1 matrix is constructed by

$$\mathbf{R}_K = \mathbf{B}\mathbf{v}_K\mathbf{v}_K^\top/\|\mathbf{v}_K\|_2^2.$$

At last, we update the matrix $\mathbf{B}$ by subtracting $\mathbf{R}_K$:

$$\mathbf{B}_K = \mathbf{B} - \epsilon\mathbf{R}_K, \tag{2}$$

where $\epsilon \in (0, 1]$ is a small positive constant. The classic convergence property of power method tells that if $\sigma_1 > \sigma_2$, $\mathbf{v}_K/\|\mathbf{v}_K\|_2$ will converge to $\mathbf{u}_1$. Therefore, $\mathbf{B}_K$ converges to $\mathbf{B} - \epsilon\sigma_1\mathbf{u}_1\mathbf{u}_1^\top$. That is

$$\lim_{K\to\infty} \mathbf{B}_K = \mathbf{U}\,\mathrm{diag}([\sigma_1(1-\epsilon), \sigma_2, \ldots, \sigma_D])\mathbf{U}^\top, \tag{3}$$

i.e., the eigenvalues of $\mathbf{B}_\infty$ remain unchanged except the largest one, which is decreased by $\epsilon\sigma_1$. More generally, $\mathbf{B}_K$ is an estimation of a normalized bilinear matrix. To be specific, it satisfies the following theorem:

Theorem 1. *Let $\mathbf{B}_K$ be obtained via Eq. (1)-(2), where $\mathbf{v}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Then the expectation of $\mathbf{B}_K$ is given by*

$$\mathbb{E}(\mathbf{B}_K) = \mathbf{U}\,\mathrm{diag}([\sigma_1(1-\epsilon\alpha_1), \cdots, \sigma_D(1-\epsilon\alpha_D))\mathbf{U}^\top, \tag{4}$$

where $1 \geq \alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_D$.

Due to limitation of the space, the proof of the Theorem 1 is given in the supplementary material. The operation in the right-hand side of Eq. (4) scales each singular value σ_i by $(1 - \epsilon\alpha_i)$. As $1 \geq \alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_D$ and $\epsilon \in [0, 1]$, thus

$$0 \leq 1 - \epsilon\alpha_1 \leq 1 - \epsilon\alpha_2 \leq \cdots \leq 1 - \epsilon\alpha_D \leq 1.$$

That is, it gives a smaller scale factor to a larger singular value, making the contributions of singular values to the final image feature more balanced and achieving the same goal as the spectral power normalization used in [20, 15].

Since computing $\mathbf{B}_K$ only requires $2K$ times matrix-vector multiplications, it only takes $\mathcal{O}(KD^2)$ complexity and is well supported in GPU platform. In experiments section, we will show when K is small, *e.g.*, $K = 2$, it has achieved an excellent performance. Nevertheless, obtaining the above approximated normalized bilinear feature $\mathbf{B}_K$ requires the original bilinear matrix $\mathbf{B}$ obtained from bilinear pooling. Thus, it is not applicable to the compact bilinear feature which has broken the structure of square matrix. To make the proposed fast matrix normalization method compatible with compact bilinear pooling, we seek to directly conduct normalization on the original feature map $\mathbf{F} \in \mathbb{R}^{N\times D}$, where $N = WH$ is the number of local features and D is the local feature dimension. It is based on following iterations:

$$\mathbf{v}_k = \mathbf{F}^\top\mathbf{F}\mathbf{v}_{k-1}, \text{for } k = 1, \ldots, K, \tag{5}$$

where the entries of $\mathbf{v}_0$ are *i.i.d.* random variables with standard normal distribution. Then we construct the updated feature map $\mathbf{F}_K$ by

$$\mathbf{F}_K = \mathbf{F} - \eta \mathbf{F}\mathbf{v}_K\mathbf{v}_K^\top / \|\mathbf{v}_K\|_2^2, \tag{6}$$

