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 $L_{\text{self}}$ and $L_{\text{cross}}$

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

$$F(S) = \frac{1}{m} \sum_{i=1}^{m} \sum_{l=0}^{E} \left( \sum_{(x_i^l, x_j^l) \in \mathcal{P}} w^{x}_{ij} S_{ij}^{x} + \sum_{(y_i^l, y_j^l) \in \mathcal{N}} w^{y}_{ij} S_{ij}^{y} \right)$$

$$- \frac{1}{m} \sum_{i=1}^{m} \sum_{l=0}^{E} \left( \sum_{(x_i^l, x_j^l) \in \mathcal{P}} w^{x}_{ij} S_{ij}^{x} + \sum_{(y_i^l, y_j^l) \in \mathcal{P}} w^{y}_{ij} S_{ij}^{y} \right).$$

(9)

Here, $w^{x}_{ij} = \left| \frac{\partial L(S)}{\partial S_{ij}} \right|_{x}$ and $w^{y}_{ij} = \left| \frac{\partial L(S)}{\partial S_{ij}} \right|_{y}$ are weights at the $l$-th iteration; $S$ 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:

$$w^{x}_{ij} = \begin{cases} \frac{1}{\tau} - \chi^{x}_{ij} \frac{(x_i^l, x_j^l) \in \mathcal{P}}{(x_i^l, x_j^l) \in \mathcal{N}} , & \text{if} \quad x_i^l \neq x_j^l \\ \chi^{x}_{ij} \frac{(x_i^l, x_j^l) \in \mathcal{N}}{(x_i^l, x_j^l) \in \mathcal{N}} , & \text{if} \quad x_i^l = x_j^l \end{cases}$$

$$\chi^{x}_{ij} = \frac{1}{\tau} e^{S_{ij}^{x}/\tau} + \sum_{(x_i^l, x_j^l) \in \mathcal{N}} e^{S_{ij}^{x}/\tau},$$

$$S_{ij}^{x} = \frac{z_{i^l}^{x} \cdot z_{j^l}^{x}}{|z_{i^l}^{x}| \cdot |z_{j^l}^{x}|}, \quad z_{j^l}^{x} \in \mathcal{M},$$

(10)

Similarly to $L_{\text{self}}$, the GPW formulation of $L_{\text{cross}}$ can be formulated as follows:

$$F(C) = \frac{1}{m} \sum_{i=1}^{m} \sum_{l=0}^{E} \left( \sum_{(x_i^l, y_j^l) \in \mathcal{N}} w^{C}_{ij} C_{ij} + \sum_{(y_i^l, y_j^l) \in \mathcal{N}} w^{\hat{C}}_{ij} \hat{C}_{ij} \right)$$

$$- \frac{1}{m} \sum_{i=1}^{m} \sum_{l=0}^{E} \left( \sum_{(x_i^l, y_j^l) \in \mathcal{P}} w^{C}_{ij} C_{ij} + \sum_{(y_i^l, y_j^l) \in \mathcal{P}} w^{\hat{C}}_{ij} \hat{C}_{ij} \right).$$

(16)

Here, $w^{C}_{ij} = \left| \frac{\partial L(C)}{\partial C_{ij}} \right|_{x}$ and $w^{\hat{C}}_{ij} = \left| \frac{\partial L(C)}{\partial C_{ij}} \right|_{y}$ are weights at the $l$-th iteration; $C$ 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:

$$w^{C}_{ij} = \begin{cases} \frac{1}{\tau} - \chi^{C}_{ij} \frac{(x_i^l, y_j^l) \in \mathcal{P}}{(x_i^l, y_j^l) \in \mathcal{N}} , & \text{if} \quad x_i^l \neq y_j^l \\ \chi^{C}_{ij} \frac{(x_i^l, y_j^l) \in \mathcal{N}}{(x_i^l, y_j^l) \in \mathcal{N}} , & \text{if} \quad x_i^l = y_j^l \end{cases}$$

$$\chi^{C}_{ij} = \frac{1}{\tau} e^{C_{ij}/\tau} + \sum_{(y_i^l, y_j^l) \in \mathcal{N}} e^{C_{ij}/\tau},$$

$$C_{ij} = \frac{z_{i^l}^{x} \cdot z_{j^l}^{y}}{|z_{i^l}^{x}| \cdot |z_{j^l}^{y}|}, \quad z_{j^l}^{y} \in \mathcal{M},$$

(17)
\begin{align}
\hat{w}_{ij} &= \begin{cases} 
\frac{1}{\tau} - \hat{\chi}_{ij} & (y^t_i, x^t_j - e^t) \in \mathcal{P}, \\
\hat{\chi}_{ij} & (y^t_i, x^t_j - e^t) \in \mathcal{N}, 
\end{cases} 
(20)
\end{align}

\begin{align}
\hat{\chi}_{ij} &= \frac{1}{\tau} e^{C_{ij}/\tau} + \sum_{(y^t_i, x^t_j - e^t) \in \mathcal{N}} e^{C_{ij}/\tau}, 
(21)
\end{align}

\begin{align}
\hat{C}_{ij} &= \frac{z^{y^t_i T} y^t_i}{|z^{y^t_i T} y^t_i|}, z^{x^t_j - e^t}_j \in M^x. 
(22)
\end{align}

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 $2E - 1$ 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.

References


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
$M = \text{Memory}()$
# define pair-based loss function
$\text{Loss} = \text{LossFunction}()$

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_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 self-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()