Supplemental Material for *"Generalized Sum Pooling for Metric Learning"*

1. Extended Empirical Study for DML

In the following sections, we explain our empirical study in detail and provide additional experiments on effect of hyperparameters as well as evaluation with the *conventional* experimental settings.

Reproducibility

We provide full detail of our experimental setup and recapitulate the implementation details for the sake of complete transparency and reproducibility. Code is available at: GSP-DML Framework.

1.1. Conventional Evaluation

We additionally follow the relatively old-fashioned conventional procedure [28] for the evaluation of our method. We use BN-Inception [11] and ResNet50 [10] architectures as the backbones. We obtain 512D (BN-Inception and ResNet50) embeddings through linear transformation after global pooling layer. Aligned with the recent approaches [13, 35, 37, 40], we use global max pooling as well as global average pooling. The rest of the settings are disclosed in Sec. 1.6.

We evaluate our method with XBM. We provide R@1 results in Tab. 1 for the comparison with SOTA. In our evaluations, we also provide MAP@R scores in parenthesis under R@1 scores. We also provide baseline XBM evaluation in our framework. The results are mostly consistent with the ones reported in the original paper [40] except for CUB and Cars datasets. In XBM [40], the authors use proxy-based trainable memory for CUB and Cars datasets. On the other hand, we use the official implementation provided by the authors, which does not include such proxy-based extensions.

We observe that our method improves XBM and XBM+GSP reaches SOTA performance in large scale datasets. With that being said, the improvement margins are less substantial than the ones in *fair evaluation*. Such a result is expected since training is terminated by *early-stopping* which is a common practice to regularize the generalization of training [5, 18]. In *conventional evaluation*, early-stopping is achieved by monitoring the test data performance, enabling good generalization to test data. Therefore, observing less improvement in generalization with GSP is something we expect owing to generalization boost that test data based early-stopping already provides.

We also observe that in a few cases, the R@1 performance of GSP is slightly worse than the baseline. Nevertheless, once we compare the MAP@R performances, GSP consistently brings improvement with no exception. We should recapitulate that R@1 is a myopic metric to assess the quality of the embedding space geometry [24] and hence, pushing R@1 does not necessarily reflect the true order of the improvements that the methods bring.

As we observe from MAP@R comparisons in Table 2 (main paper), the methods sharing similar R@1 (*i.e.*,

Table 1: Comparison with the existing methods for the retrieval task in conventional experimental settings with BN-Inception and ResNet50 backbones where superscripts denote embedding size. Red: the best. Blue: the second best. Bold: previous SOTA. ‡*Results obtained from* [33].

(a)

Backbone →	BN-Inception-512D			
Dataset →	**CUB**	**Cars196**	**SOP**	**In-shop**
Method ↓	**R@1**	**R@1**	**R@1**	**R@1**
C1+XBM [40]	65.80	82.00	**79.50**	89.90
ProxyAnchor [13]	68.40	86.10	79.10	**91.50**
DiVA [22]	66.80	84.10	78.10	-
ProxyFewer [45]	66.60	85.50	78.00	-
Margin+S2SD [30]	**68.50**	**87.30**	79.30	-
C1+XBM	64.32 (23.59)	77.63 (21.67)	79.29 (52.59)	90.16 (61.39)
C1+XBM+GSP	64.99 (25.35)	79.07 (22.51)	**79.59** (52.70)	**90.92** (63.25)

(b)

Backbone →	ResNet50			
Dataset →	**CUB**	**Cars196**	**SOP**	**In-shop**
Method ↓	**R@1**	**R@1**	**R@1**	**R@1**
C1+XBM^{128} [40]	-	-	80.60	91.30
ProxyAnchor^{512} [13]	69.70	87.70	80.00‡	92.10‡
DiVA^{512} [22]	69.20	87.60	79.60	-
ProxyNCA++^{512} [35]	66.30	85.40	80.20	88.60
Margin+S2SD^{512} [30]	69.00	**89.50**	81.20	-
LIBC^{512} [33]	70.30	88.10	**81.40**	**92.80**
MS+Metrix^{512} [37]	**71.40**	**89.60**	81.00	92.20
PAnchor+DIML^{128} [44]	66.46 (25.58)	86.13 (28.11)	79.22 (43.04)	-
LIBC+GSP^{512}	**70.70**	88.43	**81.65**	**93.30**
C1+XBM^{512}	66.68 (25.38)	82.83 (25.34)	81.44 (55.66)	91.56 (64.00)
C1+XBM+GSP^{512}	66.63 (25.51)	82.60 (25.76)	**81.54** (55.91)	91.75 (64.43)

P@1) performances can differ in MAP@R performance relatively more significantly. In that manner, we firmly believe that comparing MAP@R performances instead of R@1 technically sounds more in showing the improvements of our method.

Finally, we also apply our method with LIBC [33] to further show wide applicability of our method. We use the official implementation of LIBC and follow their default experimental settings. The evaluations on 4 benchmarks show that GSP improve LIBC by ≈ 0.5pp R@1. To offer a complete outlook on the conventional evaluation, we have included the recall at K (R@K) scores in Table 2 as well.

Table 2: R@K performances using 512D embeddings from LIBC [33] and XBM [40] with ResNet50 backbone

Dataset→	**CUB**				**Cars196**			
Method↓	**R@1**	**R@2**	**R@4**	**R@8**	**R@1**	**R@2**	**R@4**	**R@8**
LIBC+GSP	70.70	80.72	88.18	92.64	88.43	93.03	95.78	97.69
XBM+GSP	66.63	77.43	85.26	91.02	82.60	89.04	92.67	95.62
Dataset→	**SOP**				**In-shop**			
Method↓	**R@1**	**R@10**	**R@50**	**R@100**	**R@1**	**R@10**	**R@20**	**R@40**
LIBC+GSP	81.65	91.37	94.85	96.00	93.30	98.54	98.96	99.25
XBM+GSP	81.54	91.84	95.18	96.29	91.75	97.83	98.52	99.01

1.2. Application of GSP to Other Problems

GSP is applicable to any problem and architecture with a pooling layer. We believe GSP is particularly relevant to the problem of metric learning due to the geometry it enforces. Our pooling method enhances local geometry by reducing overlap of class convex hulls and improves unseen class generalization.

In order to evaluate the applicability of GSP beyond metric learning, we applied GSP to ImageNet classification tasks using ResNetV2 [10] and EfficientNetV2 [34] models. We took the official Tensorflow Keras models and only replaced pooling layers with GSP.

Table 3: Evaluation in classification task

ImageNet	**Acc.**	**P@1**	**MAP@R**
RN50V2	75.26	69.30	41.18
+GSP	**76.53**	**71.34**	**42.58**
ENV2B3	80.03	79.23	59.98
+GSP	**82.00**	**80.80**	**62.75**

The results suggests that our method is applicable beyond metric lerning as it improves ImageNet classification accuracy for both ResNetV2 and EfficientNetV2 models. We additionally assessed the metric learning performance of the embedding vectors pre-classification. By reducing the embedding dimension to 512 through LDA, we then evaluated the resulting embedding geometry using P@1 and MAP@R metrics, and observed that GSP yields better feature geometry.

