Fast and Compact Bilinear Pooling by Shifted Random Maclaurin

Tan Yu, Xiaoyun Li, Ping Li
Cognitive Computing Lab
Baidu Research
10900 NE 8th St. Bellevue, WA 98004, USA
{tanyuuynat,lixiaoyun996, pingli98}@gmail.com

Abstract

Bilinear pooling has achieved an excellent performance in many computer vision tasks. However, the high-dimension features from bilinear pooling can sometimes be inefficient and prone to over-fitting. Random Maclaurin (RM) is a widely used GPU-friendly approximation method to reduce the dimensionality of bilinear features. However, to achieve good performance, huge projection matrices are usually required in practice, making it extremely costly in computation and memory. In this paper, we propose a Shifted Random Maclaurin (SRM) strategy for fast and compact bilinear pooling. With merely negligible extra computational cost, the proposed SRM provides an estimator with a provably smaller variance than RM, which benefits accurate kernel approximation and thus the learning performance. Using a small projection matrix, the proposed SRM achieves a comparable estimation performance as RM based on a large projection matrix, and thus considerably boosts the efficiency. Furthermore, we upgrade the proposed SRM to SRM+ to further improve the efficiency and make the compact bilinear pooling compatible with fast matrix normalization. Fast and Compact Bilinear Network (FCBN) built upon the proposed SRM+ is devised, achieving an end-to-end training. Systematic experiments conducted on four public datasets demonstrate the effectiveness and efficiency of the proposed FCBN.

Introduction

Bilinear pooling has achieved excellent performance in many computer vision tasks such as fine-grained classification (Lin, RoyChowdhury, and Maji 2015; Gao et al. 2016; Wang et al. 2016), generic image recognition (Li et al. 2018), semantic segmentation (Ionescu, Vantzos, and Sminchisescu 2015) and video recognition (Wang, Li, and Zhang 2017; Koniusz, Cherian, and Porikli 2016; Cherian, Koniusz, and Gould 2017). Given N local features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{d\times N}$ (e.g. local features from a convolution layer of a CNN), bilinear pooling obtains the matrix

$$\mathbf{B}_x = \mathbf{X}\mathbf{X}^\top,$$

where $\mathbf{B}_x \in \mathbb{R}^{d\times d}$. The matrix $\mathbf{B}_x$ is reshaped into a vector $\mathrm{vec}(\mathbf{B}_x) \in \mathbb{R}^{d^2}$ as the holistic image/video representation.

Since the bilinear feature $\mathrm{vec}(\mathbf{B}_x)$ is typically high-dimensional, it brings high memory and computation cost in many practical applications. Another consequence is that, it usually requires a large number of training samples to suppress over-fitting. To tackle the existing limitations of high-dimensional bilinear features, compact bilinear pooling (CBP) (Gao et al. 2016) seeks to reduce the feature dimensionality. It discovers the connection between bilinear pooling and polynomial kernel, and exploits methods used for polynomial kernel approximation (Kar and Karnick 2012; Pham and Pagh 2013) to obtain low-dimension bilinear features. More specifically, it re-formulates bilinear pooling as

$$\mathbf{B}_x = \sum_{i=1}^{N} \mathrm{h}(\mathbf{x}_i), \tag{1}$$

where $\mathrm{h}(\mathbf{x}) = \mathbf{x}_i\mathbf{x}_i^\top \in \mathbf{R}^{d\times d}$ is the explicit feature map of the second-order polynomial kernel. CBP finds an approximation function $\phi_{\mathbf{w}} : \mathbb{R}^{d^2} \mapsto \mathbb{R}^D$ with $D \ll d^2$, such that

$$\mathrm{E}\langle\phi_{\mathbf{w}}(\mathbf{x}), \phi_{\mathbf{w}}(\mathbf{y})\rangle = \langle\mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y}))\rangle, \tag{2}$$

where $\mathbf{w}$ contains random variables and $\mathrm{E}(\cdot)$ denotes the expectation with respect to $\mathbf{w}$. Here, $\mathrm{vec}(\cdot)$ denotes the operation which unrolls a matrix into a vector by concatenating the columns. Given two images, CBP obtains their compact bilinear features $\hat{\mathbf{B}}_x = \sum_{i=1}^{N} \phi_{\mathbf{w}}(\mathbf{x}_i)$ and $\hat{\mathbf{B}}_y = \sum_{i=1}^{N} \phi_{\mathbf{w}}(\mathbf{y}_i)$. Based on Eq. (2), it is straightforward to get

$$\mathrm{E}(\langle\hat{\mathbf{B}}_x, \hat{\mathbf{B}}_y\rangle) = \sum_{i=1}^{N}\sum_{j=1}^{N} \mathrm{E}(\langle\phi_{\mathbf{w}}(\mathbf{x}_i), \phi_{\mathbf{w}}(\mathbf{y}_j)\rangle) = \langle\mathbf{B}_x, \mathbf{B}_y\rangle.$$

That is, the inner product of CBP features $\langle\hat{\mathbf{B}}_x, \hat{\mathbf{B}}_y\rangle$ is an unbiased estimator of that of bilinear features $\langle\mathbf{B}_x, \mathbf{B}_y\rangle$, which is the pivotal quantity used in subsequent learning models.

Tensor Sketch (TS) (Pham and Pagh 2013) and Random Maclaurin (RM) (Kar and Karnick 2012) are two popular methods that admit Eq. (2). TS leverages sketching functions and Fast Fourier Transform to improve the efficiency, while RM relies on random projections. Importantly, although TS has a lower computation cost than RM in theory, it requires sparse matrix-vector product, which is not friendly for the GPU. In fact, the GPU time cost of TS is larger than RM. In contrast, RM relies on dense matrix-vector product and element-wise product, which are very suited for the GPU.

Recall that the inner product between compact features $\langle\hat{\mathbf{B}}_x, \hat{\mathbf{B}}_y\rangle$ is an unbiased estimator of $\langle\mathbf{B}_x, \mathbf{B}_y\rangle$. For the unbiased estimation, the variance controls the estimation accuracy. In general, we need a huge number of projections to

achieve a small variance, and thus a good learning performance. Therefore, large-scale projection matrices are often required for RM, making the projection expensive in both memory and computation. This unsatisfactory fact motivates us to propose a new strategy called Shifted Random Maclaurin (SRM) to improve the efficiency. The idea is to use small projection matrices with shifting operation. We theoretically prove that, using the same projection matrix, the variance achieved by SRM is smaller than that of RM. To put it in another way, to achieve the same variance, the size of the projection matrix used in the proposed SRM is smaller than that of RM, thus our SRM is more efficient than RM.

