Skolkovo Institute of Science and Technology

Numerical Linear Algebra

Project Report

Fast Matrix Normalization for bilinear pooling using Rank-1 Update

Team A*

June 6, 2021

Team members

- Andreea Dogaru
- Timotei Ardelean
- Alsu Vakhitova

Distribution of the work

Member	Tasks
Andreea Dogaru	Report: Introduction, Results. Code: CBP; MIT training and evaluation. Slides, presentation.
Timotei Ardelean	Report: Results. Code: BCNN, iBCNN, RUN; CUB200 training and evaluation. Slides, presentation.
Alsu Vakhitova	Report: Theoretical Framework, Conclusion. Code: DTD training and evaluation. Slides, presentation.

GitHub Link

`https://github.com/AndreeaDogaru/BCNN-RUN`

Contents

1 Introduction

Our project aims at reproducing the results presented in the article “Toward Faster and Simpler Matrix Normalization via Rank-1 Update” [11]. The reference solution achieves a substantial speedup without compromising accuracy for three tasks: fine-grained recognition, scene recognition, and texture recognition. In the next paragraphs, we are going to introduce the main ideas and present some related works relevant to our project.

1.1 Background

Fine-Grained Visual Recognition is the Computer Vision task of discriminating between visually highly similar objects, such as classifying the species of a bird, the model of an aircraft, or the type of a flower. Fine-Grained categorization is more challenging compared with the task of generic object recognition because the intra-category visual variations are small and can be overwhelmed by factors such as pose, view perspective, or the localization of the object in the image. The problem of distinguishing between objects in this scenario requires consideration of feature levels from coarse to fine.

1.1.1 Bilinear CNN

Descriptors obtained through Convolutional Neural Networks (CNN) provide a great understanding of image details and are well suited for a wide range of Computer Vision tasks. However, these networks generally capture only first-order statistics of input features, which do not fully benefit from high-order statistics of local descriptors inside each region of the image [4]. Motivated by the past developments in handcrafted features, several works have proposed to integrate second-order pooling strategies, essentially employing covariance descriptors within CNN [4, 6, 7, 10]. This is the fundament that led to the design of second-order or Bilinear CNN whose representation power exceeds that of standard, first-order ones.

1.1.2 Matrix normalization

The output of Bilinear Pooling is a bilinear matrix which is used for making the classification. Still, to achieve high performance, normalization on the singular values of this covariance matrix needs to be conducted [4, 6, 11]. Traditional normalization methods require explicit computation of the Singular Value Decomposition (SVD) of the matrix, which is not well-suited for GPU platforms. An attempt of boosting the efficiency of this step is using Newton-Schulz [6] iteration for approximating the matrix square root, but this method uses the Lyapunov equation in the backward propagation, which requires costly computations. As a solution to the matrix normalization problem, which does not suffer from the previously mentioned limitations, Rank-1 Update Normalization (RUN) is proposed in [11]. The method is based on the power method algorithm, has a low-cost per-iteration, and is easily parallelizable.

1.1.3 Compact Bilinear Pooling

One drawback of second-order CNN is that when the bilinear matrix is vectorized before the final layer for classification, the resulted vector is orders of magnitude larger in comparison with regular CNN, which makes the Bilinear CNN memory-intensive and subject to overfitting. A solution to these limitations is Compact Bilinear Pooling, which approximates the feature map. The authors of [2] proposed two CBP methods, Random Maclaurin (RM) and Tensor Sketch (TS), which can reduce the feature dimensionality two orders of magnitude with very little loss in performance. The reference Rank-1 Update Normalization solution, in contrast to other normalization methods [6], can easily be used with compact features from RM and TS, benefiting from their improvements.

1.2 Overview of the Report

The rest of the report is structured as follows: section 2 provides a comprehensive theoretical review of the project, section 3 contains the results of the experiments, and section 4 summarizes our findings.

2 Theoretical Framework

2.1 Formal Problem Statement

In order to achieve high recognition performance it is crucial to perform normalization on singular values of the bilinear matrix. Previous approaches for this problem, SVD and Newton-Schulz (NS) iteration, have following limitations:

- Limited GPU support,
- High computational complexity,
- No Compact Bilinear Pooling support.