1.3. Evaluation of Other Pooling Alternatives

We evaluate 14 additional pooling alternatives on *Ciffar Collage* and CUB datasets with *contrastive* (C2) and *Proxy-NCA++* (PNCA) losses. We pick *contrastive* since it is one of the best performing sample-based loss. We pick *Proxy-NCA++* since most of the pooling methods are tailored for landmark-based image retrieval and use classification loss akin to *Proxy-NCA++*. We particularly consider *Cifar Collage* dataset since the images of different classes share a considerable amount of semantic entities, enabling us to assess the methods w.r.t. their ability to discard the nuisance information.

Compared methods. In addition to our method (GSP) and global average pooling (GAP), we consider: *i*) global max pooling (GMP), *ii*) GAP+GMP [13], *iii*) CBAM [41], *iv*) CroW [12], *v*) DeLF [27], *vi*) generalized max pooling (GeMax) [23], *vii*) generalized mean pooling (GeMean) [29], *viii*) GSoP [8], *ix*) optimal transport based aggregation (OTP) [15, 21], *x*) SOLAR [26], *xi*) trainable SMK (T-SMK) [36], *xii*) NetVLAD [1], *xiii*) WELDON [7], and *xiv*) visual transformer encoder with class token (TFM) [6]. Among those, OTP and VLAD are ensemble based methods and typically necessitate large embedding dimensions. Thus, we both experimented their 128D version -(8x16) (8 prototypes of 16D vectors) and 8192D version -(64x128) (64 prototypes of 128D vectors). We note that some of our baselines utilize attention based pooling. Notably, attention mechanism is the key part of DeLF, SOLAR, and GSoP. In fact, DeLF can be seen as equivalent to a single-layer residual-free transformer layer with a class token. To perform a more direct comparison with transformers, we conducted experiments by replacing GAP with transformer layers using a class token. We evaluated 1, 2, 4, and 8-layer transformers (TFM-#).

Setting. For CUB dataset, the experimental setting follows Sec. 1.6-*Fair evaluation* and we report MAP@R performance of the 4-model average at 128 dimensional embeddings each. For *Cifar Collage dataset*, the experimental setting follows Sec. 2.2 and we report MAP@R performance. We provide the results in Table 4.

Results. Evaluations show that our method is superior to other pooling alternatives including prototype based VLAD and OTP. Predominantly, for 128 dimensional embeddings, our method outperforms prototype based methods by large margin. In CUB dataset, the pooling methods either are inferior to or perform on par with GAP. The performance improvements of the superior methods are less than 1%, implying that our

improvements in the order of 1-2% reported in Table 2 (main paper) is substantial. Besides, the methods that mask the feature map outperform GAP by large margin in *Cifar Collage* dataset. That said, our method outperforms all the methods except for Contrastive+VLAD by large margin in *Cifar Collage* dataset, yet another evidence for better feature selection mechanism of our method. For instance in CUB dataset, DeLF and GeMean are on par with our method which has slightly better performance. Yet, our method outperforms both by large margin in *Cifar Collage* dataset.

Superior selection mechanism. Comparing to CroW, T-SMK and CBAM, our method outperforms them by large margin. Those methods are the built on feature magnitude based saliency, assuming that the backbone functions must be able to zero-out nuisance information. Yet, such a requirement is restrictive for the parameter space and annihilation of the non-discriminative information might not be feasible in some problems. We experimentally observe such a weakness of CroW, T-SMK and CBAM in *Cifar Collage* dataset where the nuisance information cannot be zeroed-out by the backbone. Our formulation do not have such a restrictive assumption and thus substantially superior to them.

Superior attention mechanism. Similarly, attention-based weighting methods, DeLF and GSoP, do not have explicit control on feature selection behavior and might result in poor models when jointly trained with the feature extractor [27], which we also observe in *Cifar Collage* experiments. On the contrary, we have explicit control on the pooling behavior with μ parameter and the behavior of our method is stable and consistent across datasets and with different loss functions. We also found that our method outperforms direct application of transformer based pooling.

Simpler and interpretable. Attention-based methods DeLF, GSoP, and SOLAR typically introduce several convolution layers to compute the feature weights. We only introduce an m-kernel 1×1 convolution layer (*i.e.*, m-many trainable prototypes) and obtain better results. We should note that our pooling operation is as simple as performing a convolution (*i.e.*, distance computation) and alternating normalization of a vector and a scalar. Additionally, we are able to incorporate a zero-shot regularization into our problem

Table 4: Evaluation of feature pooling methods on *Cifar Collage* and CUB datasets with Contrastive and ProxyNCA++ losses for DML task. Red: the best, Blue: the second best, Bold: the third best.

	128D - MAP@R			
Dataset→	**Cifar Collage**		**CUB**	
Method↓ Loss→	**Contrastive**	**ProxyNCA++**	**Contrastive**	**ProxyNCA++**
CBAM [41]	7.87	10.99	18.45	18.21
CroW [12]	10.09	11.48	20.88	20.42
DeLF [27]	11.44	**24.83**	**21.42**	20.51
GeMax [23]	7.04	7.83	18.85	17.83
GeMean [29]	10.97	10.60	**21.50**	**20.71**
GSoP [8]	11.15	17.73	20.52	15.72
OTP-(8x16) [21]	7.02	11.55	15.19	13.57
OTP-(64x128) [21]	7.57	11.79	20.88	20.48
SOLAR [26]	17.30	20.36	19.89	20.14
TFM-1 [6]	8.84	10.83	17.82	19.13
TFM-2 [6]	16.48	21.00	17.16	18.13
TFM-4 [6]	18.51	**21.56**	16.91	18.22
TFM-8 [6]	18.18	19.68	16.31	17.47
T-SMK [36]	9.21	13.15	21.01	20.23
VLAD-(8x16) [1]	**21.73**	19.68	15.19	13.08
VLAD-(64x128) [1]	**22.52**	21.15	16.67	16.53
WELDON [7]	13.81	20.38	20.67	20.31
GAP	8.09	10.68	20.58	20.63
GMP	9.53	11.25	20.66	20.33
GMP+GAP	10.01	11.85	20.88	**20.68**
GSP	**22.68**	**27.61**	**21.52**	**20.75**

naturally by using the prototype assignment weights. We can as well incorporate such a loss in DeLF which has 1x1 convolution to compute prototype similarities. However, we first need a mechanism to aggregate the per prototype similarities (*e.g.*, sum and normalization). Indeed, normalizing the similarities channel-wise and spatially summing them correspond to solving our problem with $\mu = 1$.

Other pooling methods, *i.e.*, GAP, GMP, GAP+GMP, GeMax, GeMean, WELDON, VLAD, OTP, do not build on discriminative feature selection. Thus, our method substantially outperforms those.