Recently, many works (Lin and Maji 2017; Li et al. 2018; Yu, Cai, and Li 2020) discover that matrix square-root normalization on the bilinear matrix $\mathbf{B}$ is vital for performance. However, the aforementioned RM and SRM break the matrix structure of the bilinear feature, and make the matrix normalization unfeasible. To overcome the limitation, MoNet (Gou et al. 2018) conducts singular value decomposition (SVD) on local features $\mathbf{X}$ rather than the bilinear matrix $\mathbf{B}$. Nevertheless, SVD is not easily parallelizable, and thus it is slow in the GPU. To boost the efficiency in the matrix normalization, we upgrade the proposed SRM to SRM+, which is compatible with the fast matrix normalization based on Newton-Schulz iteration (Li et al. 2018; Lin and Maji 2017). Besides, SRM+ is faster than SRM. Table 1 compares the proposed method with existing bilinear pooling methods. In summary, our contributions are three-fold:

- We propose a Shifted Random Maclaurin (SRM) method. Compared with existing RM method, our SRM method is more efficient for generating compact bilinear features.
- We upgrade the proposed SRM to SRM+. SRM+ is more efficient in generating compact features. More importantly, it is compatible with fast matrix normalization, making it more efficient in feature normalization as well.
- We integrate our SRM+ as a layer and propose a Fast and Compact Bilinear network (FCBN). Extensive experiments conducted on four public datasets demonstrate the efficiency and effectiveness of the proposed FCBN.

	Compact	Fast in Pooling	Support Norm	Fast in Norm
BCNN		✓		
I-BCNN		✓	✓	
iSQRT		✓	✓	✓
CBP	✓			
MoNet	✓		✓	
GN	✓		✓	
FCBN (ours)	✓	✓	✓	✓

Table 1: Pooling method overview. BCNN (Lin, RoyChowdhury, and Maji 2015), I-BCNN (Lin and Maji 2017), and iSQRT (Li et al. 2018) cannot support RM used in CBP (Gao et al. 2016). CBP generates compact bilinear features, but the pooling is slow and cannot support feature normalization. MoNet (Gou et al. 2018) and GP (Wei et al. 2018) generate compact and normalized bilinear features, but they are slow in pooling and feature normalization. In contrast, our FCBN generates compact and normalized bilinear features, and is fast in both pooling and feature normalization.

Related Work

Bilinear model was proposed by Tenenbaum and Freeman (2000) to disentangle style from content. B-CNN (Lin, RoyChowdhury, and Maji 2015) integrates bilinear pooling as a layer of a CNN which supports an end-to-end training. The following research on bilinear pooling proceeds along two tracks. The first track focuses on improving the discriminating capability of the bilinear feature for a higher recognition accuracy. G^2DeNet (Wang, Li, and Zhang 2017) combines the lower-order (first-order) pooling feature with second-order bilinear feature, achieving a better performance than the original bilinear feature. In another direction, KP (Cui et al. 2017) and Higher-order Occurrence Pooling (Koniusz et al. 2017) exploit higher-order pooling feature and also achieve a better performance. HBP (Yu et al. 2018) conducts the bilinear pooling on local features across different layers and encodes richer visual information. α-CML (Zhou et al. 2017) exploits metric learning for improving the effectiveness of the bilinear feature. Wang et al. (2015); Engin et al. (2018) extend the bilinear matrix to kernel matrix to model the nonlinearity in the local descriptor set. In parallel, some work exploit matrix normalization to improve the effectiveness of the bilinear feature. For instance, Deep O^2P (Ionescu, Vantzos, and Sminchisescu 2015) utilizes matrix-logarithm normalization and some other works (Lin and Maji 2017; Li et al. 2018; Koniusz, Zhang, and Porikli 2018; Yu, Meng, and Yuan 2018; Yu et al. 2021) to conduct matrix power normalization, achieving a higher recognition accuracy than that without normalization. Traditional matrix normalization is based on SVD, which is not friendly to GPU. To improve the efficiency, improved BCNN (Lin and Maji 2017) approximates the matrix square-root normalization through Newton-Schulz (NS) iteration (Higham 2008) in the forward propagation. iSQRT (Li et al. 2018) supports NS in the backward propagation besides forward propagation, making it efficient in the training phase as well. Recently, RUN (Yu, Cai, and Li 2020) further speeds up the matrix normalization through power method.

Since bilinear features are high-dimensional, directly utilizing them is inefficient and prone to over-fitting. Therefore, the second research track of bilinear neural network focuses on reducing the dimension of bilinear features. CBP (Gao et al. 2016) discovers the connection between the bilinear pooling and polynomial kernel. It utilizes existing kernel approximation approaches, Random Maclaurin and Tensor Sketch, and achieves low-dimensional approximation of original high-dimensional bilinear features. The low-dimensional CBP features achieve a comparable recognition accuracy with that using original bilinear features. Similarly, KP (Cui et al. 2017) applies tensor sketch to reduce dimensionality of the high-order pooling features. Nevertheless, the low-dimensional feature obtained from random Maclaurin and tensor sketch has broken the original matrix structure, making the following matrix normalization infeasible.

Algorithm 1 Random Maclaurin (RM)

Input: Local features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{d\times N}$, D
Output: The compact bilinear feature $\Phi_{\mathrm{RM}}(\mathbf{X}) \in \mathbb{R}^D$

1: Generate random matrices $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{D\times d}$, where each item is either $+1$ or -1 with equal probability.
2: **for** $i \in [1, N]$ **do**
3: $\quad \phi(\mathbf{x}_i) = \frac{1}{\sqrt{D}}(\mathbf{W}_1\mathbf{x}_i) \odot (\mathbf{W}_2\mathbf{x}_i)$, where $\odot$ denotes element-wise multiplication.
4: $\Phi_{\mathrm{RM}}(\mathbf{X}) = \sum_{i=1}^N \phi(\mathbf{x}_i)$
5: **return** $\Phi_{\mathrm{RM}}(\mathbf{X})$

To overcome this obstacle, MoNet (Gou et al. 2018) conducts SVD directly on local features before bilinear pooling. But as we mentioned, SVD is not well supported in the GPU. Parallel to reducing the dimension of bilinear features, LRBP (Kong and Fowlkes 2017) and FBN (Li et al. 2017) impose low-rank constraints on the classifier parameters, which achieve the same goal of improving the efficiency in training the classifier and suppressing over-fitting.

Tensor Sketch and Random Maclaurin

We briefly review Tensor Sketch (TS) and Random Maclaurin (RM) in Algorithm 1 and Algorithm 2, respectively. For each local feature $\mathbf{x}_i$, RM only needs to compute twice matrix-vector multiplications and a vector-vector element-wise multiplication, resulting in a computational complexity of $\mathcal{O}(dD)$, where d is the dimension of the original local feature $\mathbf{x}$ and D is the dimension of the projected feature $\phi_{\mathrm{RM}}(\mathbf{x})$. Since RM only involves matrix-vector and element-wise multiplications, it is very suited for the GPU.

In theory, since each column of $\mathbf{Q}_1$/$\mathbf{Q}_2$ in step 3 of TS has a single non-zero element, the computation complexity of step 5 of TS is $\mathcal{O}(D)$. Nevertheless, the multiplication between sparse-matrix and vector is not well supported in the GPU. A common practice converts the sparse matrix to a dense matrix. Thus, the computation complexity of the step 5 increases to $\mathcal{O}(dD)$, which is in a comparable scale with that of RM. Additionally, in step 6 of TS, twice FFT and once inverse FFT are conducted, making it slower.

