Content-Based Music-Image Retrieval Using Self- and Cross-Modal Feature Embedding Memory —Supplementary Material—

Takayuki Nakatsuka Masahiro Hamasaki Masataka Goto
National Institute of Advanced Industrial Science and Technology (AIST)
{takayuki.nakatsuka, masahiro.hamasaki, m.goto}@aist.go.jp

A. Introduction

In the supplementary material, we analyze our loss functions and present the pseudocode of the proposed self- and cross-modal feature embedding memory (SCFEM) mechanism.

B. Analysis of Loss Functions $\mathcal{L}_{self}$ and $\mathcal{L}_{cross}$

In practice, the general pair weighting (GPW) formulation [2] of $\mathcal{L}_{self}$ can be obtained by using Eq. (2) as written in our paper as follows:

$$\mathcal{F}(\boldsymbol{S}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{e=0}^{E-1}\left(\sum_{(\mathbf{x}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{N}} w_{ij}^{\mathbf{x}} S_{ij}^{\mathbf{x}} + \sum_{(\mathbf{y}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{N}} w_{ij}^{\mathbf{y}} S_{ij}^{\mathbf{y}}\right) - \frac{1}{m}\sum_{i=1}^{m}\sum_{e=0}^{E-1}\left(\sum_{(\mathbf{x}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{P}} w_{ij}^{\mathbf{x}} S_{ij}^{\mathbf{x}} + \sum_{(\mathbf{y}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{P}} w_{ij}^{\mathbf{y}} S_{ij}^{\mathbf{y}}\right). \tag{9}$$

Here, $w_{ij}^{\mathbf{x}} = \left|\frac{\partial\mathcal{L}(\boldsymbol{S})}{\partial S_{ij}^{\mathbf{x}}}\right|_l\Big|$ and $w_{ij}^{\mathbf{y}} = \left|\frac{\partial\mathcal{L}(\boldsymbol{S})}{\partial S_{ij}^{\mathbf{y}}}\right|_l\Big|$ are weights at the l-th iteration; $\boldsymbol{S}$ is a similarity matrix whose element (i,j) is defined as the cosine similarity between an instance of a mini-batch and an instance stored in the self-modal feature embedding memory, as follows:

$$w_{ij}^{\mathbf{x}} = \begin{cases} \frac{1}{\tau} - \chi_{ij}^{\mathbf{x}} & \left((\mathbf{x}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{P}\right), \\ \chi_{ij}^{\mathbf{x}} & \left((\mathbf{x}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{N}\right), \end{cases} \tag{10}$$

$$\chi_{ij}^{\mathbf{x}} = \frac{1}{\tau}\frac{e^{S_{ij}^{\mathbf{x}}/\tau}}{e^{S_{i+}^{\mathbf{x}}/\tau} + \sum_{(\mathbf{x}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{N}} e^{S_{ij}^{\mathbf{x}}/\tau}}, \tag{11}$$

$$S_{ij}^{\mathbf{x}} = \frac{\mathbf{z}_i^{\mathbf{x},t\mathsf{T}}\mathbf{z}_j^{\mathbf{x},t-e}}{|\mathbf{z}_i^{\mathbf{x},t}||\mathbf{z}_j^{\mathbf{x},t-e}|}, \ \mathbf{z}_j^{\mathbf{x},t-e}\in\boldsymbol{M}^{\mathbf{x}}, \tag{12}$$

$$w_{ij}^{\mathbf{y}} = \begin{cases} \frac{1}{\tau} - \chi_{ij}^{\mathbf{y}} & \left((\mathbf{y}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{P}\right), \\ \chi_{ij}^{\mathbf{y}} & \left((\mathbf{y}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{N}\right), \end{cases} \tag{13}$$

$$\chi_{ij}^{\mathbf{y}} = \frac{1}{\tau}\frac{e^{S_{ij}^{\mathbf{y}}/\tau}}{e^{S_{i+}^{\mathbf{y}}/\tau} + \sum_{(\mathbf{y}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{N}} e^{S_{ij}^{\mathbf{y}}/\tau}}, \tag{14}$$

$$S_{ij}^{\mathbf{y}} = \frac{\mathbf{z}_i^{\mathbf{y},t\mathsf{T}}\mathbf{z}_j^{\mathbf{y},t-e}}{|\mathbf{z}_i^{\mathbf{y},t}||\mathbf{z}_j^{\mathbf{y},t-e}|}, \ \mathbf{z}_j^{\mathbf{y},t-e}\in\boldsymbol{M}^{\mathbf{y}}. \tag{15}$$

Similarly to $\mathcal{L}_{self}$, the GPW formulation of $\mathcal{L}_{cross}$ can be formulated as follows:

$$\mathcal{F}(\boldsymbol{C}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{e=0}^{E-1}\left(\sum_{(\mathbf{x}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{N}} w_{ij}^{C} C_{ij} + \sum_{(\mathbf{y}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{N}} \hat{w}_{ij}^{C}\hat{C}_{ij}\right) - \frac{1}{m}\sum_{i=1}^{m}\sum_{e=0}^{E-1}\left(\sum_{(\mathbf{x}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{P}} w_{ij}^{C} C_{ij} + \sum_{(\mathbf{y}_i^t,\mathbf{x}_j^{t-e})\in\mathcal{P}} \hat{w}_{ij}^{C}\hat{C}_{ij}\right). \tag{16}$$