In order to overcome these limitations, a rank-1 update normalization (RUN) was proposed in [11]. Our aim is to develop and test the following RUN 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 $\mathbf{v_0} = [v_1, \ldots, v_D] \in \mathbb{R}^D$ , where $\{v_i\}_{i=1}^D$ are i.i.d. 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 $\boldsymbol{F}_K$
```

2.2 Previous approaches for matrix normalization

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. Then $\mathcal{F}$ is reshaped into a matrix $\mathbf{F} \in \mathbb{R}^{WH \times D}$. Bilinear pooling obtains the bilinear matrix by $\mathbf{B} = \mathbf{F}^\top \mathbf{F} \in \mathbb{R}^{D \times D}$. Below several approaches for $\mathbf{B}$ matrix normalization are discussed.

2.2.1 Singular values normalization

Traditional matrix square-root normalization is started by conducting singular value decomposition (SVD) on the bilinear matrix $\mathbf{B}$:

$$\mathbf{B} \rightarrow \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top. \tag{1}$$

Normalization $g(\mathbf{\Sigma})$ is conducted on singular values in an element-wise manner and the normalized feature is:

$$\hat{\mathbf{B}} \leftarrow \mathbf{U}g(\mathbf{\Sigma})\mathbf{U}^\top. \tag{2}$$

Unfortunately, SVD is not easily parallelizable and not well supported in the GPU platform, limiting its efficiency in training and inference.

2.2.2 Newton-Schulz iteration

Improved B-CNN [6] utilizes Newton-Schulz (NS) iteration to approximate the matrix square root in the forward propagation. Next, iterative matrix square root normalization (iSQRT) (proposed in [5]) is used to make the NS iteration differentiable and suitable for backward propagation, too.

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

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

where $\mathbf{Y}_k$ converges to $\mathbf{B}^{1/2}$. Only matrix-matrix product is used, therefore, thee algorithm 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. However, since D is large, computing Newton-Schulz (NS) iteration is still computationally expensive.

2.2.3 Compact Bilinear Pooling

As was mentioned before, $D \times D$ dimension is very high. Therefore, it is a) prone to over-fitting due to huge number of model parameters in the classifier; b) highly expensive in memory and computation when training the classifier. Compact Bilinear Pooling (CBP) [2] solves this issue. It treats the outer product used in bilinear pooling as a polynomial kernel embedding, and seeks to approximate the explicit kernel feature map. In particular, it rearranges the feature map $\mathcal{F}$ to $\mathbf{F} = [\mathbf{f}_1, \ldots, \mathbf{f}_{WH}]^\top$. Then the bilinear matrix $\mathbf{B}$ is calculated by

$$\mathbf{B} = \mathbf{F}^\top \mathbf{F} = \sum_{i=1}^{WH} \mathbf{f}_i \mathbf{f}_i^\top = \sum_{i=1}^{WH} \mathrm{h}\left(\mathbf{f}_i\right), \tag{4}$$

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

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \approx \langle \mathrm{vec}(\mathrm{h}(\mathbf{x})), \mathrm{vec}(\mathrm{h}(\mathbf{y})) \rangle, \tag{5}$$

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 $\tilde{\mathbf{B}} = \sum_{i=1}^{WH} \phi(\mathbf{f}_i)$. Low-dimensional features are better because they are less prone to over-fitting and speedup classifier training.

Since the compact bilinear feature $\tilde{\mathbf{B}}$ has lost 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 solve this challenge, MoNet [3] conducts SVD directly on the original local feature $\mathbf{F}$ instead of the bilinear matrix $\mathbf{B}$ and then conducts compact bilinear pooling. But once again, SVD lacks good support on GPU, which limits training and inference efficiency.

2.3 Rank-1 Update Normalization (RUN)

To overcome the limitations of previous methods, authors of [11] proposed a rank-1 update normalization (RUN) algorithm.