where $\mathbf{v}_K$ is obtained via (5) and $\eta \in (0, 1]$ is a constant. The above procedure is summarized in Algorithm 1. Since in each iteration, it only needs twice matrix-vector multiplications, in total, the computational complexity of obtaining $\mathbf{F}_K$ is $\mathcal{O}(KDN)$. Let $\mathbf{u}_{F,i}$ and $\mathbf{v}_{F,i}$ be the left and right singular vectors of $\mathbf{F}$ corresponding with its ith largest singular value $\sigma_{F,i}$. If $\sigma_{F,1} \neq \sigma_{F,2}$, $\mathbf{F}\mathbf{v}_k/\|\mathbf{v}_k\|_2$ and $\mathbf{v}_k/\|\mathbf{v}_k\|_2$ will converge to $\mathbf{u}_{F,1}$ and $\mathbf{v}_{F,1}$, respectively. In limit, we have

$$\lim_{K\to\infty} \mathbf{F}_K = \mathbf{F} - \eta\sigma_{F,1}\mathbf{u}_{F,1}\mathbf{v}_{F,1}^\top,$$

whose singular values are the same as that of $\mathbf{F}$, except the largest one, which is decreased by $\eta\sigma_{F,1}$. In fact, similar to Theorem 1, we have

Theorem 2. *Let $\mathbf{F}_K$ be obtained through Algorithm 1. Then the expectation of $\mathbf{F}_K$ can be given by*

$$\mathbb{E}(\mathbf{F}_K) = \mathbf{U}_F \widehat{\Sigma}_F \mathbf{V}_F^\top, \tag{7}$$

where $\mathbf{U}_F$, $\mathbf{V}_F$ are the left and right singular vector matrices of $\mathbf{F}$, respectively, $\widehat{\Sigma}_F$ is the diagonal matrix $\mathrm{diag}[(\sigma_{F,1}(1-\eta\beta_1), \cdots, \sigma_{F,D}(1-\eta\beta_D))]$ *with* $0 \le 1-\eta\beta_1 \le 1-\eta\beta_2 \le \cdots \le 1-\eta\beta_D \le 1$.

Its proof is similar to Theorem 1, and thus we omit it. Using the standard bilinear pooling, the normalized bilinear matrix feature can be obtained by $\bar{\mathbf{B}}_K = \mathbf{F}_K^\top\mathbf{F}_K$. When $\sigma_{F,1} \neq \sigma_{F,2}$, $\bar{\mathbf{B}}_K$ satisfies

$$\lim_{K\to\infty} \bar{\mathbf{B}}_K = \mathbf{V}_F\,\mathrm{diag}([\sigma_{F,1}^2(1-\eta)^2, \cdots, \sigma_{F,D}^2])\mathbf{V}_F^\top. \tag{8}$$

Since $\mathbf{V}_F$ in Eq. (8) is equal to $\mathbf{U}$ in Eq. (3), if we set $(1-\epsilon)$ in Eq. (8) equal to $(1-\eta)^2$ in Eq. (3), $\mathbf{B}_K$ and $\bar{\mathbf{B}}_K$ will converge to the same matrix. But the advantage of updating $\mathbf{F}$ as Eq. (6) over updating $\mathbf{B}$ as Eq. (2) is that, the former is compatible with compact bilinear pooling, which cannot be achieved by the latter. The compact normalized bilinear feature is obtained by

$$\bar{\mathbf{b}}_K = \sum_{i=1}^{N} \phi(\mathbf{F}_K[i,:]),$$

where $\mathbf{F}_K[i,:]$ is the i-th row of $\mathbf{F}_K$, ϕ is implemented by TS or RM, and $\bar{\mathbf{b}}_K \in \mathbb{R}^D$ is the compact and normalized bilinear feature where $D \ll d^2$.

The proposed RUN is summarized in Algorithm 1. We implement the proposed RUN as a layer of a CNN. The layer takes the feature map $\mathbf{F}$ as input and outputs the normalized feature map $\mathbf{F}_K$. In the forward propagation, $\mathbf{F}_K$ is computed by Eq. (6). After obtaining $\mathbf{F}_K$, it is feasible to conduct bilinear pooling (BP) or compact bilinear pooling (CBP). Fig. 1 illustrates the architecture of the proposed network. Note that, despite that one can rely on auto-grad tool in existing deep learning frameworks to automatically obtain the backward propagation based on the forward propagation, we still derive it in the supplementary material for readers to better understand the proposed algorithm.