1.4. Computational Analysis

Forward and backward computation of proposed GSP method can be implemented using only matrix-vector products. Moreover, having closed-form matrix-inversion-free expression for the loss gradient enables memory efficient back propagation since the output of each iteration must be stored otherwise.

We perform k iterations to obtain the pooling weights and at each iteration, we only perform matrix-vector products. In this sense, the back propagation can be achieved using *automatic-differentiation*. One problem with automatic differentiation is that the computation load increases with increasing k. On the other hand, with closed-form gradient expression, we do not have such issue and in fact we have constant back propagation complexity. Granted that the closed-form expression demands exact solution of the problem (*i.e.*, $k \to \infty$), forward computation puts a little computation overhead and is memory efficient since we discard the intermediate outputs. Moreover, our initial empirical study show that our problems typically converges for $k > 50$ and we observe similar performances with $k \geqslant 25$.

The choice of k is indeed problem dependent (*i.e.*, size of the feature map and the number of prototypes). Thus, its effect on computation load should be analyzed. We study the effect of k with automatic differentiation

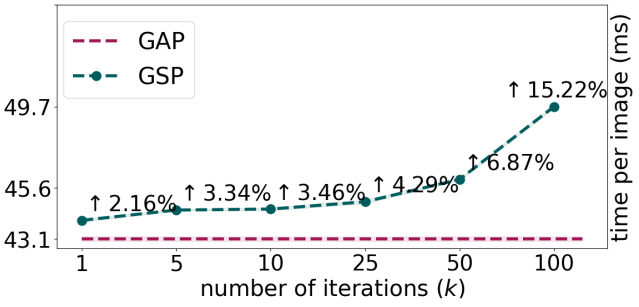


Figure 1: Computation increase (↑) in inference with GSP using k iterations

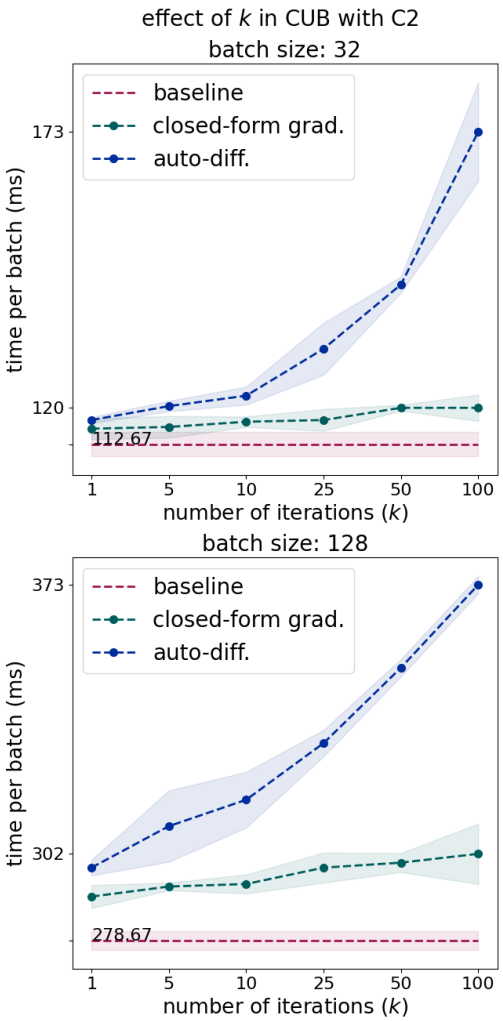


Figure 2: Comparing closed-form gradient with automatic differentiation through analyzing the effect of k on computation in CUB with C2 loss. Shaded regions represent $\mp std$.

and with our closed-form gradient expression. We provide the related plots in Fig. 2. We observe that with closed-form gradients, our method puts a little computation overhead and increasing k has marginal effect. On the contrary, with automatic differentiation, the computational complexity substantially increases.

We have further provided the inference times for various optimization steps (k) in Fig. 1. The additional computational complexity introduced by our method is minor, resulting in a less than 6% increase in the time per image (from 43.1 ms to 45.6 ms) within the typical operation interval of k (25-50). Therefore, our method remains computationally feasible for real-time processing.

1.5. Hyperparameter Selection

We first perform a screening experiment to see the effect of the parameters. We design a 2-level fractional factorial (*i.e.*, a subset of the all possible combinations) experiment.

We provide the results in Tab. 5. In our analysis, we find that *lower the better* for λ and μ. Thus, we set $\mu = 0.3$ and $\lambda = 0.1$. ε is observed to have the most effect and number of prototypes, m, seems to have no significant effect. Nevertheless, we jointly search for m and ε. To this end, we perform grid search in CUB dataset with Contrastive (C2) and Proxy NCA++ (PNCA) losses. We provide the results in Fig. 3-(a). We see that both losses have their best performance when $m = 64$. On the other hand, $\varepsilon = 5.0$ works better for C2 while $\varepsilon = 0.5$ works better for PNCA. We additionally perform a small experiment to see whether $\varepsilon = 0.5$ is the case for Proxy Anchor loss as well and observe that $\varepsilon = 0.5$ is a better choice over $\varepsilon = 5.0$. As the result of m-ε search, we set $\varepsilon = 5.0$ for non-proxy based losses and $\varepsilon = 0.5$ for proxy-based losses.

Fixing $\mu = 0.3, \lambda = 0.1, \varepsilon = 0.5($ or $5.0)$, we further experiment the effect of number of prototypes m in large datasets (*i.e.*, SOP and In-shop). We provide the corresponding performance plots in Fig. 3-(b). Supporting our initial analysis, m seemingly does not have a significant effect once it is not small (*e.g.*, $m \geqslant 64$). We observe that any choice of $m \geqslant 64$ provides performance improvement. With that being said, increasing m does not bring substantial improvement over relatively smaller values. Considering the results of the experiment, we set $m = 128$ for SOP and In-shop datasets since both C2 and PNCA losses perform slightly better with $m = 128$ than the other choices of m.