Through SVD, the bilinear feature B can be decomposed into $\mathbf{B} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$, where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_D]$ consists of orthogonal singular vectors, $\mathbf{\Sigma} = \mathrm{diag}([\sigma_1, \ldots, \sigma_{\mathrm{D}}])$ is a diagonal matrix where $\{\sigma_{\mathrm{d}}\}_{\mathrm{d}=1}^{\mathrm{D}}$ are singular values and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\mathrm{D}}$.

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

$$\mathbf{v}_k = \mathbf{B}\mathbf{v}_{k-1}, \quad \text{for} \quad k = 1, \ldots, K. \tag{6}$$

After that, the rank-1 matrix is constructed by

$$\mathbf{R}_k = \mathbf{B}\mathbf{v}_K \mathbf{v}_K^\top / \|\mathbf{v}_K\|_2^2 \tag{7}$$

Finally, the matrix $\mathbf{B}$ is updated by subtracting $\mathbf{R}_k$:

$$\mathbf{B}_k = \mathbf{B} - \epsilon \mathbf{R}_k, \tag{8}$$

where $\epsilon \in (0, 1]$ is a small positive constant.

According to classic convergence property of power method, if $\sigma_1 > \sigma_2$, $\mathbf{v}_K/\|\mathbf{v}_K\|_2$ will converge to $\mathbf{u}_1$. Hence, $\mathbf{B}_K$ converges to $\mathbf{B} - \epsilon\sigma_1\mathbf{u}_1\mathbf{u}_1^\top$. That is

$$\lim_{K\to\infty} \mathbf{B}_K = \mathbf{U}\text{diag}([\sigma_1(1-\epsilon), \sigma_2, \ldots, \sigma_\text{D}])\mathbf{U}^\top, \tag{9}$$

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. Specifically, it satisfies the following theorem:

Theorem 1 *Let $\mathbf{B}_K$ be obtained via Eq. 6-8, where $\mathbf{v}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Then the expectation of $\mathbf{B}_K$ is given by*

$$\mathbb{E}(\mathbf{B}_K) = \mathbf{U}\text{diag}([\sigma_1(1-\epsilon\alpha_1), \ldots, \sigma_\text{D}(1-\epsilon\alpha_\text{D})])\mathbf{U}^\top, \tag{10}$$

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

The proof of this theorem can be found in Appendix A. The operation in the right-hand side of Eq. 10 scales each singular value σ_i by $(1-\epsilon\alpha_i)$. Because of $1 \geq \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_D$ and *epsilon* $\in (0, 1]$:

$$0 \leq 1-\epsilon\alpha_1 \leq 1-\epsilon\alpha_2 \leq \ldots \leq 1-\epsilon\alpha_D \leq 1. \tag{11}$$

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.

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.

However, 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, the authors of [11] conducted normalization directly 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. Analogously to Eq. 6:

$$\mathbf{v}_k = \mathbf{F}^\top\mathbf{F}\mathbf{v}_{k-1}, \quad \text{for} \quad k = 1, \ldots, K. \tag{12}$$

where the entries of $\mathbf{v}_0$ are i.i.d. random variables with standard normal distribution. Then the updated feature map $\mathbf{F}_K$ is constructed as follows:

$$\mathbf{F}_k = \mathbf{F} - \eta\mathbf{F}\mathbf{v}_K\mathbf{v}_K^\top/\|\mathbf{v}_K\|_2^2 \tag{13}$$

where $\mathbf{v}_K$ is obtained from Eq. 12 and $\eta \in (0, 1]$ is a constant. This procedure is captured in the 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 $\sigma_{F,1} \neq \sigma_{F,2}$, $\mathbf{F}\mathbf{v}_k/\|\mathbf{v}_k\|_2^2$ and $\mathbf{v}_k/\|\mathbf{v}_k\|_2^2$ will converge to $\mathbf{u}_{F,1}$ and $\mathbf{v}_{F,1}$, respectively. Which means that:

$$\lim_{K\to\infty} \mathbf{F}_K = \mathbf{F} - \eta\sigma_{F,1}\mathbf{u}_{F,1}\mathbf{v}_{F,1}^\top, \tag{14}$$

whose singular values are the same as that of $\mathbf{F}$, except the largest one, which is decreased by $\eta\sigma_{F,1}$. Analogously to Theorem 1, expectation for $\mathbf{F}_K$ can be stated:

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

$$\mathbb{E}(\mathbf{F}_K) = \mathbf{U}_F \widehat{\Sigma}_F \mathbf{V}_F^\top, \tag{15}$$

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 $\mathrm{diag}[(\sigma_{\mathrm{F},1}(1-\eta\beta_1), \ldots, \sigma_{\mathrm{F,D}}(1-\eta\beta_{\mathrm{D}}))]$ *with* $0 \leq 1-\eta\beta_1 \leq 1-\eta\beta_2 \leq \ldots \leq 1-\eta\beta_D \leq 1$.