Algorithm 1 Rank-1 Update Normalization (RUN).

```
Input: Local features $\mathbf{F} \in \mathbb{R}^{N \times D}, \eta, K$.
Output: Normalized local features $\mathbf{F}_K$.
1: Generate $\mathbf{v}_0 = [v_1, ..., v_D] \in \mathbb{R}^D$, where $\{v_i\}_{i=1}^D$ are i.i.d. random variables with
   normal distribution.
2: for $k \in [1, K]$ do
3:     $\mathbf{v}_k = \mathbf{F}^\top \mathbf{F} \mathbf{v}_{k-1}$.
4: $\mathbf{F}_K = \mathbf{F} - \eta \frac{\mathbf{F}\mathbf{v}_K \mathbf{v}_K^\top}{\|\mathbf{v}_K\|_2^2}$.
5: return $\mathbf{F}_K$.
```

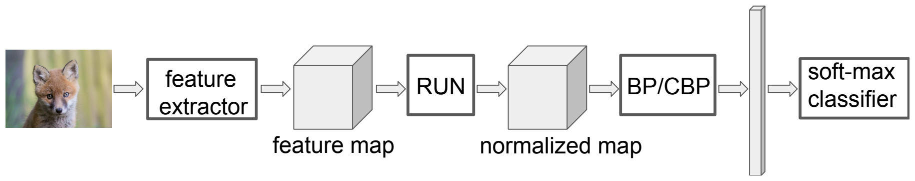

Fig. 1. The architecture of the proposed convolutional neural network. RUN denotes the proposed rank-1 update normalization, which takes input the feature map of the last convolutional layer. BP denotes the bilinear pooling and CBP represents compact bilinear pooling. The obtained BP/CBP feature is fed into the soft-max classifier.

4 Experiments

We first introduce the testing datasets and implementation details. Then we conduct ablation studies on two scenarios: 1) RUN with the standard bilinear pooling and 2) RUN with the compact bilinear pooling. After that, the comparisons with existing state-of-the-art bilinear pooling methods are conducted.

4.1 Datasets and Evaluation Metrics

Table 2. The number of training/testing samples of four datasets.

	Fine-grained		Scene	Texture
	CUB	Aircraft	MIT	DTD
classes	200	100	67	47
training	5,994	6,667	4,014	1,880
testing	5,794	3,333	1,339	3,760

As summarized in Table 2, we conduct experiments on three tasks: 1) fine-grained recognition, 2) scene recognition and 3) texture recognition. On the fine-grained recognition task, experiments are conducted on CUB [33] and Aircraft [23] datasets. On the scene recognition task, experiments are conducted on MIT [26] dataset. On the texture recognition task, we test our method on DTD [4] dataset. Since we tackle the recognition task, we evaluate the performance of the proposed method through the average classification accuracy.

4.2 Implementation Details

Following [20, 19], we use VGG16 [27] as the backbone network to make a fair comparison with existing methods. After scaling and cropping, the input size of an input image is $448 \times 448 \times 3$ and the size of the last convolutional feature map is $28 \times 28 \times 512$. After we obtain the normalized bilinear feature from our RUN, we further conduct element-wise signed square-root normalization followed by ℓ_2-normalization as BCNN [22]. Following [9], we adopt a two-phase training strategy. In the first phase, we update the weights of the last fully-connected linear layer and keep other layers unchanged. The initial learning rate is set as 0.2 on aircraft dataset and 1 on other datasets, and it decreases to 0.1 of current learning rate. We set weight decay as 10^{-8} in the first phase. The first phase finishes in 50 epochs. In the second phase, we update the weights of all layers and the initial learning rate is set as 0.02 on CUB dataset and 0.01 on other datasets, and it decreases to 0.1 of the current learning rate if the validation error does not drop in continuous 5 epochs. We set weight decay as 10^{-5} in the second phase. The second phase finishes in 40 epochs. We use paddlepaddle. Note that, iSQRT [18] pre-trains the network on ImageNet dataset.

4.3 Ablation Study on RUN with Original Bilinear Pooling

In this section, we test the proposed RUN using original high-dimensional bilinear pooling features. The feature dimension is $512 * 512 = 262$K.

