# Discriminative Covariance Oriented Representation Learning for Face Recognition with Image Sets

Wen Wang[1,2], Ruiping Wang[1,2,3], Shiguang Shan[1,2,3], Xilin Chen[1,2,3]
[1]Key Laboratory of Intelligent Information Processing of Chinese Academy of Sciences (CAS),
Institute of Computing Technology, CAS, Beijing, 100190, China
[2]University of Chinese Academy of Sciences, Beijing, 100049, China
[3]Cooperative Medianet Innovation Center, China
wen.wang@vipl.ict.ac.cn, {wangruiping, sgshan, xlchen}@ict.ac.cn

In this supplementary material, we give a detailed derivation of the gradient computing formulas for solving the optimization problem in Sec. 4.3. Then a detailed description is given for the structure of the feature learning network. Besides, we show some examples for the datasets used in our experiments.

## 1. Gradient derivation

In this section, we derive the formulas in Sec. 4.3 for computing the gradient of objective function $J$ with respect to network parameters $\Theta$ in the Graph Embedding scheme and the Softmax Regression scheme respectively.

### 1.1. Graph Embedding Scheme

As defined by Eq. (7) in Sec. 4.3.1, the optimization objective is defined as:

$$\Theta = \arg\min_{\Theta} J(\Theta), \tag{S1}$$

where

$$J(\Theta) = \frac{1}{4}\sum_{i,j} A_{ij} LEM^2(C_i, C_j) \tag{S2}$$

To derive the gradient of $J$ with respect to $\Theta$, we first compute the derivative of $J$ with respect to $h_i$ according to the Chain Rule [5].

$$\begin{aligned}\frac{\partial J}{\partial h_i} &= \frac{1}{4}\sum_j A_{ij}\frac{\partial}{h_i}\|\log(C_i)-\log(C_j)\|_F^2\\ &= \frac{1}{2}\sum_j A_{ij}\frac{\partial}{h_i}\{Tr(\log(C_i)-\log(C_j))\}\left(\log(C_i)-\log(C_j)\right)\\ &= \frac{1}{2}\sum_j A_{ij}\frac{\partial}{h_i}(Tr\log(C_i))\left(\log(C_i)-\log(C_j)\right)\end{aligned} \tag{S3}$$

By exploiting the fact that $Tr\log(X) = \ln\det(X)$, we have

$$\begin{aligned}\frac{\partial J}{\partial h_i} &= \frac{1}{2}\sum_j A_{ij}\frac{\partial}{h_i}\ln\det(C_i)\cdot(\log(C_i)-\log(C_j))\\ &= \frac{1}{2}\sum_j A_{ij}\frac{\partial}{h_i}\ln\det\left((h_i)^T J_{N_i} h_i\right)\\ &\cdot(\log(C_i)-\log(C_j))\\ &= \sum_j A_{ij} J_{N_i} h_i C_i^{-1}\cdot(\log(C_i)-\log(C_j))\,.\end{aligned} \tag{S4}$$

Then we accordingly back propagate to the successive layer using the same mechanism of the Siamese network in [3].

### 1.2. Softmax Regression Scheme

The optimization objective is formulated as Eq. (9) in Sec. 4.3.2, *i.e.*,

$$\Theta' = \arg\min_{\Theta'} J(\Theta'), \tag{S5}$$

where

$$J(\Theta') = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} 1\{y_i = j\}\log(o_{ij}), \tag{S6}$$

Firstly, we refer to [2] and give the gradient of $J$ with respect to parameters $W$ and $b$ for the softmax regression machine as follows,

$$\begin{aligned}\frac{\partial J}{\partial W_j} &= -\frac{1}{n}\sum_i^n v_i(1\{y_i=j\}-o_{ij}) + \lambda W_j\\ \frac{\partial J}{\partial b_j} &= -\frac{1}{n}\sum_i^n(1\{y_i=j\}-o_{ij})\end{aligned} \tag{S7}$$

where $W_j$ denotes the $j$-row vector of $W$, and $b_j$ is the $j$-th element of $b$.

Since this is the output layer, we can directly measure the error term $\Delta o_i$ by the difference between the network output $o_i$ and the true target value $y_i$, *i.e.*, $\Delta o_i = y_i - o_i$.