In the sequel, we introduce the properties of the RM. For simplicity, we first consider the case where $D = 1$ (one projection direction). Let us consider two random projection vectors $\mathbf{w}^1, \mathbf{w}^2 \in \mathbb{R}^d$. Each element of $\mathbf{w}^1$ and $\mathbf{w}^2$ is randomly drawn from $\{+1, -1\}$ with equal probability. We define a function $\phi(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w}^1\rangle\langle \mathbf{x}, \mathbf{w}^2\rangle$. Given two constant vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, it is straightforward to obtain that

$$\begin{aligned}\mathrm{E}(\phi(\mathbf{x})\phi(\mathbf{y})) &= \mathrm{E}(\langle \mathbf{x}, \mathbf{w}^1\rangle\langle \mathbf{x}, \mathbf{w}^2\rangle\langle \mathbf{y}, \mathbf{w}^1\rangle\langle \mathbf{y}, \mathbf{w}^2\rangle) \\ &= \mathrm{E}(\langle \mathbf{x}, \mathbf{w}^1\rangle\langle \mathbf{y}, \mathbf{w}^1\rangle)\mathrm{E}(\langle \mathbf{x}, \mathbf{w}^2\rangle\langle \mathbf{y}, \mathbf{w}^2\rangle) \\ &= \langle \mathbf{x}, \mathbf{y}\rangle^2 = \langle \mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y}))\rangle,\end{aligned}$$

where $h(\mathbf{x}) = \mathbf{x}\mathbf{x}^\top$. Thus, $\phi(\mathbf{x})\phi(\mathbf{y})$ is an unbiased estimation of $\langle \mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y}))\rangle$, as required by Eq. (2).

Algorithm 2 Tensor Sketch (TS)

Input: Local features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{d\times N}$, D
Output: The compact bilinear feature $\Phi_{\mathrm{TS}}(\mathbf{X}) \in \mathbb{R}^D$

1: Generate random vectors $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{N}^d$. Each entry in $\mathbf{h}_1$ and $\mathbf{h}_2$ is uniformly sampled from $\{1, 2, \cdots, D\}$.
2: Generate random vectors $\mathbf{s}_1, \mathbf{s}_2 \in \{+1, -1\}^d$. Each entry in $\mathbf{s}_1$ and $\mathbf{s}_2$ is uniformly sampled from $\{+1, -1\}$.
3: Generate vectors $\mathbf{Q}_1 = [\mathbf{q}_1^1, \cdots, \mathbf{q}_1^D]$ and $\mathbf{Q}_2 = [\mathbf{q}_2^1, \cdots, \mathbf{q}_2^D]$, where $\mathbf{q}_1^j(t) = \mathbf{s}_1(t)$ if $\mathbf{h}_1(t) = j$ otherwise 0, and $\mathbf{q}_2^j(t) = \mathbf{s}_2(t)$ if $\mathbf{h}_2(t) = j$ otherwise 0.
4: **for** $i \in [1, N]$ **do**
5: $\quad \psi_1(\mathbf{x}_i) = \mathbf{Q}_1^\top\mathbf{x}_i$, $\psi_2(\mathbf{x}_i) = \mathbf{Q}_2^\top\mathbf{x}_i$
6: $\quad \phi(\mathbf{x}_i) = \mathrm{FFT}^{-1}(\mathrm{FFT}(\psi_1(\mathbf{x}_i) \odot \mathrm{FFT}(\psi_2(\mathbf{x}_i)))$
7: $\Phi_{\mathrm{TS}}(\mathbf{X}) = \sum_{i=1}^N \phi(\mathbf{x}_i)$
8: **return** $\Phi_{\mathrm{TS}}(\mathbf{X})$

We denote the variance of $\phi(\mathbf{x})\phi(\mathbf{y})$ by σ^2 and define

$$\phi_{\mathrm{RM}}(\mathbf{x}) = \frac{1}{\sqrt{D}}\mathbf{W}_1\mathbf{x} \odot \mathbf{W}_2\mathbf{x} = \frac{1}{\sqrt{D}}[\phi_1(\mathbf{x}), \cdots, \phi_D(\mathbf{x})],$$

where $\mathbf{W}_1 = [\mathbf{w}_1^1, \cdots, \mathbf{w}_1^D]$ and $\mathbf{W}_2 = [\mathbf{w}_2^1, \cdots, \mathbf{w}_2^D]$, and $\phi_i(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w}_1^i\rangle\langle \mathbf{x}, \mathbf{w}_2^i\rangle$. It is straightforward to obtain

$$\begin{aligned}\mathrm{E}(\langle \phi_{\mathrm{RM}}(\mathbf{x}), \phi_{\mathrm{RM}}(\mathbf{y})\rangle) &= \frac{1}{D}\sum_{i=1}^D \mathrm{E}(\langle \phi_i(\mathbf{x}), \phi_i(\mathbf{y})\rangle) \\ &= \langle \mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y}))\rangle.\end{aligned}$$

Given $i \neq j$, the projection vectors $\{\mathbf{w}_1^i, \mathbf{w}_2^i\}$ and $\{\mathbf{w}_1^j, \mathbf{w}_2^j\}$ are independent. Therefore, when $i \neq j$, $\langle \phi_i(\mathbf{x}), \phi_i(\mathbf{y})\rangle$ and $\langle \phi_j(\mathbf{x}), \phi_j(\mathbf{y})\rangle$ are independent, which leads to

$$\begin{aligned}\mathrm{Var}(\langle \phi_{\mathrm{RM}}(\mathbf{x}), \phi_{\mathrm{RM}}(\mathbf{y})\rangle) &= \frac{1}{D^2}\sum_{i=1}^D \mathrm{Var}(\langle \phi_i(\mathbf{x}), \phi_i(\mathbf{y})\rangle) \\ &\triangleq \sigma^2/D.\end{aligned}$$

To achieve a small variance, D should be large. As the complexity of RM is $\mathcal{O}(dD)$, a large D leads to a high computation cost. In this work, we seek to improve the efficiency.

Shifted Random Maclaurin (SRM)

Our goal is to design a strategy that reduces the estimation variance of RM, with extra cost as small as possible. We decompose the RM into two phases: 1) two matrix-vector multiplications $\mathbf{v}_1 = \mathbf{W}_1\mathbf{x}$ and $\mathbf{v}_2 = \mathbf{W}_2\mathbf{x}$, with a complexity of $\mathcal{O}(dD)$; 2) a vector-vector element-wise multiplication $\mathbf{v}_1 \odot \mathbf{v}_2$, taking only $\mathcal{O}(D)$ complexity, which is basically negligible. In light of the complexity gap between these two phases, we propose Shifted Random Maclaurin (SRM) method, which takes the same phase 1 as RM but conducts the second phase for K times with shifting. Algorithm 3 describes the proposed SRM. The complexity of our SRM is $\mathcal{O}((d + K)D)$. Since $K \ll d$, the computation complexity of SRM is comparable with RM with $\mathcal{O}(dD)$ complexity. That is, by shifting $\mathbf{v}_2$ for $K - 1$ times with almost no extra cost, the proposed SRM is able to get $(K - 1)D$ more

Algorithm 3 Shifted Random Maclaurin (SRM)

Input: Local features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{d\times N}$, D, K
Output: $\Phi_{\mathrm{SRM}}(\mathbf{X}) \in \mathbb{R}^D$