Influence of η. η in Eq. (8) controls the strength of suppressing the large singular values. Recall from Eq. (8) that, $\bar{\mathbf{B}}_K$ converges to:

$$\mathbf{V}^F \mathrm{diag}[(1-\eta)^2\sigma_{F,1}^2, \cdots, \sigma_{F,d}^2]\mathbf{V}_F^\top.$$

From the above equation, we observe that, when $\eta \in (2, +\infty) \cup (-\infty, 0)$, the largest value of the normalized bilinear matrix $\bar{\mathbf{B}}_K$ is even larger than that of the original bilinear matrix $\mathbf{B}$. Hence a good value of η should be in the range $[0, 2]$. Ideally, we can select the value of η according to the gap between $\sigma_{F,1}$ and $\sigma_{F,2}$. Since singular values change for different samples or different epochs, we can compute $\sigma_{F,1}$ and $\sigma_{F,2}$ online for each sample in each epoch. But computing $\sigma_{F,1}$ and $\sigma_{F,2}$ will double the time cost compared with using a manually set η which only needs compute $\sigma_{F,1}$. Thus, we simply use a manually set η.

As shown in Table 3, when $\eta = 0$, *i.e.*, without RUN, the accuracies are not as good as that when $\eta \in [0.4, 1.5]$. Note that, on Aircraft dataset, the accuracy gap between the case when $\eta \in [0.4, 1.5]$ and the case when $\eta = 0$ is not significant. This is due to that the value of $\sigma_{F,1}/\sigma_{F,2}$ on Aircraft dataset (shown in Table 4) is small. It means that, the singular values are already balanced and thus the

Table 3. The influence of η on the proposed RUN.

η	CUB	Aircraft	MIT	DTD
0.0	84.1	88.9	79.8	65.6
0.1	84.8	89.3	80.6	66.6
0.2	85.3	89.5	**81.0**	67.8
0.4	86.0	89.6	80.5	68.3
0.6	86.3	**89.8**	80.8	**68.4**
0.8	86.2	89.7	80.7	**68.4**
1.0	**86.4**	**89.8**	80.9	68.3
1.2	86.0	**89.8**	80.9	68.2
1.5	86.2	89.7	80.5	68.3
2.0	83.9	89.0	79.7	65.7

Table 4. The average $\sigma_{F,1}/\sigma_{F,2}$ on four datasets.

	CUB	Aircraft	MIT	DTD
first epoch	2.23	1.53	2.38	5.09
last epoch	3.68	1.69	4.53	7.40

matrix normalization does not play an important role. In contrast, on DTD dataset, the accuracy gap is considerable, it is also in accordance with the large value of $\sigma_{F,1}/\sigma_{F,2}$ on DTD dataset as shown in Table 4.

As Table 4 shows that the average value of $\sigma_{F,1}/\sigma_{F,2}$ varies significantly on four datasets, thus we might expect that the optimal η are different on four dataset. Surprisingly, as shown in Table 3, when $\eta \in [0.4, 1.5]$ the performance is stable and not sensitive to the change of η. That is, in practice, the choice of η is quite easy for the user. By default, we set $\eta = 0.6$ on all datasets. Another observation is that, when $\eta = 2.0$, its performance is as bad as that when $\eta = 0.0$. The bad performance when $\eta = 2.0$ is expected since it leads to the condition that $(1-\eta)^2 = 1$. It is equivalent to removing the matrix normalization.

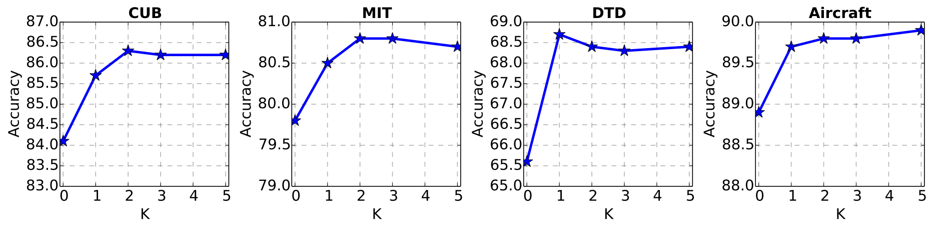

Fig. 2. The influence of K on the proposed RUN.

Influence of K. K in Eq. (6) represents the number of iterations in our RUN. The time cost of the proposed RUN is linear with K. Recall from Eq. (8) that, when K is large, the normalization focuses only on the largest singular value and keeps the others unchanged. In contrast, if K is not large, it also normalizes other large singular values besides the largest one. As shown in Fig. 2, on CUB, MIT and DTD datasets, our RUN achieves the optimal accuracy within 2 iterations. In contrast, on Aircraft dataset, it achieves the best accuracy with 5 iterations. But using 2 iterations, the accuracy on Aircraft dataset is comparable with that using 5 iterations. By default, we set $K = 2$ on all datasets.