To further compute the gradient of the update layers using back-propagation, we also need start with computing the derivative of $J$ with respect to $h_i$, i.e.,

$$(\frac{\partial J}{\partial h_i})_{pq} = \sum_{kl} (\Delta C_i)_{kl} \frac{\partial (C_i)_{kl}}{\partial (h_i)_{pq}}, \tag{S8}$$

where $\Delta C_i = \frac{\partial J}{\partial C_i}$ and the subscripts $(\cdot)_{pq}$ denote the element of the $p$-th row $q$-th column. Since we have $C_i = h_i^T J_{N_i} h_i$, the above equation is derived as:

$$\begin{aligned}(\frac{\partial J}{\partial h_i})_{pq} = & \sum_{k \neq q = l} (\Delta C_i)_{kq} (h_i^T J_{N_i})_{kp} \\ & + \sum_{k = q \neq l} (\Delta C_i)_{ql} (J_{N_i} h_i)_{pl} \\ & + (\delta_{C_i})_{qq} (h_i^T J_{N_i})_{qp} + (\Delta C_i)_{qq} (J_{N_i} h_i)_{pq} \\ = & \sum_k (\Delta C_i)_{kq} (h_i^T J_{N_i})_{kp} + \sum_l (\Delta C_i)_{ql} (J_{N_i} h_i)_{pl} \\ = & (J_{N_i}^T h_i \Delta C_i + J_{N_i} h_i \Delta C_i^T)_{pq},\end{aligned} \tag{S9}$$

*i.e.*,

$$\frac{\partial J}{\partial h_i} = J_{N_i}^T h_i \Delta C_i + J_{N_i} h_i \Delta C_i^T. \tag{S10}$$

Then we give the derivation of $\Delta C_i$.

$$\Delta C_i = \frac{\partial J}{\partial C_i} = \sum_{kl} (\Delta v_i)_{kl} \frac{\partial (\log C_i)_{kl}}{\partial C_i}. \tag{S11}$$

Let $C_i = U_i \Sigma_i U_i^T$ be the eigen-decomposition of $C_i$, i.e., $(\Sigma_i)_{tt}$ is an eigenvalue of $C_i$ and the $t$-th column vector of $U_i$ is the corresponding eigenvector. Thus its log-covariance matrix is formulated as Eq. (5) in Sec. 4.3.1, *i.e.*, $\log C_i = U_i \log \Sigma_i U_i^T$. Next we attempt to derive the numerical expression of $\frac{\partial (\log C_i)_{kl}}{\partial C_i}$ element by element.

$$\begin{aligned}\frac{\partial (\log C_i)_{kl}}{\partial (C_i)_{pq}} &= \frac{\partial (U_i \log \Sigma_i U_i^T)_{kl}}{\partial (C_i)_{pq}} \\ &= \sum_t \frac{\partial (U_i \log \Sigma_i U_i^T)_{kl}}{\partial (\Sigma_i)_{tt}} \frac{\partial (\Sigma_i)_{tt}}{\partial (C_i)_{pq}} \\ &\quad + \sum_{st} \frac{\partial (U_i \Sigma_i U_i^T)_{kl}}{\partial (U_i)_{st}} \frac{\partial (U_i)_{st}}{\partial (C_i)_{pq}}.\end{aligned} \tag{S12}$$

To derive $\frac{\partial (\Sigma_i)_{tt}}{\partial (C_i)_{pq}}$ and $\frac{\partial (U_i)_{st}}{\partial (C_i)_{pq}}$, we refer to [1] and introduce a lemma.

**Lemma 1** *Let $A$ is real and symmetric, $\lambda_i$ and $v_i$ are distinct eigenvalues and eigenvectors of $A$ with $v_i^T v_i = 1$, then*

$$\begin{aligned}\partial \lambda_i &= v_i^T \partial(A) v_i \\ \partial v_i &= (\lambda_i I - A)^+ \partial(A) v_i,\end{aligned} \tag{S13}$$

*where $A^+$ denote the pseudo inverse (or Moore-Penrose inverse) of $A$.*

Based on this lemma, Eq. (S12) equals to the equation below.

$$\begin{aligned}\frac{\partial (\log C_i)_{kl}}{\partial (C_i)_{pq}} = & \sum_t (U_i)_{kt} (\Sigma_i)_{tt}^{-1} (U_i)_{lt} (U_i)_{pt} (U_i)_{qt} \\ & + \sum_{st} (\delta_{ls} (U_i \log \Sigma_i)_{kt} + \delta_{ks} (\log \Sigma_i U_i^T)_{tl}) \\ & ((\Sigma_i)_{tt} I - C_i)_{sp}^+ (U_i)_{qt},\end{aligned} \tag{S14}$$

where $\delta_{ij} = 1$ when $i = j$ and $\delta_{ij} = 0$ otherwise.