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:

$$\lim_{K\to\infty} \bar{\mathbf{B}}_K = \mathbf{V}_F \mathrm{diag}[(\sigma_{\mathrm{F},1}^2(1-\eta)^2, \ldots, \sigma_{\mathrm{F,D}}^2)]\mathbf{V}_F^\top, \tag{16}$$

Since $\mathbf{V}_F$ in Eq. 16 is equal to $\mathbf{U}$ in Eq. 9, if we set $(1-\epsilon)$ in Eq. 9 equal to $(1-\eta)^2$ in Eq. 16, $\mathbf{B}_K$ and $\bar{\mathbf{B}}_K$ will converge to the same matrix. But the advantage of updating $\mathbf{F}$ as in Eq. 13 over updating $\mathbf{B}$ as Eq. 8 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

$$\bar{\mathbf{b}}_K = \sum_{i=1}^{N} \phi(\mathbf{F}_K[i,:]), \tag{17}$$

where $\mathbf{F}_K[i,:]$ is the i-th row of $\mathbf{F}_K$, ϕ is implemented by TS or RM (see Section 1.1.3), 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. 13. After obtaining $\mathbf{F}_K$, it is feasible to conduct bilinear pooling (BP) or compact bilinear pooling (CBP). Figure 1 illustrates the architecture of the proposed network.

2.4 Summary

In this section Rank-1 Update Normalization (RUN) algorithm was discussed in detail. It's advantages are:

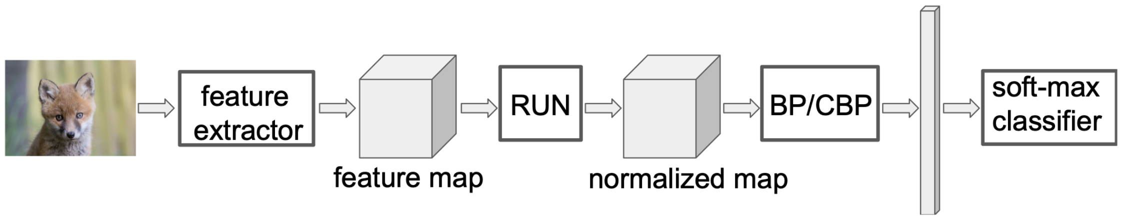

Figure 1: *The architecture of the proposed convolutional neural network. RUN denotes the proposed rank-1 update normalization, which takes input the feature map of the last convolutional layer. BP denotes the bilinear pooling and CBP represents compact bilinear pooling. The obtained BP/CBP feature is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

- RUN is based on iterations of matrix-vector multiplications, which are computationally cheaper than the matrix-matrix multiplications,
- RUN supports normalization on compact bilinear pooling features,
- RUN is differentiable. It can be plugged it into a neural network for an end-to-end training.

3 Results

We evaluate the benefits of bilinear pooling and different normalization methods of the bilinear features in terms of both accuracy and speed. For experiments three different datasets are used, which highlight the use-cases for the proposed method.

3.1 Datasets

Bilinear CNN are used in image classification problems for three main tasks: Fine-grained recognition, Scene recognition, and Texture recognition. We select for each of these tasks an appropriate dataset to validate the considered techniques:

- CUB200 (Caltech-UCSD Birds 200) [9] - Illustrative for the fine-grained recognition task. The dataset contains 200 bird species with 5994 images for training and 5794 images for testing.
- MIT indoor scenes [8] - Selected for the scene recognition task. The dataset contains 67 indoor scenarios with 4014 images for training and 1339 images for testing.
- DTD (Describable Textures Dataset) [1] - Representative for the texture recognition task. The dataset contains 47 describable texture attributes with 3760 images for training and 1880 images for testing.

3.2 Implementation details

We follow the same preprocessing techniques for all three datasets. The size of an input image is $448 \times 448 \times 3$, and the size of the last convolutional feature map, obtained with the backbone network, is $28 \times 28 \times 512$. For all the experiments, following the reference works [6, 7, 11], we adopt a two-phase training strategy. In the first phase, we perform weight update only for the last layers of the network, maintaining the weights of the backbone intact. We use Stochastic Gradient Descent optimizer with the initial learning rate set to 1 and a weight decay of 10^{-8}. In the second phase, we fine-tune the entire network, performing updates on all the layers. For this phase, we use the same optimizer, but with an initial learning rate 10^{-2} and a weight decay of 10^{-5}. As Inference Time we report the average GPU time, starting after the last convolutional layer, until the fully-connected classification layer, for the entire test set in batches of size 32. This interval represents all the computations for the Bilinear Matrix and its normalization, and provides a fair comparison for all of the following experiments.