Table 5: Initial 2-level fractional factorial screening experiments for parameter tuning (conducted in CUB)

Setting				MAP@R	
m	μ	ε	λ	C2	PNCA
16	0.3	0.5	0.1	40.63	40.59
16	0.7	0.5	0.5	40.41	40.34
128	0.3	0.5	0.5	40.22	40.35
128	0.7	0.5	0.1	40.07	40.85
16	0.3	20	0.5	36.11	40.51
16	0.7	20	0.1	39.11	39.88
128	0.3	20	0.1	39.61	39.12
128	0.7	20	0.5	35.36	39.92
Baseline				39.77	39.90

1.6. Experimental Setup

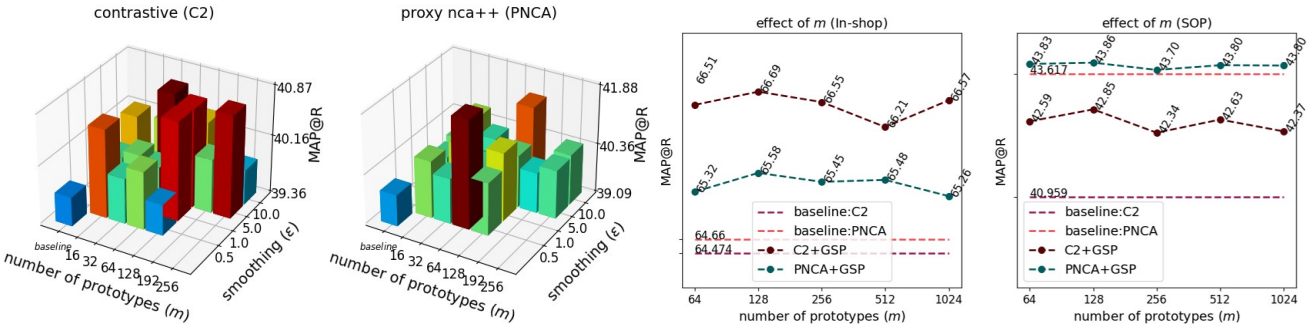
1.6.1 Datasets

We perform our experiments on 4 widely-used benchmark datasets: Stanford Online Products (SOP) [28], In-shop [19], Cars196 [16] and, CUB-200-2011 (CUB) [38].

SOP has 22,634 classes with 120,053 product images. The first 11,318 classes (59,551 images) are split for training and the other 11,316 (60,502 images) classes are used for testing.

In-shop has 7,986 classes with 72,712 images. We use 3,997 classes with 25,882 images as the training set. For the evaluation, we use 14,218 images of 3,985 classes as the query and 12,612 images of 3,985 classes as the gallery set.

Cars196 contains 196 classes with 16,185 images. The first 98 classes (8,054 images) are used for training and remaining 98 classes (8,131 images) are reserved for testing.



(a) (b)

Figure 3: Parameter search for m : number of prototoypes and ε: entropy smoothing coefficient. We fix $\mu = 0.3$ and $\lambda = 0.1$. (a) Searching $m - \varepsilon$ space in CUB dataset. (b) Effect of m once we fix $\varepsilon = 5$ for Contrastive (C2) and $\varepsilon = 0.5$ for Proxy NCA++ (PNCA).

CUB-200-2011 dataset consists of 200 classes with 11,788 images. The first 100 classes (5,864 images) are split for training, the rest of 100 classes (5,924 images) are used for testing.

Data augmentation follows [24]. During training, we resize each image so that its shorter side has length 256, then make a random crop between 40 and 256, and aspect ratio between 3/4 and 4/3. We resize the resultant image to 227x227 and apply random horizontal flip with 50% probability. During evaluation, images are resized to 256 and then center cropped to 227x227.

1.6.2 Training Splits

Fair evaluation. We split datasets into disjoint training, validation and test sets according to [24]. In particular, we partition 50%/50% for training and test, and further split training data to 4 partitions where 4 models are to be trained by exploiting 1/4 as validation while training on 3/4.

Conventional evaluation. Following relatively *old-fashioned* conventional evaluation [28], we use the whole train split of the dataset for training and we use the test split for evaluation as well as monitoring the training for early stopping.

Hyperparameter tuning. For the additional experiments related to the effect of hyperparameters, we split training set into 5 splits and train a single model on the 4/5 of the set while using 1/5 for the validation.

1.6.3 Evaluation Metrics

We consider precision at 1 (P@1) and mean average precision (MAP@R) at R where R is defined for each query[1] and is the total number of true references as the query. Among those, MAP@R performance metric is shown to better reflect the geometry of the embedding space and to be less noisy as the evaluation metric [24]. Thus, we use MAP@R to monitor training in our experiments except for conventional evaluation setting where we monitor P@1.

P@1: Find the nearest reference to the query. The score for that query is 1 if the reference is of the same class, 0 otherwise. Average over all queries gives P@1 metric.

P@R: For a query, i, find R_i nearest references to the query and let r_i be the number of true references in those R_i-neighbourhood. The score for that query is $\text{P@R}_i = {}^{r_i}/_{R_i}$. Average over all queries gives P@R metric, *i.e.*, $\text{P@R} = \frac{1}{n} \sum_{i \in [n]} \text{P@R}_i$, where n is the number of queries.

[1]A query is an image for which similar images are to be retrieved, and the references are the images in the searchable database.

MAP@R: For a query, i, we define $\text{MAP@R}_i := \frac{1}{R_i} \sum_{i \in [R_i]} P(i)$, where $P(i) = \text{P@R}_i$ if i^{th} retrieval is correct or 0 otherwise. Average over all queries gives MAP@R metric, *i.e.*, $\text{MAP@R} = \frac{1}{n} \sum_{i \in [n]} \text{MAP@R}_i$, where n is the number of queries.

1.6.4 Training Procedure

Fair evaluation. We use Adam [14] optimizer with constant 10^{-5} learning rate, 10^{-4} weight decay, and default moment parameters, β_1=.9 and β_2=.99. We use batch size of 32 (4 samples per class). We evaluate validation MAP@R for every 100 steps of training in CUB and Cars196, for 1000 steps in SOP and In-shop. We stop training if no improvement is observed for 15 steps in CUB and Cars196, and 10 steps in SOP and In-shop. We recover the parameters with the best validation performance. Following [24], we train 4 models for each 3/4 partition of the train set. Each model is trained 3 times. Hence, we have $3^4 = 81$ many realizations of 4-model collections. We present the average performance as well as the standard deviation (*std*) of such 81 models' evaluations.

Conventional evaluation. We use Adam [14] optimizer with default moment parameters, β_1=.9 and β_2=.99. Following recent works [13], we use *reduce on plateau* learning rate scheduler with patience 4. The initial learning rate is 10^{-5} for CUB, and 10^{-4} for Cars, SOP and In-shop. We use 10^{-4} weight decay for BNInception backbone and $4\,10^{-4}$ wight decay for ResNet50 backbone. We use batch size of 128 (4 samples per class) for BNInception backbone and 112 (4 samples per class) for ResNet backbone (following [31]). We evaluate validation P@1 for every 25 steps of training in CUB and Cars196, for 250 steps in SOP and In-shop. We stop training if no improvement is observed for 15 steps in CUB and Cars196, and 10 steps in SOP and In-shop. We recover the parameters with the best validation performance. We repeat each experiment 3 times and report the best result. For the evaluations on LIBC framework [33], we follow their experimental setting.

Hyperparameter tuning. We use Adam [14] optimizer with constant 10^{-5} learning rate, 10^{-4} weight decay, and default moment parameters, β_1=.9 and β_2=.99. We use batch size of 32 (4 samples per class). We evaluate validation MAP@R for every 100 steps of training in CUB and Cars196, for 1000 steps in SOP and In-shop. We stop training if no improvement is observed for 10 steps in CUB and Cars196, and 7 steps in SOP and In-shop. We recover the parameters with the best validation performance. We train a single model on the 4/5

of the training set while using 1/5 for the validation. We repeat each experiment 3 times and report the averaged result.