Generate random matrices $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{D\times d}$, where each item is either $+1$ or -1 with equal probability.
for $i \in [1, N]$ **do**
  $\mathbf{v}_1 \leftarrow \mathbf{W}_1\mathbf{x}_i$, $\mathbf{v}_2 \leftarrow \mathbf{W}_2\mathbf{x}_i$
  $\phi^{(0)}_{\mathrm{SRM}}(\mathbf{x}_i) \leftarrow \mathbf{v}_1 \odot \mathbf{v}_2$
  for $k \in [1, K-1]$ **do**
    Get $\mathbf{v}_2^{(k)}$ by circularly shifting $\mathbf{v}_2$ by k entries
    $\phi^{(k)}_{\mathrm{SRM}}(\mathbf{x}) \leftarrow \mathbf{v}_1 \odot \mathbf{v}_2^{(k)}$
  $\phi_{\mathrm{SRM}}(\mathbf{x}_i) \leftarrow [\phi^{(0)}_{\mathrm{SRM}}(\mathbf{x}_i), \cdots, \phi^{(K-1)}_{\mathrm{SRM}}(\mathbf{x}_i)]/\sqrt{KD}$
$\Phi_{\mathrm{SRM}}(\mathbf{X}) = \sum_{i=1}^N \phi_{\mathrm{SRM}}(\mathbf{x}_i)$
return $\Phi_{\mathrm{SRM}}(\mathbf{X})$

samples and thus encodes richer information than RM. It is straightforward to see that SRM also provides an unbiased estimator of the second-order polynomial kernel $\mathrm{h}(\cdot)$, that is

$$\begin{aligned}\mathrm{E}[\langle\phi_{\mathrm{SRM}}(\mathbf{x}), \phi_{\mathrm{SRM}}(\mathbf{y})\rangle] &= \mathrm{E}[\langle\phi_{\mathrm{RM}}(\mathbf{x}), \phi_{\mathrm{RM}}(\mathbf{y})\rangle] \\ &= \langle\mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y}))\rangle.\end{aligned}$$

The merit of SRM is its smaller estimation variance compared with RM, which benefits accurate kernel distance recovery among the learned representations. We summarize the variance reduction of SRM in the following theorem.

Theorem 1 *Let ϕ_{RM} and ϕ_{SRM} be defined in Algorithm 1 and Algorithm 3, respectively. We have*

$$\mathrm{Var}[\langle\phi_{\mathrm{SRM}}(\mathbf{x}), \phi_{\mathrm{SRM}}(\mathbf{y})\rangle] = \epsilon\mathrm{Var}[\langle\phi_{\mathrm{RM}}(\mathbf{x}), \phi_{\mathrm{RM}}(\mathbf{y})\rangle],$$

where

$$\epsilon = \frac{1}{K} + \frac{K-1}{K}\cdot\frac{2}{t+2} \le 1,$$

$$t = \frac{\|\mathbf{x}\|^2\|\mathbf{y}\|^2 - \langle\mathbf{x}\odot\mathbf{x}, \mathbf{y}\odot\mathbf{y}\rangle}{\langle\mathbf{x},\mathbf{y}\rangle^2} \ge 0.$$

Since $\epsilon \le 1$, the proposed SRM always improves the approximation accuracy over RM by reducing the variance. To gain a rough knowledge about the magnitude of t, we notice that the local features obtained from a ReLU layer are non-negative and most values are around 0. If we assume that $\mathbf{x}$, $\mathbf{y}$ come from $i.i.d.$ half-normal distribution, it is not hard to show that $\frac{2}{t+2} \approx \frac{2}{2+\pi^2/4} \approx 0.448$. When K equals to 4 or 8, the variance can be reduced by $40\% \sim 50\%$. Again, we would like to re-emphasize that we only need to pay very little for this improvement in approximation. The complexity is $\mathcal{O}((d+K)D)$ for our SRM, while $\mathcal{O}(dD)$ for the vanilla RM. Since we set $K \ll d$, the extra cost becomes negligible.

Remark 1 *Another useful comparison is when the length of bilinear feature is fixed as D. In RM, the random projection matrices $\mathbf{W}_1^{\mathrm{RM}}, \mathbf{W}_1^{\mathrm{RM}} \in \mathbb{R}^{D\times d}$, while in SRM, the random projection matrices $\mathbf{W}_1^{\mathrm{SRM}}, \mathbf{W}_2^{\mathrm{SRM}} \in \mathbb{R}^{(D/K)\times d}$, saving the memory and computation cost by a factor of K in the random projection. That is, the SRM achieves a K* ***times speed-up ratio*** *in random projection over RM when the dimension of the compact bilinear feature is identical.*

From SRM to SRM+

Both RM and the proposed SRM are conducted on local features $\mathbf{X}$ before bilinear pooling. Since the matrix structure has been broken after RM or SRM (by decomposition $\mathbf{B}_x = \sum_{i=1}^N \mathrm{h}(\mathbf{x}_i)$), the normalization step is infeasible in this case. In this section, we upgrade the SRM to SRM+ to make it compatible with matrix square-root normalization. Our goal is to find a function $\Phi_{\mathbf{w}}(\cdot)$ with randomness $\mathbf{w}$, which takes a matrix as input and achieves

$$\mathrm{E}(\langle\Phi_{\mathbf{w}}(\mathbf{A}), \Phi_{\mathbf{w}}(\mathbf{B})\rangle) = \langle\mathrm{vec}(\mathbf{A}), \mathrm{vec}(\mathbf{B})\rangle. \tag{3}$$

Note here that $\mathbf{A}, \mathbf{B}$ are arbitrary matrices, not restricted to the form of $\mathbf{x}\mathbf{x}^\top$ as in Eq. (1). Before we introduce SRM+, we first introduce RM+ defined as

$$\Phi_{\mathrm{RM+}}(\mathbf{A}) = \frac{1}{\sqrt{d_2}}((\mathbf{W}_1\mathbf{A}) \odot \mathbf{W}_2)\mathbf{1}, \tag{4}$$

where $\mathbf{A} \in \mathbb{R}^{d_1\times d_2}$, and $\mathbf{W}_1, \mathbf{W}_2$ are random matrices with each entry uniformly sampled from $\{-1, 1\}$, and $\mathbf{1}$ is a column vector with each entry set as 1. RM+ satisfies the property in the following theorem:

Theorem 2 *Let Φ_{RM} be defined in Algorithm 1, $\Phi_{\mathrm{RM+}}$ be defined in Eq. (4), and $\mathbf{Y} \in \mathbb{R}^{d\times N}$ be any matrix, then*

$$\Phi_{\mathrm{RM+}}(\mathbf{Y}\mathbf{Y}^\top) = \Phi_{\mathrm{RM}}(\mathbf{Y}).$$

We can see that RM+ achieves the goal defined in Eq. (3). Meanwhile, due to the equivalence, RM+ achieves the same estimation variance as RM.