Table 5. Comparisons with Lyapunov equation [20] and Newton-Schulz (NS) iteration.

Algorithm	FLOPs	GPU Time	Accuracy			
			CUB	Aircraft	MIT	DTD
Lyapunov equation	8.83G	1841ms	85.8	88.5	80.6	68.4
NS iteration	4.03G	833ms	85.7	89.6	80.5	68.3
RUN (ours)	3.2M	2.5ms	86.3	89.8	80.8	68.4

Time cost evaluation. We compare the time cost in matrix normalization in the GPU platform of the proposed method with existing methods based on Lyapunov equation [20] and Newton-Schulz (NS) iteration. We conduct experiments based on 4 NVIDIA K40 GPU cards and set the batch size as 32. As shown in Table 5, the FLOPs of ours is less than 0.1% of the NS iteration. Meanwhile, considering the GPU time, the factual speed-up ratio of ours over the NS iteration is beyond 330. The significant reduction in FLOPs and GPU time is contributed by two factors. Firstly, in each iteration, we only need two matrix-vector multiplications whereas NS iteration takes three times of matrix-matrix multiplications. Secondly, ours takes only 2 iterations for a good performance whereas NS iteration takes 5 iterations to achieve a good performance suggested by [18]. Note that, iSQRT [18] reduces the dimension of local convolutional features from 512 to 256 through a convolution layer, reducing the computation cost of the NS iteration. Our RUN can also be faster using the 256-dimension features, but that is not the focus of this paper.

4.4 Ablation Study on RUN with Compact Bilinear Pooling

Influence of the dimension. We adopt two types of CBP, Tensor Sketch (TS) and Random Maclaurin (RM). We set $\eta = 0.6$ and iteration number $K = 2$, and change the dimension after CBP among $\{1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 10\text{K}\}$. As shown in Fig. 3, the accuracies generally increase as the dimension increases. It is expected since a larger dimension leads to a better approximation for the polynomial kernel. Meanwhile, the accuracies achieved by TS are comparable with that achieved by RM. By default, we use TS for compact bilinear pooling.

Time cost evaluation. We evaluate the time cost used in matrix normalization for compact bilinear pooling (CBP). Since the Lyapunov equation and Newton-Schulz iteration cannot be conducted on the original feature $\mathbf{F}$, it is incompatible

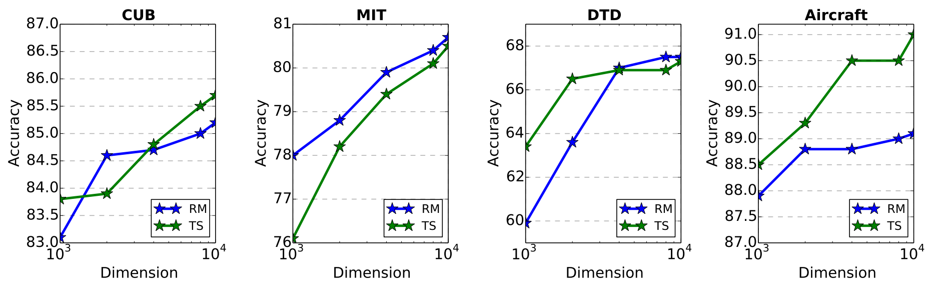

Fig. 3. The influence of the feature dimension of compact features from RM and TS.