3.3 Experiments

We perform experiments to compare three models: BCNN, iBCNN and RUN, on the three datasets described. We use the same training settings for all experiment and report the results for both phases of training. In the tables below, the suffix '-fc' denotes the model after the parameters of the last fully connected layer were updated. The suffix '-all' denotes the model fully trained, including the parameters of the feature extracting backbone.

3.3.1 BCNN

In order to asses the benefits of the normalization procedure proposed in RUN [11], we firstly implemented and tested the original Bilinear-CNN setup: The features are extracted from the second-to-last layer of a VGG-D(16) network pretrained on ImageNet. From that, we obtain the bilinear features and normalize them using the element-wise signed squared root followed by L_2 normalization. The resulting vector is run through a fully connected layer to obtain class scores.

Dataset	Accuracy-fc	Accuracy-all	Inference Time (ms)
CUB200	74.54	84.45	5.678
DTD	70.79	70.85	2.42
MIT	75.44	77.76	4.613

3.3.2 iBCNN

The idea from Lin et al. [6] is to improve the BCNN by performing *matrix square-root* normalization (before the element-wise signed squared root normalization). Following the results from the aforementioned paper, for a good accuracy with reasonable computation time, we perform 10 iterations of NS in the forward pass and also when solving Lyapunov equation to compute gradients. The rest of the network, feature extraction and training process are kept the same as in the previous experiment.

Dataset	Accuracy-fc	Accuracy-all	Inference Time (ms)
CUB200	76.89	84.92	100.605
DTD	70.80	72.45	143.62
MIT	75.60	78.88	100.640

3.3.3 RUN

In this experiment we evaluate the effectiveness of Rank-1 Update Normalization. We fix the number of iterations for power method $K = 2$ as stated in the paper and use $\eta = 1$ for all datasets. An important finding is that RUN works better with uniform distribution initialization, compared to the normal distribution used in the original paper for initializing the vector for power method. This small looking change proved to play an important role in obtaining stable training and a well performing network.

Dataset	Accuracy-fc	Accuracy-all	Inference Time (ms)
CUB200	77.3	85.47	9.579
DTD	70.96	71.01	5.693
MIT	76.19	80.30	9.707

3.3.4 CBP

Additionally, we evaluate the Rank-1 Update Normalization with features obtained through Compact Bilinear Pooling. We use Tensor Sketch, reducing the dimensionality from 262144 features to 10000 features, and the same parameters for RUN as before. The accuracy on the MIT dataset after fully training the network is 78.65. Therefore, RUN performs well even with limited number of features.

4 Conclusions

We implemented the fast rank-1 update normalization (RUN) method that was proposed in [11]. Since it only takes several times of matrix-vector multiplications, the RUN algorithm not only has low computation complexity in theory but also is well supported in the GPU platform in practice. More importantly, the RUN supports normalization on compact bilinear features, which have broken the matrix structure. Moreover, RUN is differentiable and therefore can be easily plugged into a convolutional neural network, which supports an end-to-end training. In our experiments we managed to reproduce the most significant parts of the original paper. The experiments proved that RUN is significantly faster than iBCNN with Newton-Schulz iteration while leading to similar or even superior accuracy.

References

[1] M. Cimpoi, Subhransu Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014.

[2] Y. Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 317–326, 2016.

[3] Mengran Gou, Fei Xiong, Octavia I. Camps, and Mario Sznaier. Monet: Moments embedding network. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3175–3183. IEEE Computer Society, 2018.