We first show the efficiency advantage of RM+ over RM in the feature normalization. By setting $\mathbf{Y} = \sqrt{\mathbf{X}}$, RM+ takes as input the normalized matrix $\sqrt{\mathbf{X}}\sqrt{\mathbf{X}}^\top = \sqrt{\mathbf{X}\mathbf{X}^\top}$ directly and obtain the compact normalized bilinear feature $\Phi_{\mathrm{RM+}}[\sqrt{\mathbf{X}\mathbf{X}^\top}]$. Using NS iteration (Li et al. 2018), $\sqrt{\mathbf{X}\mathbf{X}^\top}$ can be efficiently obtained. In contrast, RM is operated on the local features $\mathbf{X}$ and is incompatible with normalized bilinear features $\sqrt{\mathbf{X}\mathbf{X}^\top}$ obtained from NS iteration. MoNet (Gou et al. 2018) conducts SVD on $\mathbf{X}$ to obtain $\sqrt{\mathbf{X}}$ and then conducts $\Phi_{\mathrm{RM}}(\sqrt{\mathbf{X}})$ to obtain the compact and normalized bilinear feature. But SVD is difficult to be parallelized and is not well supported in the GPU platform.

Besides the faster normalization, we gain an additional reduction in computational cost from RM+. RM requires two random projections $\mathbf{W_1}\sqrt{\mathbf{X}}$ and $\mathbf{W_2}\sqrt{\mathbf{X}}$, while RM+ only needs one, namely $\mathbf{W}_1\mathbf{C}$ where $\mathbf{C} = \sqrt{\mathbf{X}\mathbf{X}^\top}$. Then an element-wise multiplication is applied to $\mathbf{W}_1\mathbf{C}$ and $\mathbf{W}_2$. Later in the experiment section, we will show that, the time cost of $\Phi_{\mathrm{RM+}}(\sqrt{\mathbf{X}\mathbf{X}^\top})$ is less than that of $\Phi_{\mathrm{RM}}(\sqrt{\mathbf{X}})$ not only in matrix normalization but also in other parts.

Analogously, for the shifted version, SRM+ can be extended from SRM in a similar manner by

$$\begin{aligned}\Phi_{\mathrm{SRM+}}(\mathbf{A}) = \frac{1}{\sqrt{DK}}[&((\mathbf{W}_1\mathbf{A}) \odot \mathbf{W}_2^{(0)})\mathbf{1}, \\ &\cdots, ((\mathbf{W}_1\mathbf{A}) \odot \mathbf{W}_2^{(K-1)})\mathbf{1}],\end{aligned} \tag{5}$$

where $\mathbf{W}_1\mathbf{A}$ is computed for only once and $\mathbf{W}_2^{(k)}$ is obtained by shifting the columns of $\mathbf{W}_2$ circularly by k. $\Phi_{\mathrm{SRM+}}$ satisfies the property in the following theorem:

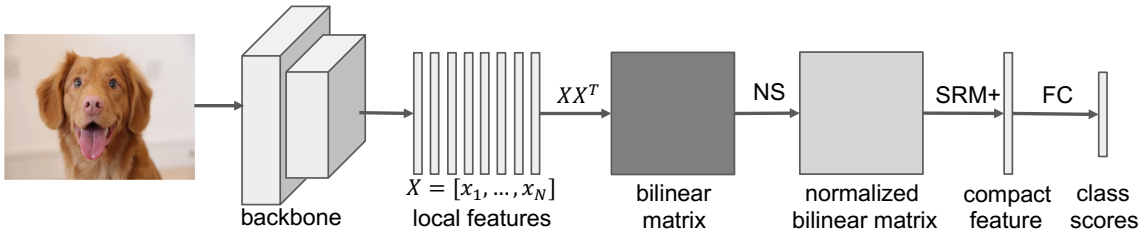


Figure 1: The architecture of the proposed FCBN. Given an image, the backbone network extracts local features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]$. The bilinear matrix is obtained by $\mathbf{C} = \mathbf{X}\mathbf{X}^\top$. Through Newton-Schulz (NS) iteration, the normalized bilinear feature $\bar{\mathbf{C}}$ is generated. The proposed SRM+ maps the normalized bilinear feature $\bar{\mathbf{C}}$ into a compact vector $\mathbf{b}$, followed by an element-wise square root normalization layer (sgnsqrt) and an ℓ_2 normalization. The normalized feature vector is further fed into a fully-connected (FC) layer and generates the class scores, which are used for computing cross-entropy loss.

Theorem 3 *Let Φ_{SRM} be defined in Algorithm 3, $\Phi_{\mathrm{SRM+}}$ be defined in Eq. (5) and $\mathbf{Y}$ be any matrix, we have*

$$\Phi_{\mathrm{SRM+}}(\mathbf{Y}\mathbf{Y}^\top) = \Phi_{\mathrm{SRM}}(\mathbf{Y}).$$

Again, $\Phi_{\mathrm{SRM+}}(\mathbf{Y}\mathbf{Y}^\top)$ has same statistical properties as $\Phi_{\mathrm{SRM}}(\mathbf{Y})$, with the same two-fold advantages: 1) SRM+ is compatible with NS iteration, which achieves a faster feature normalization; 2) SRM+ only needs once random projection and thus is more efficient than SRM which takes twice random projections in generating compact bilinear features.

Fast and Compact Bilinear Network

Figure 1 visualizes the architecture of the proposed Fast and Compact Bilinear Network (FCBN). The image goes through a backbone CNN to obtain a set of local features $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ from the last convolutional layer. Then the bilinear pooling obtains the bilinear matrix:

$$\mathbf{C} = \mathbf{X}\mathbf{X}^\top.$$

After that, the Newton-Schulz (NS) iteration (Li et al. 2018) generates the normalized bilinear matrix. Specifically, NS method conducts T times following iterations:

$$\mathbf{Y}_t = \frac{1}{2}\mathbf{Y}_{t-1}(3\mathbf{I} - \mathbf{Z}_{t-1}\mathbf{Y}_{t-1}),$$
$$\mathbf{Z}_t = \frac{1}{2}(3\mathbf{I} - \mathbf{Z}_{t-1}\mathbf{Y}_{t-1})\mathbf{Z}_{t-1},$$

where $\mathbf{Y}_0$ is initialized by the bilinear matrix $\mathbf{C}$ and $\mathbf{Z}_0$ is initialized by the identity matrix $\mathbf{I}$. Since $\mathbf{Y}_t$ converges to $\mathbf{C}^{1/2}$, we set the normalized bilinear matrix $\bar{\mathbf{C}} = \mathbf{Y}_T$. Following Li et al. (2018), we conduct pre-normalization and post-normalization in NS iteration. The SRM+ layer takes input the normalized bilinear matrix $\bar{\mathbf{C}}$ and generates the compact and normalized bilinear feature $\bar{\mathbf{b}}$ by

$$\bar{\mathbf{b}} = \frac{1}{\sqrt{DK}}[\Phi^{(0)}_{\mathrm{SRM+}}(\bar{\mathbf{C}}), \cdots, \Phi^{(K-1)}_{\mathrm{SRM+}}(\bar{\mathbf{C}})], \qquad (6)$$

where $\Phi^{(k)}_{\mathrm{SRM+}}(\bar{\mathbf{C}}) = ((\mathbf{W}_1\bar{\mathbf{C}}) \odot \mathbf{W}_2^{(k)})\mathbf{1}$. After that, following Lin and Maji (2017), an element-wise square-root and an ℓ_2-normalization are conducted. Note that, all operations in Eq. (6) are differentiable, and thus it readily supports backward propagation. In practice, we can directly use the autograd tool in existing deep learning frameworks to back-propagate gradients based on the forward propagation.