with CBP. Thus, we only compare with Monet-2 [9] which conducts SVD on $\mathbf{F}$. $\mathbf{F} \in \mathbb{R}^{784\times512}$ is in a larger size than $\mathbf{B} \in \mathbb{R}^{512\times512}$. Meanwhile, $\mathbf{B}$ is symmetric and only needs compute its left singular vectors $\mathbf{U}$ as well as the singular values $\mathbf{\Sigma}$. But $\mathbf{F}$ is asymmetric and thus needs compute its right singular vectors $\mathbf{V}_F$, left singular vectors $\mathbf{U}_F$ and singular values $\mathbf{\Sigma}_F$. Therefore, the FLOPs of computing SVD on $\mathbf{F}$ shown in Table 6 is larger than the FLOPs of computing SVD on $\mathbf{B}$ shown in Table 5. In contrast, the FLOPs of our RUN used for CBP is as the same as that used for original BP. As shown in Table 6, achieving comparable or even better accuracies, we reduce the FLOPs from 4.21G to 3.2M. Moreover, we reduce the time cost in the GPU from 13850ms to 2.5ms, *i.e.*, we achieve a 5540$\times$ speedup. Note that, the GPU time cost speedup is larger than the FLOPs reduction ratio since the proposed RUN better supported than SVD in the GPU platform. We also test the time cost of SVD using CPU with 16 threads, it takes 7.6s, which is still much slower than our RUN using only 2.5ms.

Table 6. Comparisons between ours and MoNet-2 [9].

Method	Algorithm	Dimension	FLOPs	GPU Time	Accuracy	
					CUB	Aircraft
MoNet-2 [9]	SVD	10,000	4.21G	13850ms	85.7	86.7
Ours	power method	10,000	3.2M	2.5ms	85.7	91.0

4.5 Comparison with other pooling methods

Firstly, we compare with two baselines, which replace bilinear pooling by max-pooling and sum-pooling, respectively. As shown in Table 7, features from max-pooling and sum-pooling are compact. But accuracies achieved by them are lower than methods based on bilinear pooling. We further compare with B-CNN [22]. Benefited from bilinear pooling, B-CNN has achieved a good performance but the obtained bilinear features are high-dimensional. Nevertheless, since there is no matrix normalization, the performance of B-CNN is not as good as its

Table 7. Comparisons with other pooling methods. We compare the feature dimension, the algorithm as well as the time cost for matrix normalization per batch (Normalization Algorithm/Time) and the accuricies on four public benchmarks.

Method	Dimension	Normalization Method/Time	CUB	Aircraft	MIT	DTD
Max-pooling	512	-	69.6	78.9	50.4	55.1
Sum-pooling	512	-	71.7	82.1	58.7	58.2
BCNN [22]	262K	-	84.0	84.1	–	–
Improved BCNN [20]	262K	SVD/6.3s	85.8	88.5	–	–
BCNN + NS	262K	NS/833ms	85.7	89.6	80.5	68.3
iSQRT [18]	32K	NS/107ms	87.2	90.0	–	–
CBP [8]	8.2K	-	84.0	–	76.2	64.5
LRBP [13]	8.2K	-	84.2	87.3	–	65.8
MoNet-2 [9]	10K	SVD/13.9s	85.7	86.7	–	–
MoNet [9]	10K	SVD/13.9s	86.4	89.3	–	–
KP [5]	12.8K	-	86.2	86.9	–	–
HBP [34]	24K	-	87.1	90.3	–	–
GP [32]	4K	SVD/15.6s	85.8	89.8	–	–
DeepKSPD [6]	262K	SVD/6.3s	86.5	91.0	**81.0**	–
DBTNet-50 [37]	2K	–	**87.5**	**91.2**	–	–
BP + RUN (Ours)	262K	RUN/2.5ms	86.3	89.8	80.8	**68.4**
CBP + RUN (Ours)	10K	RUN/2.5ms	85.7	91.0	80.5	67.3

counterparts with matrix normalization. We further compare with CBP [8] and LRBP [13]. CBP uses Tensor Sketch and Random Maclaurin to reduce the feature dimension, whereas LRBP adopts the low-rank strategy for a compact feature. Nevertheless, neither CBP nor LRBP adopts matrix normalization. Thus their classification accuracies are not as high as compact bilinear methods with matrix normalization such as MoNet [9], GP [32] and our RUN.

We then compare with Improved BCNN [20] and BCNN + Newton-Schulz (NS). To make a fair comparison with BCNN + NS, we directly use the NS layer released by the authors of iSQRT [18], and keep all other settings identical. As shown in Table 7, they achieve high accuracies but generate high-dimension features and take high cost in matrix normalization. We further compare with iSQRT [18]. In iSQRT [18], the authors conduct an additional feature dimension reduction operation on the local feature, and reduces the feature dimension from 512 to 256. The dimension reduction on local features decreases the normalization time 833ms from 107ms. Using the 256-dimension local features, the normalization cost of our RUN can also be reduced, but that is not the focus of this paper. Meanwhile, as shown in Table 7, on CUB dataset, iSQRT achieves a higher accuracy than ours and other compared methods. The better performance might be contributed to that iSQRT is pre-trained on ImageNet dataset. In contrast, ours and other methods adopt a vanilla VGG16 pretrained model and adds the bilinear pooling operation only in the fine-tuning stage. More promising results might be achieved if we pre-train VGG16 with our RUN layer on ImageNet dataset, but that is not the focus of this paper, either.