1.6.5 Embedding vectors

Fair evaluation. Embedding dimension is fixed to 128. During training and evaluation, the embedding vectors are $\ell 2$ normalized. We follow the evaluation method proposed in [24] and produce two results: *i*) Average performance (128 dimensional) of 4-fold models and *ii*) Ensemble performance (concatenated 512 dimensional) of 4-fold models where the embedding vector is obtained by concatenated 128D vectors of the individual models before retrieval.

Conventional evaluation. Embedding dimension is 512 in both BNInception and ResNet50 experiments.

Hyperparameter tuning. Embedding dimension is fixed to 128.

1.6.6 Baselines with GSP

We evaluate our method with *C1+XBM+GSP*: Cross-batch memory (XBM) [40] with original Contrastive loss (C1) [9], *C2+GSP*: Contrastive loss with positive margin [42], *MS+GSP*: Multi-similarity (MS) loss [39], *Triplet+GSP*: Triplet loss [32], *PNCA+GSP*: ProxyNCA++ loss [35], *PAnchor+GSP*: ProxyAnchor loss [13].

1.6.7 Hyperparameters

For the hyperparameter selection, we exploit the work [24] that performed parameter search via Bayesian optimization on variety of losses. We further experiment the suggested parameters from the original papers and official implementations. We pick the best performing parameters. We perform no further parameter tuning for the baseline methods' parameters when applied with our method to purely examine the effectiveness of our method.

C1: We adopted XBM's official implementation for fair comparison. We use 0.5 margin for all datasets.

C2: C2 has two parameters, (m^+, m^-): positive margin, m^+, and negative margin. We set (m^+, m^-) to $(0, 0.3841)$, $(0.2652, 0.5409)$, $(0.2858, 0.5130)$, $(0.2858, 0.5130)$ for CUB, Cars196, In-shop and SOP, respectively.

Triplet: We set its margin to 0.0961, 0.1190, 0.0451, 0.0451 for CUB, Cars196, In-shop and SOP, respectively.

MS: We set its parameters (α, β, λ) to $(2, 40, 0.5)$, $(14.35, 75.83, 0.66)$, $(8.49, 57.38, 0.41)$, $(2, 40, 0.5)$ for CUB, Cars196, In-shop and SOP, respectively.

ProxyAnchor: We set its two paremeters (δ, α) to $(0.1, 32)$ for all datasets. We use 1 sample per class in batch setting (*i.e.*, 32 classes with 1 samples per batch), we perform 1 epoch warm-up training of the embedding layer, and we apply learning rate multiplier of 100 for the proxies during training. For SOP dataset, we use $5\,10^{-6}$ learning rate.

ProxyNCA++: We set its temperature parameter to 0.11 for all datasets. We use 1 sample per class in batch setting (*i.e.*, 32 classes with 1 samples per batch), we perform 1 epoch warm-up training of the embedding layer, and we apply learning rate multiplier of 100 for the proxies.

XBM: We evaluate XBM with C1 since in the original paper, contrastive loss is reported to be the best performing baseline with XBM. We set the memory size of XBM according to the dataset. For CUB and Cars196, we use memory size of 25 batches. For In-shop, we use 400 batches and for SOP we use 1400 batches. We perform 1K steps of training with the baseline loss prior to integrate XBM loss in order to ensure XBM's *slow drift* assumption.

GSP: For the hyperparameters of our method, we perform parameters search, details of which are provided in Sec. 1.5. We use 64-many prototypes in CUB and Cars, and 128-many prototypes in SOP and In-shop. We use ε=0.5 for proxy-based losses and ε=5.0 for non-proxy losses. For the rest, we set μ=0.3, ϵ=0.05, and we iterate until k=100 in Proposition 4.1. For zero-shot prediction loss coefficient (*i.e.*, (1-λ)$\mathcal{L}_{DML} + \lambda\mathcal{L}_{ZS}$), we set λ=0.1.

2. Details of the Other Empirical Work

2.1. Synthetic Study

We design a synthetic empirical study to evaluate GSP in a fully controlled manner. We consider 16-class problem such that classes are defined over trainable tokens. In this setting, tokens correspond to semantic entities but we choose to give a specific working to emphasize that they are trained as part of the learning. Each class is defined with 4 distinct tokens and there are also 4 background tokens shared by all classes. For example, a *"car"* class would have tokens like *"tire"* and *"window"* as well as background tokens of *"tree"* and *"road"*.

We sample class representations from both class specific and background tokens according to a mixing ratio $\tilde{\mu} \sim \mathcal{N}(0.5, 0.1)$. We sample a total of 50 tokens and such a 50-many feature collection will correspond to a training sample (*i.e.*, we are mimicking CNN's output with trainable tokens). For instance, given class tokens for class-c, $\nu^{(c)} = \{\nu_1^{(c)}, \nu_2^{(c)}, \nu_3^{(c)}, \nu_4^{(c)}\}$

and shared tokens, $\nu^{(b)} = \{\nu_1^{(b)}, \nu_2^{(b)}, \nu_3^{(b)}, \nu_4^{(b)}\}$; we first sample $\mu = 0.4$ and then sample 20 tokens from $\nu^{(c)}$ with replacement, and 30 tokens from $\nu^{(b)}$, forming a feature collection for a class-c, *i.e.*, $f^{(c)} = \{\nu_3^{(c)}, \nu_1^{(c)}, \nu_1^{(c)}, \nu_3^{(c)}, \ldots, \nu_4^{(b)}, \nu_3^{(b)}, \nu_4^{(b)}, \nu_1^{(b)}, \ldots\}$ We then obtain global representations using GAP and GSP.

We do not apply $\ell 2$ normalization on the global representations. We also constrain the range of the token vectors to be in between $[-0.3, 0.3]$ to bound the magnitude of the learned vectors. We use default Adam optimizer with 10^{-4} learning rate and perform early stopping with 30 epoch patience by monitoring MAP@R. In each batch, we use 4 samples from 16 classes.

2.2. Cifar Collage

We consider the 20 *super-classes* of Cifar100 dataset [17] where each has 5 sub-classes. For each super-class, we split the sub-classes for train (2), validation (1), and test (2). We consider 4 super-classes as the shared classes and compose 4x4-stitched collage images for the rest 16 classes. In particular, we sample an image from a class and then sample 3 images from shared classes. We illustrate a sample formation process in Fig. 4.