Experiments

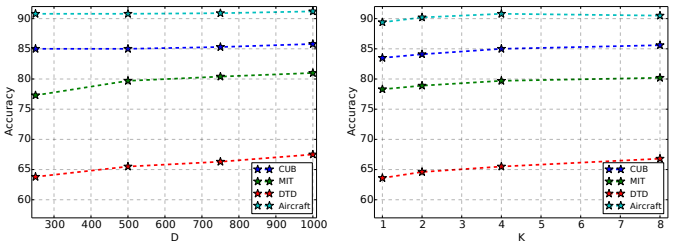
We evaluate the proposed methods based on VGG-16 network (Simonyan and Zisserman 2015) pre-trained on ImageNet dataset. On all experiments, we use 448×448 input image size and obtain a $28 \times 28 \times 512$ feature map. The training is conducted through two stages. In the first stage, all layers except the last fully-connected (FC) layer are fixed and only weights of the last FC layer are updated. The batch size is 32, the initial learning rate is set as 1. After 30 epochs, we drop learning rate by 10 every 10 epochs until the 60-th epoch. In the second stage, we update weights of all layers. The batch size is set as 32, the initial learning rate is set as 1×10^{-2}. After 30 epochs, the learning rate is decreased to 1×10^{-3} and the training process finishes in 40 epochs.

Experiments are conducted in fine-grained, scene and texture recognition. In fine-grained recognition task, we test on Caltech-UCSD birds (CUB) dataset (Welinder et al. 2010) and FGVC-Aircraft Benchmark (Aircraft) (). CUB contains $5,994$ training images and $5,794$ testing images from 200 categories. Aircraft contains $6,667$ training images and $3,333$ testing images from 100 categories. In scene recognition task, we test on MIT-scene dataset (MIT) (Quattoni and Torralba 2009), which contains $4,014$ training images and $1,339$ testing images from 67 classes. In texture recognition task, we test on Describable Texture (DTD) (Cimpoi et al. 2014), containing $1,880$ training images and 3760 testing images from 47 classes.

Ablation Study

Influence of D. We vary D among $\{250, 500, 750, 1000\}$ and fix K as 4. The accuracy is shown in Figure 2(a). As we can see, the accuracy of the proposed SRM+ improves as D increases. For instance, on CUB dataset, when D increases from 250 to 1000, the accuracy increases from 85.0 to 85.8.

Influence of K. We set $D = 500$ on all testing datasets. Notice that when $K = 1$, the proposed SRM+ degenerates to RM+. As shown in Figure 2(b), on CUB, MIT and DTD datasets, the accuracy improves as K increases. Meanwhile, the proposed SRM+ ($K > 1$) consistently outperforms RM+ ($K = 1$). For instance, on CUB dataset, when $K = 1$, it only achieves a 83.5 accuracy whereas it achieves a 85.6 accuracy when $K = 8$. On MIT dataset, RM+ ($K = 1$) only achieves



(a) The influence of D. (b) The influence of K.

Figure 2: The influence of D and K on the proposed SRM+.

Norm	CUB	MIT	DTD	Aircraft
No	83.7	78.1	66.1	87.7
Yes	**85.6**	**80.2**	**66.8**	**90.5**

Table 2: Influence of matrix normalization on the accuracy.

a 78.3 recognition accuracy whereas our SRM+ ($K = 8$) achieves a 80.2 accuracy. Note that, on Aircraft dataset, the best performance is achieved when $K = 4$. The worse performance when $K = 8$ might be caused by over-fitting.

Influence of normalization. As previously discussed, we implement the matrix normalization by Newton-Schulz iteration as in Li et al. (2018). As shown in Table 2, the normalization step improves the recognition accuracy.

Comparisons With Other Pooling Methods

We further compare with other pooling methods including fully-connected (FC), sum-pooling (Sum), max-pooling (Max), bilinear pooling (BP) and improved bilinear pooling (IBP). We set $K = 8$ and $D = 500$ on four datasets and thus the dimension of the proposed SRM+ feature is $D \times K = 4000$. As shown in Table 3, BP, IBP and ours consistently outperform FC, Sum and Max on four testing datasets, which validates the effectiveness of the bilinear pooling on fine-grained image recognition, texture recognition and scene recognition. Meanwhile, benefited from matrix normalization, IBP consistently outperforms original BP. Our method achieves a comparable accuracy with IBP with the compact feature of a much smaller dimension.

Method	Dimension	CUB	MIT	DTD	Aircraft
FC	4096	80.4	67.8	60.1	74.1
Sum	512	71.7	58.7	58.2	82.1
Max	512	69.6	50.4	51.1	78.9
BP	262K	84.0	77.5	67.5	86.9
IBP	262K	**85.6**	79.7	**67.7**	88.5
Ours	4000	**85.6**	**80.2**	66.8	**90.5**

Table 3: Comparisons with other pooling methods.

Dimension	2000	4000	8000	16000
RM	3.21G	6.42G	12.8G	25.7G
RM+	1.46G	2.51G	4.61G	8.81G
SRM+ ($K = 2$)	937M	1.46G	2.51G	4.61G
SRM+ ($K = 4$)	675M	939M	1.46G	2.52G
SRM+ ($K = 8$)	544M	677M	943M	1.47G

Table 4: FLOPs comparisons among RM, RM+ and SRM+.

	Dim	Time	Accuracy			
			CUB	MIT	DTD	Air
RM+	4K	29ms	85.2	80.5	67.0	90.5
SRM +	4K	7ms	85.6	80.2	66.8	90.5
RM+	8K	53ms	85.5	80.7	67.5	90.2
SRM+	8K	13ms	85.5	80.3	67.8	90.5

Table 5: Comparisons between RM+ and SRM+.

Efficiency Analysis

Advantage of RM+ over RM. Recall from Theorem 2 that $\Phi_{\mathrm{RM}+}(\mathbf{C}) = \Phi_{\mathrm{RM}}(\mathbf{Y})$ when $\mathbf{C} = \mathbf{Y}\mathbf{Y}^\top$. It is explicit that RM+ can achieve a faster matrix normalization than RM since it can be conducted on normalized matrix efficiently obtained from iSQRT. In the following, we confirm our statement that, besides faster matrix normalization, computing $\Phi_{\mathrm{RM}+}(\mathbf{C})$ is faster than $\Phi_{\mathrm{RM}}(\mathbf{Y})$ in other parts where $\mathbf{Y} = \sqrt{\mathbf{X}}$ and $\mathbf{C} = \sqrt{\mathbf{X}\mathbf{X}^\top}$. First we rewrite $\Phi_{\mathrm{RM}}(\mathbf{Y})$ as

$$\Phi_{\mathrm{RM}}(\mathbf{Y}) = ((\mathbf{W}_1\mathbf{Y}) \odot (\mathbf{W}_2\mathbf{Y}))\mathbf{1}. \tag{7}$$