After that, we compare with MoNet-2 and MoNet [9]. MoNet-2 achieves high accuracies and generate compact feature, but the time cost in matrix normalization is extremely high. MoNet improves MoNet-2 by fusing the first-order information, achieving higher accuracies, but is also slow in matrix normalization. By fusing higher-order features, KP [5] achieves an excellent performance on CUB dataset even without normalization. A better performance of the proposed RUN might be achieved by fusing the first-order and higher-order features likewise MoNet and KP, but it is not the focus of this paper. We also compare with HBP [34] and GP [32]. HBP ensembles three types of features and achieves a better performance than ours on CUB dataset. We can also achieve a better performance by ensembling several features, but that is not the focus on this paper, either. GP conducts SVD along the feature map and selects the first a few singular vectors as the image representation. It achieves a comparable performance with the proposed RUN on CUB and Aircraft datasets, using only 4K-dimension features. But it takes twice SVD, which is inefficient on the GPU.

We further compare with two recent work DeepKSPD [6] and DBTNet-50 [37]. By exploiting the Gaussian RBF kernel, DeepKSPD achieves an excellent performance. Our RUN is orthogonal to the kernel used in DeepKSPD and the RUN can also be used for normalization the bilinear matrix from DeepKSPD. DBTNet-50 [37] exploits the bilinear transformation in early layers, which relies on pretraining on a large-scale dataset. In contrast, we only exploit bilinear pooling in the late stage of the network, and does not reply on the pre-training.

5 Conclusion

We propose a simple and fast rank-1 update normalization (RUN) to improve the effectiveness of the bilinear matrix. Since it only takes several iterations of matrix-vector multiplications, the proposed RUN not only takes cheap computation and memory complexity in theory but also is well supported in the GPU platform in practice. More importantly, the proposed RUN supports normalization on compact bilinear features, which cannot be achieved by existing fast normalization methods based on the NS iteration. In addition, our RUN is much simpler than NS iteration and considerably easier for implementation. Meanwhile, RUN is differentiable and hence we plug it into a convolutional neural network, achieving an end-to-end training. Our systematic experiments on four datasets show that, combined with original bilinear pooling, we achieve comparable or even better accuracies with a substantial speedup over NS iteration on the GPU. When using compact bilinear pooling, we achieve comparable accuracies with a significant speedup over the SVD-based method on the GPU.

References

1. Arsigny, V., Fillard, P., Pennec, X., Ayache, N.: Geometric means in a novel vector space structure on symmetric positive-definite matrices. Siam Journal on Matrix