Here, $w_{ij}^{C} = \left|\frac{\partial\mathcal{L}(\boldsymbol{C})}{\partial C_{ij}}\right|_l\Big|$ and $\hat{w}_{ij}^{C} = \left|\frac{\partial\mathcal{L}(\boldsymbol{C})}{\partial \hat{C}_{ij}}\right|_l\Big|$ are weights at the l-th iteration; $\boldsymbol{C}$ is a similarity matrix whose element (i,j) is defined as the cosine similarity between an instance of a mini-batch and an instance stored in the cross-modal feature embedding memory, as follows:

$$w_{ij}^{C} = \begin{cases} \frac{1}{\tau} - \chi_{ij}^{C} & \left((\mathbf{x}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{P}\right), \\ \chi_{ij}^{C} & \left((\mathbf{x}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{N}\right), \end{cases} \tag{17}$$

$$\chi_{ij}^{C} = \frac{1}{\tau}\frac{e^{C_{ij}/\tau}}{e^{C_{i+}/\tau} + \sum_{(\mathbf{x}_i^t,\mathbf{y}_j^{t-e})\in\mathcal{N}} e^{C_{ij}/\tau}}, \tag{18}$$

$$C_{ij} = \frac{\mathbf{z}_i^{\mathbf{x},t\mathsf{T}}\mathbf{z}_j^{\mathbf{y},t-e}}{|\mathbf{z}_i^{\mathbf{x},t}||\mathbf{z}_j^{\mathbf{y},t-e}|}, \ \mathbf{z}_j^{\mathbf{y},t-e}\in\boldsymbol{M}^{\mathbf{y}}, \tag{19}$$

$$\hat{w}_{ij}^{C} = \begin{cases} \frac{1}{\tau} - \hat{\chi}_{ij}^{C} & \left((\mathbf{y}_i^t, \mathbf{x}_j^{t-e}) \in \mathcal{P}\right), \\ \hat{\chi}_{ij}^{C} & \left((\mathbf{y}_i^t, \mathbf{x}_j^{t-e}) \in \mathcal{N}\right), \end{cases} \tag{20}$$

$$\hat{\chi}_{ij}^{C} = \frac{1}{\tau} \frac{e^{\hat{C}_{ij}/\tau}}{e^{\hat{C}_{i+}/\tau} + \sum_{(\mathbf{y}_i^t, \mathbf{x}_j^{t-e}) \in \mathcal{N}} e^{\hat{C}_{ij}/\tau}}, \tag{21}$$

$$\hat{C}_{ij} = \frac{\mathbf{z}_i^{\mathbf{y},t\mathsf{T}} \mathbf{z}_j^{\mathbf{x},t-e}}{|\mathbf{z}_i^{\mathbf{y},t}||\mathbf{z}_j^{\mathbf{x},t-e}|}, \ \mathbf{z}_j^{\mathbf{x},t-e} \in \boldsymbol{M}^{\mathbf{x}}. \tag{22}$$

C. Algorithm for Self- and Cross-Modal Feature Embedding Memory

Algorithm 1 presents the pseudocode of the proposed SCFEM mechanism. This pseudocode is written in PyTorch [1]. The SCFEM mechanism, which stores both the music and image feature embeddings across epochs, is based on cross-batch memory (XBM) [3]. The difference from XBM is that we initialize memory at the beginning of the training to store the embeddings in memory across epochs. Then, the SCFEM mechanism can mine at most $2E - 1 (E \geq 1)$ informative positive pairs from both the self- and cross-modal feature embedding memory for the case of a positive (*i.e.*, original) music-image pair, where E is the number of epochs to be stored in the feature memories as written in our paper. The proposed mechanism can thus be directly integrated into existing pair-based deep metric learning (DML) frameworks.

Algorithm 1 Pseudocode of SCFEM.

```
# warm-up the music encoder f_M and the
    image encoder f_I with T epochs

# initialize memory which stores embeddings
M = Memory()

# define pair-based loss function
Loss = LossFunction()

for epoch in range(T+1, EPOCH_END):
  for x, y, ids in dataloader: # x: music,
      y: image, ids: song_ids

    # embed music and image
    mus_emb = f_M(x)
    img_emb = f_I(y)

    # store the embeddings in memory
    # replace the earliest embeddings in
       memory with the embeddings at the
       current iteration when the number
       of the stored embeddings exceeds
       the size of memory
    M.update(mus_emb.detach(), img_emb.
       detach(), ids)

    # calculate similarity matrix
    sim_1 = torch.matmul(torch.t(mus_emb),
       img_emb)
    sim_2 = torch.matmul(torch.t(img_emb),
       mus_emb)

    # calculate pair-based loss
    loss = Loss(sim_1, ids) + Loss(sim_2,
       ids)

    # extract the stored embeddings from M
    mus_emb_M, img_emb_M, ids_M = M.get()

    # calculate similarity matrix using the
       embeddings in the self-modal
       feature embedding memory
    sim_s1 = torch.matmul(torch.t(mus_emb),
       mus_emb_M)
    sim_s2 = torch.matmul(torch.t(img_emb),
       img_emb_M)

    # calculate similarity matrix using the
       embeddings in the cross-modal
       feature embedding memory
    sim_c1 = torch.matmul(torch.t(mus_emb),
       img_emb_M)
    sim_c2 = torch.matmul(torch.t(img_emb),
       mus_emb_M)

    # calculate pair-based loss using both
       the self-modal feature embedding
       memory and the cross-modal feature
       embedding memory
    loss += Loss(sim_s1, ids_M)
          + Loss(sim_s2, ids_M)
          + Loss(sim_c1, ids_M)
          + Loss(sim_c2, ids_M)

    # update trainable parameters of the
       encoders
    loss.backward()
    optimizer.step()
```

References

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.

[2] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5022–5030, 2019.

[3] Xun Wang, Haozhi Zhang, Weilin Huang, and Matthew R Scott. Cross-batch memory for embedding learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6388–6397, 2020.