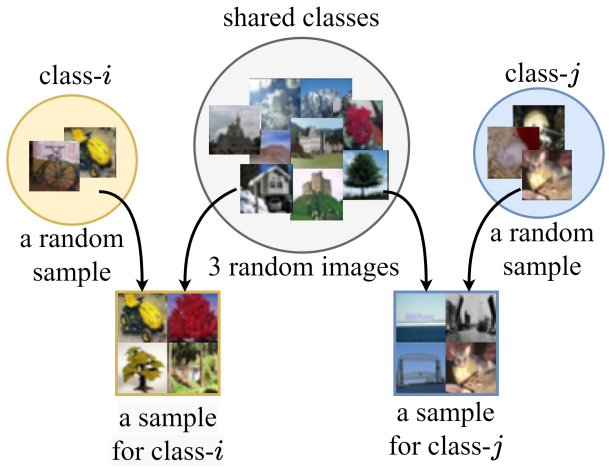


Figure 4: Sample generation for Cifar Collage dataset

We should note that the classes exploited in training, validation and test are disjoint. For instance, if a *tree* class is used as a shared class in training, then that *tree* class does not exist in validation or test set as a shared feature. Namely, in our problem setting, both the background and the foreground classes are disjoint across training, validation and test sets. Such a setting is useful to analyze zero-shot transfer capability of our method.

We use ResNet20 (*i.e.*, 3 stages, 3 blocks) backbone pretrained on Cifar100 classification task. We use $\ell 2$ normalization on global representations. We use default Adam optimizer with initial 0.001 learning rate. We use *reduce on plateau* with 0.5 decay factor and 5 epochs patience. For GSP, we set $m = 128, \mu = 0.2, \varepsilon = 10, \lambda = 0.5$. We use 4 samples from 16 classes in a batch.

2.3. Evaluation of Zero-shot Prediction Loss

We train on Cifar10 [17] dataset with 8 prototypes using ProxyNCA++ [35] (PNCA) loss with and without $\mathcal{L}_{ZS}$. We then use test set to compute attribute histograms for each class. Namely, we aggregate the marginal transport plans of each sample in a class to obtain the histogram. Similarly, for each class, we compute the mean embedding vector (*i.e.*, we average embedding vectors of the samples of a class). Our aim is to fit a linear predictor to map attribute vectors to the mean embeddings.

To quantify the zero-shot prediction performance, we randomly split the classes into half and apply cross-batch zero-shot prediction. Specifically, we fit a linear predictor using 5 classes and then use that transformation to map the other 5 classes to their mean embeddings. We then compute pairwise distance between the predicted means and the true means. We then evaluate the nearest neighbour classification performance. We use both $\ell 2$ distance and cosine distance while computing the pairwise distances. We repeat the experiment 1000 times with different class splits.

References

[1] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307, 2016.

[2] Heinz H Bauschke and Adrian S Lewis. Dykstras algorithm with bregman projections: A convergence proof. *Optimization*, 48(4):409–427, 2000.

[3] Lev M Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217, 1967.

[4] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. *Advances in neural information processing systems*, 32, 2019.

[5] M Dong, X Yang, R Zhu, Y Wang, and J Xue. Generalization bound of gradient descent for non-convex metric learning. Neural Information Processing Systems Foundation, 2020.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner,

Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[7] Thibaut Durand, Nicolas Thome, and Matthieu Cord. Weldon: Weakly supervised learning of deep convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4743–4752, 2016.

[8] Zilin Gao, Jiangtao Xie, Qilong Wang, and Peihua Li. Global second-order pooling convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3024–3033, 2019.

[9] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[12] Yannis Kalantidis, Clayton Mellina, and Simon Osindero. Cross-dimensional weighting for aggregated deep convolutional features. In *European conference on computer vision*, pages 685–701. Springer, 2016.

[13] Sungyeon Kim, Dongwon Kim, Minsu Cho, and Suha Kwak. Proxy anchor loss for deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3238–3247, 2020.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Soheil Kolouri, Navid Naderializadeh, Gustavo K Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. In *International Conference on Learning Representations*, 2020.

[16] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014.

[17] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[18] Yunwen Lei, Mingrui Liu, and Yiming Ying. Generalization guarantee of sgd for pairwise learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[19] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1096–1104, 2016.

[20] Tzon-Tzer Lu and Sheng-Hua Shiou. Inverses of 2×2 block matrices. *Computers & Mathematics with Applications*, 43(1-2):119–129, 2002.

[21] Grégoire Mialon, Dexiong Chen, Alexandre d'Aspremont, and Julien Mairal. A trainable optimal transport embedding for feature aggregation and its relationship to attention. In *ICLR 2021-The Ninth International Conference on Learning Representations*, 2021.

[22] Timo Milbich, Karsten Roth, Homanga Bharadhwaj, Samarth Sinha, Yoshua Bengio, Björn Ommer, and Joseph Paul Cohen. Diva: Diverse visual feature aggregation for deep metric learning. In *European Conference on Computer Vision*, pages 590–607. Springer, 2020.

[23] Naila Murray, Hervé Jégou, Florent Perronnin, and Andrew Zisserman. Interferences in match kernels. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1797–1810, 2016.

[24] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *European Conference on Computer Vision*, pages 681–699. Springer, 2020.

[25] Navid Naderializadeh, Joseph F Comer, Reed Andrews, Heiko Hoffmann, and Soheil Kolouri. Pooling by sliced-wasserstein embedding. *Advances in Neural Information Processing Systems*, 34:3389–3400, 2021.

[26] Tony Ng, Vassileios Balntas, Yurun Tian, and Krystian Mikolajczyk. Solar: second-order loss and attention for image retrieval. In *European conference on computer vision*, pages 253–270. Springer, 2020.

[27] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[28] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016.

[29] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Fine-tuning cnn image retrieval with no human annotation. *IEEE transactions on pattern analysis and machine intelligence*, 41(7):1655–1668, 2018.

[30] Karsten Roth, Timo Milbich, Bjorn Ommer, Joseph Paul Cohen, and Marzyeh Ghassemi. S2sd: Simultaneous similarity-based self-distillation for deep metric learning. In *ICML 2021: 38th International Conference on Machine Learning*, pages 9095–9106, 2021.

[31] Karsten Roth, Timo Milbich, Samarth Sinha, Prateek Gupta, Bjorn Ommer, and Joseph Paul Cohen. Revisiting training strategies and generalization performance in deep metric learning. In *International Conference on Machine Learning*, pages 8242–8252. PMLR, 2020.

[32] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE confer-*

ence on computer vision and pattern recognition, pages 815–823, 2015.

[33] Jenny Seidenschwarz, Ismail Elezi, and Laura Leal-Taixé. Learning intra-batch connections for deep metric learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9410–9421. PMLR, 2021.

[34] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021.

[35] Eu Wern Teh, Terrance DeVries, and Graham W Taylor. Proxynca++: Revisiting and revitalizing proxy neighborhood component analysis. In *European Conference on Computer Vision (ECCV)*. Springer, 2020.

[36] Giorgos Tolias, Tomas Jenicek, and Ondřej Chum. Learning and aggregating deep local descriptors for instance-level recognition. In *European Conference on Computer Vision*, pages 460–477. Springer, 2020.