Analysis and Applications **29**(1), 328–347 (2006)
2. Burden, R.L., Faires, J.D.: Numerical analysis: 4th ed (1988)
3. Cherian, A., Koniusz, P., Gould, S.: Higher-order pooling of cnn features via kernel linearization for action recognition. In: Applications of Computer Vision (2017)
4. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., Vedaldi, A.: Describing textures in the wild. In: CVPR (2014)
5. Cui, Y., Zhou, F., Wang, J., Liu, X., Lin, Y., Belongie, S.: Kernel pooling for convolutional neural networks. In: CVPR. IEEE (2017)
6. Engin, M., Wang, L., Zhou, L., Liu, X.: Deepkspd: Learning kernel-matrix-based spd representation for fine-grained image recognition. In: ECCV. pp. 612–627. Springer (2018)
7. Fukui, A., Park, D.H., Yang, D., Rohrbach, A., Darrell, T., Rohrbach, M.: Multimodal compact bilinear pooling for visual question answering and visual grounding. In: EMNLP (2016)
8. Gao, Y., Beijbom, O., Zhang, N., Darrell, T.: Compact bilinear pooling. In: CVPR. IEEE (2016)
9. Gou, M., Xiong, F., Camps, O., Sznaier, M.: Monet: Moments embedding network. In: CVPR. IEEE (2018)
10. Higham, N.J.: Functions of matrices: theory and computation, vol. 104. Siam (2008)
11. Ionescu, C., Vantzos, O., Sminchisescu, C.: Matrix backpropagation for deep networks with structured layers. In: ICCV. IEEE (2015)
12. Kar, P., Karnick, H.: Random feature maps for dot product kernels. In: AISTATS (2012)
13. Kong, S., Fowlkes, C.: Low-rank bilinear pooling for fine-grained classification. In: CVPR. pp. 365–374. IEEE (2017)
14. Koniusz, P., Cherian, A., Porikli, F.: Tensor representations via kernel linearization for action recognition from 3d skeletons. In: ECCV. Springer (2016)
15. Koniusz, P., Yan, F., Gosselin, P.H., Mikolajczyk, K.: Higher-order occurrence pooling for bags-of-words: Visual concept detection. T-PAMI **39**(2), 313–326 (2017)
16. Koniusz, P., Zhang, H., Porikli, F.: A deeper look at power normalizations. In: CVPR. IEEE (2018)
17. Lei, W., Zhang, J., Zhou, L., Chang, T., Li, W.: Beyond covariance: Feature representation with nonlinear kernel matrices. In: ICCV. IEEE (2015)
18. Li, P., Xie, J., Wang, Q., Gao, Z.: Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In: CVPR. IEEE (2018)
19. Li, P., Xie, J., Wang, Q., Zuo, W.: Is second-order information helpful for large-scale visual recognition? In: ICCV. IEEE (2017)
20. Lin, T.Y., Maji, S.: Improved bilinear pooling with cnns. In: BMVC (2017)
21. Lin, T.Y., Maji, S., Koniusz, P.: Second-order democratic aggregation. In: ECCV. Springer (2018)
22. Lin, T.Y., Roychowdhury, A., Maji, S.: Bilinear cnn models for fine-grained visual recognition. In: ICCV. IEEE (2015)
23. Maji, S., Kannala, J., Rahtu, E., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. Tech. rep. (2013)
24. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: ICLR (2018)
25. Pham, N., Pagh, R.: Fast and scalable polynomial kernels via explicit feature maps. In: SIGKDD. pp. 239–247. ACM (2013)

26. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: CVPR. IEEE (2009)
27. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
28. Tu, Z., Xie, W., Qin, Q., Poppe, R., Veltkamp, R.C., Li, B., Yuan, J.: Multi-stream cnn: Learning representations based on human-related regions for action recognition. PR **79**, 32–43 (2018)
29. Wang, Q., Li, P., Zhang, L.: G2denet: Global gaussian distribution embedding network and its application to visual recognition. In: CVPR. IEEE (2017)
30. Wang, Q., Li, P., Zuo, W., Lei, Z.: Raid-g: Robust estimation of approximate infinite dimensional gaussian with application to material recognition. In: CVPR. IEEE (2016)
31. Wang, Y., Long, M., Wang, J., Yu, P.S.: Spatiotemporal pyramid network for video action recognition. In: CVPR. IEEE (2017)
32. Wei, X., Zhang, Y., Gong, Y., Zhang, J., Zheng, N.: Grassmann pooling as compact homogeneous bilinear pooling for fine-grained visual classification. In: ECCV. Springer (2018)
33. Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., Perona, P.: Caltech-ucsd birds 200 (2010)
34. Yu, C., Zhao, X., Zheng, Q., Zhang, P., You, X.: Hierarchical bilinear pooling for fine-grained visual recognition. In: ECCV. Springer (2018)
35. Yu, T., Meng, J., Yuan, J.: Multi-view harmonized bilinear network for 3d object recognition. In: CVPR. IEEE (2018)
36. Yu, Z., Yu, J., Fan, J., Tao, D.: Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In: ICCV. IEEE (2017)
37. Zheng, H., Fu, J., Zha, Z.J., Luo, J.: Learning deep bilinear transformation for fine-grained image representation. In: Advances in Neural Information Processing Systems. pp. 4277–4286. Curran Associates, Inc. (2019)
38. Zhou, L., Lei, W., Zhang, J., Shi, Y., Yang, G.: Revisiting metric learning for spd matrix based visual representation. In: CVPR. IEEE (2017)