Supplementary Material for
Adaptive Label Noise Cleaning with Meta-Supervision for Deep Face Recognition

Yaobin Zhang, Weihong Deng*, Yaoyao Zhong, Jiani Hu
Beijing University of Posts and Telecommunications
{zhangyaobin, whdeng, zhongyaoyao, jnhu}@bupt.edu.cn

Xian Li, Dongyue Zhao, Dongchao Wen
Canon Innovative Solution (Beijing) Co., Ltd
{lixian, zhaodongyue, wendongchao}@canon-is.com.cn

Abstract

In this supplementary material, we present fully detailed information on 1) derivation of the meta-learning update (formula 11) in the paper; 2) the threshold-aware loss; 3) network structure of the GCN cleaner.

1. Meta-Learning Update

In the paper, we combine the meta-train and meta-test loss to get the final meta update loss as

$$\mathcal{L}^{\text{meta}} = \gamma \mathcal{L}^{\text{train}}(\theta) + (1-\gamma)\mathcal{L}^{\text{test}}(\theta') \tag{1}$$

Recall the update equation of SGD, the parameter θ is updated as

$$\theta \leftarrow \theta - \alpha \frac{1}{B} \sum_{b=1}^{B} \nabla_{\theta} \mathcal{L}_b^{\text{meta}}(\theta, \theta') \tag{2}$$

The computation of Eq. 11 in the paper by backpropagation can be understood by the following derivation:

$$\begin{aligned} \alpha \tfrac{1}{B} \sum_{b=1}^{B} \nabla_{\theta} \mathcal{L}_b^{\text{meta}}(\theta, \theta') &= \alpha \tfrac{1}{B} \sum_{b=1}^{B} \left(\gamma \tfrac{\partial \mathcal{L}_b^{\text{train}}(\theta)}{\partial \theta} + (1-\gamma) \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta} \right) \\ &= \tfrac{\gamma \cdot \alpha}{B} \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{train}}(\theta)}{\partial \theta} + \tfrac{(1-\gamma)\cdot \alpha}{B} \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} \cdot \tfrac{\partial \theta'}{\partial \theta} \end{aligned} \tag{3}$$

where $\frac{\partial \theta'}{\partial \theta}$ is determined in the meta-training phase and can be extracted from the sum, therefore the second term

$$\begin{aligned} \tfrac{(1-\gamma)\cdot \alpha}{B} \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} \cdot \tfrac{\partial \theta'}{\partial \theta} &= \tfrac{(1-\gamma)\cdot \alpha}{B} \cdot \tfrac{\partial \theta'}{\partial \theta} \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} \\ &= \tfrac{(1-\gamma)\cdot \alpha}{B} \left(1 - \tfrac{\alpha}{B} \sum_{b=1}^{B} \tfrac{\partial^2 \mathcal{L}_b^{\text{train}}(\theta)}{\partial \theta^2} \right) \cdot \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} \\ &= \tfrac{(1-\gamma)\cdot \alpha}{B} \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} - \tfrac{(1-\gamma)\cdot \alpha^2}{B^2} \sum_{b=1}^{B} \tfrac{\partial^2 \mathcal{L}_b^{\text{train}}(\theta)}{\partial \theta^2} \sum_{b=1}^{B} \tfrac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} \end{aligned} \tag{4}$$

By substituting Eq. 3 and Eq. 4 into Eq. 2, the meta-update formula is as shown in the paper:

$$\theta \leftarrow \theta - \frac{\gamma \cdot \alpha}{B} \sum_{b=1}^{B} \frac{\partial \mathcal{L}_b^{\text{train}}(\theta)}{\partial \theta} - \frac{(1-\gamma)\cdot \alpha}{B} \sum_{b=1}^{B} \frac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} + \frac{(1-\gamma)\cdot \alpha^2}{B^2} \sum_{b=1}^{B} \frac{\partial^2 \mathcal{L}_b^{\text{train}}(\theta)}{\partial \theta^2} \sum_{b=1}^{B} \frac{\partial \mathcal{L}_b^{\text{test}}(\theta')}{\partial \theta'} \tag{5}$$

It can be seen that three terms control the gradient descent of parameter θ. The first term optimize meta-train set with learning rate $\gamma \cdot \alpha$ towards θ', then the second term optimize meta-test set on the updated model parameter θ' with learning rate $(1-\gamma)\cdot\alpha$. So that the parameter is trained to perform well on both meta-train and meta-test domains. The third term applies gradient ascent on parameter θ' with learning rate $\frac{(1-\gamma)\cdot\alpha^2}{B}\sum_{b=1}^{B}\frac{\partial^2\mathcal{L}_b^{\text{train}}(\theta)}{\partial\theta^2}$, which is constrained by the second-order derivative from meta-train domain. For instance, when the model reaches the local-minimum of the meta-train set, which means the second-order derivative is positive, then the third term corrects the second term to step less on the meta-test set to retain the learned knowledge from the meta-train set and to ensure convergence. Inversely, if the model is not stable on the meta-train set, then it follows more on the meta-test set to find the space that is fitted to both domains.