In contrast, to obtain $\Phi_{\mathrm{RM}+}(\mathbf{C})$, we need compute:

$$\Phi_{\mathrm{RM}+}(\mathbf{C}) = ((\mathbf{W}_1\mathbf{C}) \odot \mathbf{W}_2)\mathbf{1}.$$

Meanwhile, before computing $\Phi_{\mathrm{RM}+}(\mathbf{C})$, we need compute an additional matrix-matrix multiplication $\mathbf{X}\mathbf{X}^\top$. Given $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{d\times N}$, $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{D\times d}$, the number of float operations (FLOPs) of computing $\Phi_{\mathrm{RM}}(\mathbf{Y})$ in Eq. (7) is

$$\mathrm{FLOPs(RM)} = 4DNd - D. \tag{8}$$

In contrast, FLOPs of computing $\Phi_{\mathrm{RM}+}(\mathbf{C})$ and $\mathbf{X}\mathbf{X}^\top$ is

$$\mathrm{FLOPs(RM+)} = d^2(2N + 2D - 1) + D(d - 1). \tag{9}$$

Note that, the FLOPs in Eq. (9) and Eq. (8) do not take the computational cost of matrix normalization into consideration since here we show efficiency advantage of RM+ over RM on other parts besides matrix normalization. In our case, $d = 512$, $N = 28^2$ and D is defined by the user. We compare the FLOPs of RM and RM+ when the dimension D varies among $\{2\mathrm{K}, 4\mathrm{K}, 8\mathrm{K}, 16\mathrm{K}\}$. As shown in Table 4, the FLOPs of RM+ is less than that of RM, validating the efficiency advantage of RM+ over RM in other parts besides matrix normalization.

Advantage of SRM+ over RM+. $\Phi_{\mathrm{SRM}+}(\mathbf{X}\mathbf{X}^\top)$ computes

$$\tilde{\mathbf{W}}_1 = \mathbf{W}_1\mathbf{C}, \{(\tilde{\mathbf{W}}_1 \odot \mathbf{W}_2^{(k)})\mathbf{1}\}_{k=0}^{K-1}. \tag{10}$$

When the feature dimension is D, in SRM+, $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{\frac{D}{K}\times d}$. That is, when the feature dimension is fixed, a

Method	Dimension	Pooling Time	Norm Time	Accuracy			
				CUB	MIT	DTD	Aircraft
BCNN (Lin, RoyChowdhury, and Maji 2015)	262K	3.3ms	–	84.0	77.5	67.5	84.0
I-BCNN (Lin and Maji 2017)	262K	3.3ms	6.3s	85.8	–	–	88.5
G^{2}DeNet (Wang, Li, and Zhang 2017)	263K	3.4ms	6.5s	87.1	–	–	89.0
iSQRT (Li et al. 2018)	32K	0.9ms	0.1s	**87.2**	–	–	90.0
CBP-TS (Gao et al. 2016)	8K	215.6ms	–	84.0	76.2	**67.8**	–
CBP-RM (Gao et al. 2016)	8K	123.3ms	–	83.9	73.9	66.7	–
LRBP (Kong and Fowlkes 2017)	10K	2.2ms	–	84.2	–	65.8	87.3
HBP (Yu et al. 2018)	24K	352.1ms	–	87.1	–	–	90.3
MoNet-2 (Gou et al. 2018)	10K	131.6ms	13.6s	85.7	–	–	88.1
GP (Wei et al. 2018)	4K	15.3s		85.8	–	–	89.8
FCBN (ours)	4K	7.4ms	0.8s	85.6	80.2	66.8	**90.5**
FCBN (ours)	8K	12.8ms	0.8s	85.5	**80.3**	**67.8**	**90.5**

Table 6: Compare with state-of-the-art methods.

larger K leads to smaller projection matrices $\mathbf{W}_1, \mathbf{W}_2$. The FLOPs used in computing 2 steps in Eq (10) and $\mathbf{X}\mathbf{X}^\top$ is

$$\text{FLOPs}(\text{SRM+}) = d^2(2N + \frac{2D}{K} - 1) + D(2d - 1 - \frac{d}{K}).$$

Note that, when $K = 1$, FLOPs(SRM+) = FLOPs(RM+). This is in accord with the fact that, RM+ is a special case of SRM+ when $K = 1$. We also show FLOPs of SRM+ when K varies among $\{2, 4, 8\}$ in Table 4. As shown in the table, FLOPs of SRM+ is considerably less than that of RM+.

We further compare RM+ and SRM+ on the factual time cost and the classification accuracies on benchmarks. We set $K = 8$ and change the feature dimension between 4000 and 8000. Consequently, the size of projection matrices used in RM+ is $512 \times 4000(8000)$. In contrast, due to K times shifting, the size of projection matrices used in SRM+ is only $512 \times 500(1000)$. As shown in Table 5, SRM+ achieves comparable accuracies with RM+ on four datasets. Meanwhile, the time cost of our SRM+ is considerably less than RM+. For example, when the dimension is set as 4000, our SRM+ only takes 7ms whereas the RM+ takes 29ms.

Comparisons With State-of-the-art Methods

We compare the pooling time, normalization time, and accuracies on four datasets with existing state-of-the-art methods in Table 6. We implement all methods in the same server.

We first compare with non-compact bilinear methods including original B-CNN, I-BCNN, G^{2}DeNet and iSQRT. As shown in Table 6, BCNN, I-BCNN, G^{2}DeNet and iSQRT achieve comparable or even higher accuracies than ours. The bilinear pooling used in BCNN, I-BCNN, G^{2}DeNet and iSQRT is faster than ours and other compact methods. But they generate high-dimension features. Note that, despite we use the same NS iteration as iSQRT, the normalization time of ours is 8 times as that of iSQRT. This is due to the fact that iSQRT reduces the dimension of local features from 512 to 256 though a convolution layer. Our FCBN will achieve the same normalization time with iSQRT when reducing the dimension of local convolutional features to 256.

Secondly, we compare with methods based on compact bilinear features. Due to lack of normalization, accuracies of CBP and LRBP are not as high as ours. Note that LRBP projects local features into low-dimension features and thus it is faster than ours in pooling. In fact, we can also further reduce the computation cost of our SRM+ by projecting the local feature into low-dimension features. We also compare with HBP which fuses features from three convolution layers. Benefited from the feature fusion, HBP achieves a better performance than ours on CUB. We can also improve the accuracy of our FCBN by fusing features of multiple layers, but that is not the focus of this paper. We further compare with MoNet-2. It achieves comparable accuracies with ours, but their pooling time and normalization time are larger than ours. To be specific, MoNet-2 takes 131.6ms in compact bilinear pooling through random Maclaurin and 13.6s for normalization based on SVD. In contrast, our FCBN takes only 12.8ms in the proposed SRM+ and 0.8s in normalization based on NS iteration. We then compare with Grassmann Pooling (GP). GP conducts the feature pooling and normalization simultaneously, therefore, we report the total time used in the feature pooling normalization of GP. As shown in Table 6, GP and ours achieve comparable accuracy. But GP is much slower than ours in pooling and normalization.