[37] Shashanka Venkataramanan, Bill Psomas, Ewa Kijak, laurent amsaleg, Konstantinos Karantzalos, and Yannis Avrithis. It takes two to tango: Mixup for deep metric learning. In *International Conference on Learning Representations*, 2022.

[38] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[39] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[40] 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*, pages 6388–6397, 2020.

[41] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[42] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2840–2848, 2017.

[43] Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. *Advances in Neural Information Processing Systems*, 33:20520–20531, 2020.

[44] Wenliang Zhao, Yongming Rao, Ziyi Wang, Jiwen Lu, and Jie Zhou. Towards interpretable deep metric learning with structural matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9887–9896, 2021.

[45] Yuehua Zhu, Muli Yang, Cheng Deng, and Wei Liu. Fewer is more: A deep graph metric learning perspective using fewer proxies. *Advances in Neural Information Processing Systems*, 33:17792–17803, 2020.

A. Appendix

A.1. Proof for Theorem 4.1

Proof. ρ^* is obtained as the solution of the following optimal transport problem:

$$\rho^*, \pi^* = \underset{\substack{\rho,\pi \geqslant 0 \\ \rho_j + \Sigma_i \pi_{ij} = 1/n \\ \Sigma_{ij} \pi_{ij} = \mu}}{\arg\min} \textstyle\sum_{ij} c_{ij} \pi_{ij}.$$

Given solutions (ρ^*, π^*), for μ=1, from the 3^{rd} constraint, we have $\Sigma_{ij}\pi^*_{ij}$=1. Then, using the 2^{nd} constraint, we write:

$$\textstyle\sum_j \rho^*_j + \sum_j \sum_i \pi^*_{ij} = \sum_j \frac{1}{n}$$

where $j \in [n]$ for n-many convolutional features. Hence, we have $\sum_j \rho^* = 0$ which implies ρ^*=0 owing to non-negativity constraint. Finally, pooling weights becomes $p_i = \frac{1/n - \cancel{\rho^*_i}}{\underset{=1}{\mu}} = 1/n$.

□

A.2. Proof for Proposition 4.2

Before starting our proof, we first derive an iterative approach for the solution of (P2). We then prove that the iterations in Proposition 4.2 can be used to obtain the solution.

We can write (P2) equivalently as:

$$\begin{aligned}\rho^{(\varepsilon)}, \pi^{(\varepsilon)} = \underset{\substack{\rho,\pi \geqslant 0 \\ \rho_j + \Sigma_i \pi_{ij} = 1/n \\ \Sigma_{ij} \pi_{ij} = \mu}}{\arg\min} & \textstyle\sum_{ij} c_{ij}\pi_{ij} + \frac{1}{\varepsilon}\big(\sum_{ij} \pi_{ij} \log \pi_{ij} + \sum_j \rho_j \log \rho_j\big) \\ & + \textstyle\sum_j 0\rho_j - \sum_{ij} \pi_{ij} - \sum_j \rho_j + \sum_{ij} \mathrm{e}^{-\varepsilon c_{ij}} + \sum_j \mathrm{e}^{-\varepsilon 0}\end{aligned}$$

Rearranging the terms we get:

$$\rho^{(\varepsilon)}, \pi^{(\varepsilon)} = \underset{\substack{\rho,\pi \geqslant 0 \\ \rho_j + \Sigma_i \pi_{ij} = 1/n \\ \Sigma_{ij} \pi_{ij} = \mu}}{\arg\min} \textstyle\sum_{ij} \pi_{ij} \log \frac{\pi_{ij}}{\mathrm{e}^{-\varepsilon c_{ij}}} + \sum_j \rho_j \log \frac{\rho_j}{\mathrm{e}^{-\varepsilon 0}} - \sum_{ij} \pi_{ij} - \sum_j \rho_j + \sum_{ij} \mathrm{e}^{-\varepsilon c_{ij}} + \sum_j \mathrm{e}^{-\varepsilon 0}$$

which is generalized *Kullback–Leibler divergence* (KLD) between (ρ, π) and $(\exp(\text{-}\varepsilon 0), \exp(\text{-}\varepsilon c))$. Hence, we reformulate the problem as a KLD prjoection onto a convex set, which can be solved by *Bregman Projections* (*i.e.*, alternating projections onto constraint sets) [2, 3]. Defining $\mathcal{C}_1 \coloneqq \{(\rho, \pi) \mid \rho_j + \sum_{ij} \pi_{ij} = 1/n \ \forall j\}$ and $\mathcal{C}_2 \coloneqq \{(\rho, \pi) \mid \sum_{ij} \pi_{ij} = \mu\}$, and omitting constants, we can write the problem as:

$$\rho^{(\varepsilon)}, \pi^{(\varepsilon)} = \underset{\substack{\rho,\pi \geqslant 0 \\ (\rho,\pi) \in \mathcal{C}_1 \cap \mathcal{C}_2}}{\arg\min} \ \textstyle\sum_{ij} \pi_{ij}(\log \frac{\pi_{ij}}{\mathrm{e}^{-\varepsilon c_{ij}}} \text{ - } 1) + \sum_j \rho_j(\log \frac{\rho_j}{\mathrm{e}^{-\varepsilon 0}} \text{ - } 1) \tag{P2$'$}$$

Given, $(\rho^{(k)}, \pi^{(k)}))$, at iteration k, KLD projection onto $\mathcal{C}_1$, *i.e.*, $(\rho^{(k+1)}, \pi^{(k+1)}) \coloneqq \mathcal{P}^{KL}_{\mathcal{C}_1}(\rho^{(k)}, \pi^{(k)})$, reads:

$$\begin{aligned}\rho^{(k+1)}_j &= 1/n(\rho^{(k)}_j + \textstyle\sum_i \pi^{(k)}_{ij})^{\text{-}1}\rho^{(k)}_j\,, \\ \pi^{(k+1)} &= 1/n(\rho^{(k)}_j + \textstyle\sum_i \pi^{(k)}_{ij})^{\text{-}1}\pi^{(k)}_{ij}\end{aligned}$$

where the results follow from *method of Lagrange multipliers*. Similarly, for $\mathcal{P}^{KL}_{\mathcal{C}_2}(\rho^{(k)}, \pi^{(k)})$, we have:

$$\rho^{(k+1)} = \rho^{(k)}\,, \quad \pi^{(k+1)} = \frac{\mu}{\sum_{ij} \pi^{(k)}_{ij}} \pi^{(k)}\,.$$