2. Threshold-Aware Loss

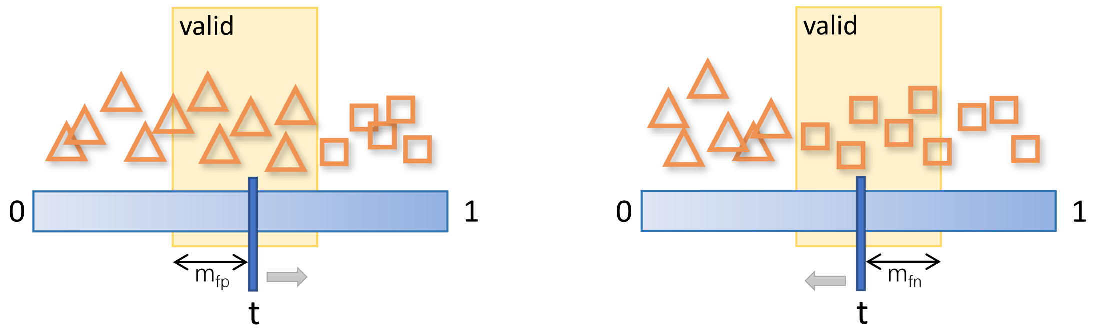
To effectively train the threshold adapter $\mathbf{T}$, a threshold-aware loss $\mathcal{L}^{\text{th}}$ is designed in the paper as below:

$$\mathcal{L}^{\text{th}} = -\frac{1}{n}\sum_{i=1}^{n}\left[\hat{\boldsymbol{y}}_i^t \cdot \log\left(1 - \left[1 - \boldsymbol{p}_i^t - [1 - \boldsymbol{t} - m_{\text{fn}}]_+\right]_+\right) + \left(1 - \hat{\boldsymbol{y}}_i^t\right) \cdot \log\left(1 - \left[\boldsymbol{p}_i^t - [\boldsymbol{t} - m_{\text{fp}}]_+\right]_+\right)\right] \tag{6}$$

For sample $\left(\boldsymbol{x}_i^t, \hat{\boldsymbol{y}}_i^t\right)$ in class (X^t, Y^t), we discuss the form of $\mathcal{L}^{\text{th}}$ in different situation in Algorithm 1. In fact, there are only two cases (Eq. 7 and Eq.10) where the gradients with repect to $\boldsymbol{t}$ are valid. In Figure 1, we show a simplified situation

Algorithm 1 Deviation of the threshold-aware loss.

```
Require: Threshold-aware loss $\mathcal{L}^{\text{th}}$, a sample $\left(\boldsymbol{x}_i^t, \hat{\boldsymbol{y}}_i^t\right)$ in class $(X^t, Y^t)$.
  if $\hat{\boldsymbol{y}}_i^t = 0$ then
    if $\boldsymbol{t} > m_{\text{fp}}$ then
      if $\boldsymbol{p}_i^t > (\boldsymbol{t} - m_{\text{fp}})$ then
        $$\mathcal{L}^{\text{th}} = -\log\left(1 - \left(\boldsymbol{p}_i^t - (\boldsymbol{t} - m_{\text{fp}})\right)\right) \tag{7}$$
      else
        $$\mathcal{L}^{\text{th}} = 0 \tag{8}$$
      end if
    else
      $$\mathcal{L}^{\text{th}} = -\log\left(1 - \boldsymbol{p}_i^t\right) \tag{9}$$
    end if
  else
    if $\boldsymbol{t} + m_{\text{fn}} < 1$ then
      if $\boldsymbol{p}_i^t < (\boldsymbol{t} + m_{\text{fn}})$ then
        $$\mathcal{L}^{\text{th}} = -\log\left(1 - \left(\boldsymbol{t} + m_{\text{fn}} - \boldsymbol{p}_i^t\right)\right) \tag{10}$$
      else
        $$\mathcal{L}^{\text{th}} = 0 \tag{11}$$
      end if
    else
      $$\mathcal{L}^{\text{th}} = -\log \boldsymbol{p}_i^t \tag{12}$$
    end if
  end if
```