[4] Catalin Ionescu, O. Vantzos, and C. Sminchisescu. Matrix backpropagation for deep networks with structured layers. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2965–2973, 2015.

[5] Peihua Li, Jiangtao Xie, Qilong Wang, and Zilin Gao. Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 947–955. IEEE Computer Society, 2018.

[6] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. *ArXiv*, abs/1707.06772, 2017.

[7] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnns for fine-grained visual recognition. *arXiv: Computer Vision and Pattern Recognition*, 2015.

[8] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009.

[9] C. Wah, S. Branson, P. Welinder, P. Perona, and Serge J. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[10] Kaicheng Yu and M. Salzmann. Statistically motivated second order pooling. *ArXiv*, abs/1801.07492, 2018.

[11] Tan Yu, Y. Cai, and Ping Li. Toward faster and simpler matrix normalization via rank-1 update. In *ECCV*, 2020.

Appendices

A Proof of the Theorem 1

In this section, we prove the Theorem 1 in Section 2.3.

From Eq. 8:

$$\mathbf{B}_k = \mathbf{B} - \epsilon \mathbf{R}_k \tag{18}$$

where from Eq. :

$$\mathbf{R}_k = \mathbf{B}\mathbf{v}_K\mathbf{v}_K^\top / \|\mathbf{v}_K\|_2^2 \tag{19}$$

Using SVD we decompose:

$$\mathbf{B} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top \tag{20}$$

According to Eq. 6:

$$\mathbf{v}_k = \mathbf{B}^k\mathbf{v}_0 = \mathbf{U}\boldsymbol{\Sigma}^{\mathbf{K}}\mathbf{U}^\top\mathbf{v}_0 = \mathbf{U}\boldsymbol{\Sigma}^{\mathbf{K}}\mathbf{a} \tag{21}$$

where $\mathbf{a} = \mathbf{U}^T\mathbf{v}_0$.

Plugging Eq. 20 and Eq. 21 into Eq. 19, we have:

$$\mathbf{R}_K = \frac{\mathbf{U}\boldsymbol{\Sigma}^{K+1}\mathbf{a}\mathbf{a}^\top\boldsymbol{\Sigma}^K\mathbf{U}^\top}{\mathbf{a}^\top\boldsymbol{\Sigma}^{2K}\mathbf{a}} = \mathbf{U}\mathbf{H}\mathbf{U}^\top \tag{22}$$

where $\mathbf{H} = (\boldsymbol{\Sigma}^{K+1}\mathbf{a}\mathbf{a}^\top\boldsymbol{\Sigma}^K)/(\mathbf{a}^\top\boldsymbol{\Sigma}^{2K}\mathbf{a})$.

As $\mathbf{v}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$, therefore $a \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. That is, $\mathbf{a}$'s entries $a_1, a_2, \ldots, a_d$ are i.i.d random variables with normal distribution. Therefore, the expectation of each off-diagonal entry of $\mathbf{H}$ is 0. Hence, $\mathbb{E}(\mathbf{H})$ is a diagonal matrix. We rewrite $\mathbb{E} = \mathrm{diag}(h_1, \ldots, h_D)$ and

$$h_l = \mathbb{E}(\sigma_l(a_l\sigma_l^k)^2 / \sum_{i=1}^{D}(a_i\sigma_i^k)^2) = \sigma_l\alpha_l \tag{23}$$

where

$$\alpha_l = \mathbb{E}((a_l\sigma_l^k)^2 / \sum_{i=1}^{D}(a_i\sigma_i^k)^2) \tag{24}$$

In this case proving Theorem 1 is equivalent to proving that $\alpha_s \geq \alpha_t$ if $s < t$. As we know

$$\alpha_s - \alpha_t = \mathbb{E}\left(\frac{a_s^2\sigma_s^{2k} - a_s^2\sigma_t^{2k}}{\sum_{i=1}^{D} a_i^2\sigma_i^{2k}}\right) \tag{25}$$

We define $b_i = a_i^2$ and $y_i = \sigma_i^{2k}$, then the goal is to prove:

$$\alpha_s - \alpha_t = \mathbb{E}\left(\frac{b_s y_s - b_t y_t}{\sum_{i=1}^{D} b_i y_i}\right) \geq 0, \quad if s < t \tag{26}$$

As $y_s \geq y_t$ and $y_1 \geq y_2 \geq \ldots \geq y_D$, we obtain

$$\frac{b_s y_s - b_t y_t}{\sum_{i=1}^{D} b_i y_i} \geq \frac{y_t}{y_1} \frac{b_s - b_t}{\sum_{i=1}^{D} b_i} \tag{27}$$

Thus,

$$\mathbb{E}\left(\frac{b_s y_s - b_t y_t}{\sum_{i=1}^{D} b_i y_i}\right) \geq \frac{y_t}{y_1} \mathbb{E}\left(\frac{b_s - b_t}{\sum_{i=1}^{D} b_i}\right) \tag{28}$$

Since $\{a_i\}_1^D$ are *i.i.d.*, $\{b_i\}_1^D$ are also *i.i.d.*. Therefore,

$$\mathbb{E}\left(\frac{b_s - b_t}{\sum_{i=1}^{D} b_i}\right) = \mathbb{E}\left(\frac{b_s}{\sum_{i=1}^{D} b_i}\right) - \mathbb{E}\left(\frac{b_t}{\sum_{i=1}^{D} b_i}\right) = 0 \tag{29}$$

Plugging Eq. 29 into Eq. 28, we obtain

$$\mathbb{E}\left(\frac{b_s y_s - b_t y_t}{\sum_{i=1}^{D} b_i y_i}\right) \geq 0, \quad if s < t \tag{30}$$