Conclusion

In this paper, we propose a Shifted Random Maclaurin (SRM) method to improve the efficiency of compact bilinear pooling. Using the same projection matrix, SRM achieves a better estimation performance than RM. Meanwhile, we further upgrade SRM to SRM+ to further boost the efficiency and make it compatible with fast matrix normalization. Based on the proposed SRM+, we build a Fast and Compact Bilinear Network (FCBN) for an effective and efficient image recognition. Systematic experiments conducted on four public benchmark datasets demonstrate the effectiveness and the efficiency of the proposed method.

For future work, it is also possible to explore other techniques developed for random projections such as "very sparse random projections" (Li 2007) or "quantized random projections" (Li, Mitzenmacher, and Shrivastava 2014; Li and Li 2019), to improve SRM from different directions.

References

Cherian, A.; Koniusz, P.; and Gould, S. 2017. Higher-Order Pooling of CNN Features via Kernel Linearization for Action Recognition. In *Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 130–138. Santa Rosa, CA.

Cimpoi, M.; Maji, S.; Kokkinos, I.; Mohamed, S.; and Vedaldi, A. 2014. Describing Textures in the Wild. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3606–3613. Columbus, OH.

Cui, Y.; Zhou, F.; Wang, J.; Liu, X.; Lin, Y.; and Belongie, S. J. 2017. Kernel Pooling for Convolutional Neural Networks. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3049–3058. Honolulu, HI.

Engin, M.; Wang, L.; Zhou, L.; and Liu, X. 2018. DeepKSPD: Learning Kernel-Matrix-Based SPD Representation For Fine-Grained Image Recognition. In *Proceedings of the 15th European Conference on Computer Vision (ECCV), Part II*, 629–645. Munich, Germany.

Gao, Y.; Beijbom, O.; Zhang, N.; and Darrell, T. 2016. Compact Bilinear Pooling. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 317–326. Las Vegas, NV.

Gou, M.; Xiong, F.; Camps, O. I.; and Sznaier, M. 2018. MoNet: Moments Embedding Network. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3175–3183. Salt Lake City, UT.

Higham, N. J. 2008. *Functions of matrices-theory and computation*. SIAM.

Ionescu, C.; Vantzos, O.; and Sminchisescu, C. 2015. Matrix Backpropagation for Deep Networks with Structured Layers. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 2965–2973. Santiago, Chile.

Kar, P.; and Karnick, H. 2012. Random Feature Maps for Dot Product Kernels. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 583–591. La Palma, Canary Islands, Spain.

Kong, S.; and Fowlkes, C. C. 2017. Low-Rank Bilinear Pooling for Fine-Grained Classification. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7025–7034. Honolulu, HI.

Koniusz, P.; Cherian, A.; and Porikli, F. 2016. Tensor Representations via Kernel Linearization for Action Recognition from 3D Skeletons. In *Proceedings of the 14th European Conference on Computer Vision (ECCV), Part IV*, 37–53. Amsterdam, The Netherlands.

Koniusz, P.; Yan, F.; Gosselin, P.; and Mikolajczyk, K. 2017. Higher-Order Occurrence Pooling for Bags-of-Words: Visual Concept Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 39(2): 313–326.

Koniusz, P.; Zhang, H.; and Porikli, F. 2018. A Deeper Look at Power Normalizations. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5774–5783. Salt Lake City, UT.

Li, P. 2007. Very sparse stable random projections for dimension reduction in $l_\alpha(0 < \alpha \leq 2)$ norm. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 440–449. San Jose, CA.

Li, P.; Mitzenmacher, M.; and Shrivastava, A. 2014. Coding for Random Projections. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, 676–684. Beijing, China.

Li, P.; Xie, J.; Wang, Q.; and Gao, Z. 2018. Towards Faster Training of Global Covariance Pooling Networks by Iterative Matrix Square Root Normalization. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 947–955. Salt Lake City, UT.

Li, X.; and Li, P. 2019. Random Projections with Asymmetric Quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 10857–10866. Vancouver, Canada.

Li, Y.; Wang, N.; Liu, J.; and Hou, X. 2017. Factorized Bilinear Models for Image Recognition. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2098–2106. Venice, Italy.

Lin, T.; and Maji, S. 2017. Improved Bilinear Pooling with CNNs. In *Proceedings of British Machine Vision Conference (BMVC)*. London, UK.

Lin, T.; RoyChowdhury, A.; and Maji, S. 2015. Bilinear CNN Models for Fine-Grained Visual Recognition. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 1449–1457. Santiago, Chile.

Pham, N.; and Pagh, R. 2013. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 239–247. Chicago, IL.

Quattoni, A.; and Torralba, A. 2009. Recognizing indoor scenes. In *Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 413–420. Miami, FL.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. San Diego, CA.

Tenenbaum, J. B.; and Freeman, W. T. 2000. Separating Style and Content with Bilinear Models. *Neural Comput.* 12(6): 1247–1283.

Wang, L.; Zhang, J.; Zhou, L.; Tang, C.; and Li, W. 2015. Beyond Covariance: Feature Representation with Nonlinear Kernel Matrices. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 4570–4578. Santiago, Chile.

Wang, Q.; Li, P.; and Zhang, L. 2017. G2DeNet: Global Gaussian Distribution Embedding Network and Its Application to Visual Recognition. In *Proceedings of the 2017*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6507–6516. Honolulu, HI.

Wang, Q.; Li, P.; Zuo, W.; and Zhang, L. 2016. RAID-G: Robust Estimation of Approximate Infinite Dimensional Gaussian with Application to Material Recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4433–4441. Las Vegas, NV.

Wei, X.; Zhang, Y.; Gong, Y.; Zhang, J.; and Zheng, N. 2018. Grassmann Pooling as Compact Homogeneous Bilinear Pooling for Fine-Grained Visual Classification. In *Proceedings of the 15th European Conference on Computer Vision (ECCV), Part III*, 365–380. Munich, Germany.

Welinder, P.; Branson, S.; Mita, T.; Wah, C.; Schroff, F.; Belongie, S.; and Perona, P. 2010. Caltech-UCSD birds 200 .

Yu, C.; Zhao, X.; Zheng, Q.; Zhang, P.; and You, X. 2018. Hierarchical Bilinear Pooling for Fine-Grained Visual Recognition. In *Proceedings of the 15th European Conference on Computer Vision (ECCV), Part XVI*, 595–610. Munich, Germany.

Yu, T.; Cai, Y.; and Li, P. 2020. Toward Faster and Simpler Matrix Normalization via Rank-1 Update. In *Proceedings of the 16th European Conference on Computer Vision (ECCV), Part XIX*, 203–219. Glasgow, UK.

Yu, T.; Meng, J.; Yang, M.; and Yuan, J. 2021. 3D object representation learning: A set-to-set matching perspective. *IEEE Transactions on Image Processing* 30: 2168–2179.

Yu, T.; Meng, J.; and Yuan, J. 2018. Multi-view harmonized bilinear network for 3d object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 186–194. Salt Lake City, UT.

Zhou, L.; Wang, L.; Zhang, J.; Shi, Y.; and Gao, Y. 2017. Revisiting Metric Learning for SPD Matrix Based Visual Representation. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7111–7119. Honolulu, HI.