(a) Negative samples

(b) Positive samples

Figure 1: Training of threshold adaptor.

that the negative (triangle) and positive (rectangle) samples are well separated, but the threshold $\boldsymbol{t}$ is biased. For a noise sample, which means $\hat{\boldsymbol{y}}_i^t = 0$, if the predicted threshold $\boldsymbol{t}$ is greater than m_{fp} and the predicted score for the sample $\boldsymbol{p}_i^t$ is greater than $\boldsymbol{t} - m_{\text{fp}}$, we treat it as a false positive sample. As shown in Figure 1a, the gradients of false negatives pull the threshold $\boldsymbol{t}$ along the arrow in the positive direction. The threshold-aware loss is designed in cross-entropy way as in Eq. 7, and the gradient to the parameter ϕ in the threshold adapter is

$$\nabla_\phi \mathcal{L}^{\text{th}} = -\frac{\nabla_\phi \boldsymbol{t}}{1 - (\boldsymbol{p}_i^t - (\boldsymbol{t} - m_{\text{fp}}))} \tag{13}$$

Similarly, for a signal sample, if the predicted threshold $\boldsymbol{t}$ is less than $1 - m_{\text{fn}}$ and the predicted score for the sample $\boldsymbol{p}_i^t$ is less than $\boldsymbol{t} + m_{\text{fn}}$, we treat it as a false negative sample. As shown in Figure 1b, the gradients of false positives pull the threshold $\boldsymbol{t}$ along the arrow in the negative direction. The threshold-aware loss is designed in cross-entropy way as in Eq. 10, and the gradient to the parameter ϕ in the threshold adapter is

$$\nabla_\phi \mathcal{L}^{\text{th}} = \frac{\nabla_\phi \boldsymbol{t}}{1 - (\boldsymbol{t} + m_{\text{fn}} - \boldsymbol{p}_i^t)} \tag{14}$$

3. Graph Convolutional Network

In this paper, the GCN cleaner is designed as a binary vertex classification network, of which structure is designed similar to a recent GCN-based data cleaning work [4]. Specifically, the GCN forward propagation function [1, 2] from layer l to layer $l+1$ of one sample vertex i on the graph is

$$\boldsymbol{h}_i^{(l+1)} = \sigma\left[F_{j\in\mathcal{N}_i}\left(\boldsymbol{h}_j^{(l)}\right)\mathbf{W}^{(l)}\right] \tag{15}$$

where $\boldsymbol{h}_i^{(l)}$ is the embedding representation of vertex i in the l-th layer, $\mathbf{W}^{(l)}$ is a learnable linear transformation in the l-th layer, and σ denotes the activation function, where sigmoid is used in the last layer, and ReLU is used in the other layers. F is designed as the transforming function that aggregates vertex i and its neighbors with the similarity between vertices as the weight, and outputs a new expression of the vertex:

$$F_{j\in\mathcal{N}_i}\left(\boldsymbol{h}_j^{(l)}\right) = \left[\boldsymbol{h}_i^{(l)} \| \sum_{j\in\mathcal{N}_i} \sigma\left(\tilde{s}_{ij}\boldsymbol{h}_j^{(l)}\mathbf{A}^{(l)} + \mathbf{b}^{(l)}\right)\right] \tag{16}$$

where $\mathcal{N}_i$ is a collection of all neighbors of vertex i, $\tilde{s}_{ij} = \frac{s_{ij}}{\sum_{k\in\mathcal{N}_i} s_{ik}}$ is the normalized similarity score between vertex i and vertex j, $\|$ is the concatenation operator, $\mathbf{A}^{(l)}$ and $\mathbf{b}^{(l)}$ are deployed to learn the face aggregating principle in the l-th layer. Therefore, there are three learnable parameters $\mathbf{W}^{(l)}$, $\mathbf{A}^{(l)}$ and $\mathbf{b}^{(l)}$ in one layer of the cleaner. The network is implemented with DGL [3] by PyTorch deep learning framework.

References

[1] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[2] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[3] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[4] Yaobin Zhang, Weihong Deng, Mei Wang, Jiani Hu, Xian Li, Dongyue Zhao, and Dongchao Wen. Global-local gcn: Large-scale label noise cleansing for face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7731–7740, 2020.