By putting Eq. (14) into Eq. (11), $\Delta C_i$ can be derived as follows:

$$\begin{aligned}(\Delta C_i)_{pq} = & \sum_{klt} (U_i)_{pt} (U_i^T)_{tk} (\Delta v_i)_{kl} (U_i)_{lt} (\Sigma_i)_{tt}^{-1} (U_i^T)_{tq} \\ & + \sum_{klt} ((\Sigma_i)_{tt} I - C_i)_{pl}^+ (\Delta v_i^T)_{lk} (U_i \log \Sigma_i)_{kt} (U_i^T)_{tq} \\ & + \sum_{klt} ((\Sigma_i)_{tt} I - C_i)_{pk}^+ (\Delta v_i)_{kl} (U_i \log \Sigma_i)_{lt} (U_i^T)_{tq} \\ = & \Big( \Delta v_i U_i \Sigma_i^{-1} U_i^T + ((\Sigma_i)_{tt} I - C_i)^+ \Delta v_i^T \log C_i \\ & + ((\Sigma_i)_{tt} I - C_i)^+ \Delta v_i \log C_i \Big)_{pq}\end{aligned} \tag{S15}$$

i.e.,

$$\begin{aligned}&\Delta C_i \\ &= \Delta v_i U_i \Sigma_i^{-1} U_i^T + ((\Sigma_i)_{tt} I - C_i)^+ (\Delta v_i + \Delta v_i^T) \log C_i.\end{aligned} \tag{S16}$$

Here $\Delta v_i$ can be back propagated from the error term $\Delta o_i$ in the output layer as follows:

$$\Delta v_i = \Delta o_i W^T = (y_i - o_i) W^T. \tag{S17}$$

Plugging this into Eq. (S14), we obtain:

$$\begin{aligned}&\Delta C_i \\ &= (y_i - o_i) W^T U_i \Sigma_i^{-1} U_i^T + ((\Sigma_i)_{tt} I - C_i)^+ \\ &\quad ((y_i - o_i) W^T + W (y_i - o_i)^T) \log C_i.\end{aligned} \tag{S18}$$

Finally by using Eq. (S18) to substitute $\Delta C_i$, the derivative $\frac{\partial J}{\partial h_i}$ is obtained. Hence, we can use back propagation algorithm to successively compute the gradients for the earlier layer similarly with [6].

| Layer | Output Size | Kernel Size | Kernel Num | Stride |
|---|---|---|---|---|
| conv1 | $52 \times 44 \times 20$ | $4 \times 4 \times 3$ | 20 | 1 |
| pool1 | $26 \times 22 \times 20$ | – | – | 2 |
| conv2 | $24 \times 20 \times 40$ | $3 \times 3 \times 20$ | 40 | 1 |
| pool2 | $12 \times 10 \times 40$ | – | – | 2 |
| conv3 | $10 \times 8 \times 60$ | $3 \times 3 \times 40$ | 60 | 1 |
| pool3 | $5 \times 4 \times 60$ | – | – | 2 |
| conv4 | $4 \times 3 \times 80$ | $2 \times 2 \times 60$ | 80 | 1 |
| full | 160 | – | – | 1 |

Table 1: Feature learning network configuration.

## 2. Feature Learning Network Configuration

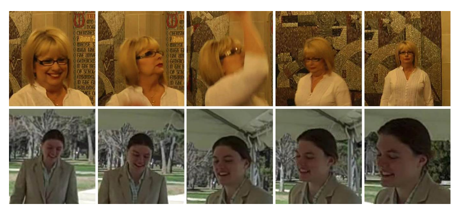
Due to the property of compact structure and thereby relatively low computation complexity, we choose a feature learning network which has similar structure with the one in [6]. It takes the RGB image of size $55 \times 47$ as input and consists of four convolutional layers and a fully-connected layer. The first three convolutional layers are followed by max-pooling and the ReLU [4] is used to equip with all the convolutional layers and the fully-connected layer. The fully-connected layer is connected to both the third and fourth convolutional layers. By passing through the feature learning network, a 160-dimensional feature vector is extracted for each sample image. Besides, the network structure is shown in Tab. 1, where the output size, kernel size, number of kernels and stride are indicated for each layer.

## 3. Dataset Examples

Fig. 1 shows some examples for the three datasets used in our experiments, *i.e*., YouTube Celebrities (YTC), YouTube Face DB (YTF) and Point-and-Shoot Challenge (PaSC).

(a) YTC

(b) YTF



(c) PaSC

Figure 1: Some examples of the datasets.

## References


[1] K. T. Abou-Moustafa. On derivatives of eigenvalues and eigenvectors of the generalized eigenvalue problem., October 2010. McGill Technical Report.

[2] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] S. Chopra, R. Hadsell, and Y. Lecun. Learning a similarity metric discriminatively, with application to face verification, 2005. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[4] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine learning (ICML)*, 2010.

[5] K. B. Petersen, M. S. Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 2008.

[6] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.