Given initialization, $(\rho^{(0)}, \pi^{(0)}) = (\mathbf{1}_n, \exp(\text{-}\varepsilon c))$, we obtain the pairs $(\rho^{(k)}, \pi^{(k)})$ for $k = 0, 1, 2, \ldots$ as:

$$\begin{aligned}\rho^{(k+1)} = 1/n(\rho^{(k)} + \pi^{(k)\intercal}\mathbf{1}_m)^{\text{-}1} \odot \rho^{(k)}\,, \quad \pi^{(k+1)} = \mu(\mathbf{1}^\intercal_m \hat{\pi} \mathbf{1}_n)^{\text{-}1}\hat{\pi} \\ \text{where} \quad \hat{\pi} = \pi^{(k)} Diag\big(1/n(\rho^{(k)} + \pi^{(k)\intercal}\mathbf{1}_m)^{\text{-}1}\big)\end{aligned} \tag{A.1}$$

Proof. We will prove by induction. From Proposition 4.2, we have

$$\rho^{(k+1)} = {}^1\!/_n \, (1 + t^{(k)} \exp(\text{-}\varepsilon c)^\intercal \mathbf{1}_m)^{\text{-}1}, \;\; t^{(k+1)} = \mu \, (\mathbf{1}_m^\intercal \exp(\text{-}\varepsilon c)\rho^{(k+1)})^{\text{-}1}$$

and $\pi^{(k)} = t^{(k)} \exp(\text{-}\varepsilon c) Diag(\rho^{(k)})$. It is easy to show that the expressions hold for the pair $(\rho^{(1)}, \pi^{(1)})$. Now, assuming that the expressions also holds for arbitrary $(\rho^{(k')}, \pi^{(k')})$. We have

$$\rho^{(k'+1)} = {}^1\!/_n (\rho^{(k')} + \pi^{(k')\intercal} \mathbf{1}_m)^{\text{-}1} \odot \rho^{(k')}$$

Replacing $\pi^{(k')} = t^{(k')} \exp(\text{-}\varepsilon c) Diag(\rho^{(k')})$ we get:

$$\rho^{(k'+1)} = {}^1\!/_n (\rho^{(k')} + Diag(\rho^{(k')}) t^{(k')} \exp(\text{-}\varepsilon c)^\intercal \mathbf{1}_m)^{\text{-}1} \odot \rho^{(k')}$$

where $\rho^{(k')}$ terms cancel out, resulting in:

$$\rho^{(k'+1)} = {}^1\!/_n (1 + t^{(k')} \exp(\text{-}\varepsilon c)^\intercal \mathbf{1}_m)^{\text{-}1}.$$

Similarly, we express $\hat{\pi}$ as:

$$\hat{\pi} = t^{(k')} \exp(\text{-}\varepsilon c) Diag(\rho^{k'}) Diag\Big({}^1\!/_n \big(\rho^{(k')} + Diag(\rho^{(k')}) t^{(k')} \exp(\text{-}\varepsilon c)^\intercal \mathbf{1}_m\big)^{\text{-}1}\Big)$$

again $\rho^{(k')}$ terms cancel out, resulting in:

$$\hat{\pi} = t^{(k')} \exp(\text{-}\varepsilon c) Diag({}^1\!/_n (1 + t^{(k')} \exp(\text{-}\varepsilon c)^\intercal \mathbf{1}_m)^{\text{-}1}) = t^{(k')} \exp(\text{-}\varepsilon c) Diag(\rho^{(k'+1)}).$$

Hence, $\pi^{(k'+1)}$ becomes:

$$\begin{aligned}
\pi^{(k'+1)} &= \mu (\mathbf{1}_m^\intercal t^{(k')} \exp(\text{-}\varepsilon c) Diag(\rho^{(k'+1)}) \mathbf{1}_n)^{\text{-}1} t^{(k')} \exp(\text{-}\varepsilon c) Diag(\rho^{(k'+1)}) \\
&= \tfrac{1}{t^{(k')}} \underbrace{\mu (\mathbf{1}_m^\intercal \exp(\text{-}\varepsilon c) \rho^{(k'+1)})^{\text{-}1}}_{= t^{(k'+1)}} t^{(k')} \exp(\text{-}\varepsilon c) Diag(\rho^{(k'+1)}) \\
&= t^{(k'+1)} \exp(\text{-}\varepsilon c) Diag(\rho^{(k'+1)}),
\end{aligned}$$

meaning that the expressions also hold for the pair $(\rho^{(k'+1)}, \pi^{(k'+1)})$. □

A.3. Proof for Proposition 4.3

Proof. We start our proof by expressing (P2′) in a compact form as:

$$x^{(\varepsilon)} = \underset{\substack{x \geqslant 0 \\ Ax = b}}{\arg\min} \, \bar{c}^\intercal x + \tfrac{1}{\varepsilon} x^\intercal (\log x - \mathbf{1}_{(m+1)n})$$

where x denotes the vector formed by stacking ρ and the row vectors of π, $\bar{c}$ denotes the vector formed by stacking n-dimensional zero vector and the row vectors of c, and A and b are such that $Ax = b$ imposes transport constraints as:

$$A = \begin{bmatrix} I_{n\times n} & \overbrace{I_{n\times n} \quad \cdots \quad I_{n\times n}}^{m} \\ \mathbf{0}_n^\intercal & \mathbf{1}_{m\,n}^\intercal \end{bmatrix}, \quad b = [{}^1\!/_n \mathbf{1}_n^\intercal \quad \mu]^\intercal$$

From *Lagrangian dual*, we have:

$$x^{(\varepsilon)} = \exp(\text{-}\varepsilon(\bar{c} + A^\intercal \lambda^*))$$

where λ^* is the optimal dual Lagrangian satisfying:

$$A \exp(\text{-}\varepsilon(\bar{c} + A^\intercal \lambda^*)) = b$$

Defining $[\frac{\partial x}{\partial c}]_{ij} \coloneqq \frac{\partial x_j}{\partial c_i}$, we have;

$$\tfrac{\partial x^{(\varepsilon)}}{\partial c} = -\varepsilon \bar{I} (I + \tfrac{\partial \lambda^*}{\partial \bar{c}} A) Diag(x^{(\varepsilon)})$$

where $\bar{I} \coloneqq [\mathbf{0}_{(mn)\times n} \quad I_{(mn)\times((m+1)n)}]$. Similarly, for the dual variable, we have:

$$-\varepsilon(I + \frac{\partial \lambda^*}{\partial \bar{c}} A) Diag(x^{(\varepsilon)}) A^\intercal = 0 \Rightarrow \frac{\partial \lambda^*}{\partial \bar{c}} = \text{-}Diag(x^{(\varepsilon)}) A^\intercal (ADiag(x^{(\varepsilon)}) A^\intercal)^{\text{-}1}.$$

Putting back the expression for $\frac{\partial \lambda^*}{\partial \bar{c}}$ in $\frac{\partial x^{(\varepsilon)}}{\partial c}$, we obtain:

$$\frac{\partial x^{(\varepsilon)}}{\partial c} = -\varepsilon \bar{I} \big(Diag(x^{(\varepsilon)}) - Diag(x^{(\varepsilon)}) A^\intercal (ADiag(x^{(\varepsilon)}) A^\intercal)^{\text{-}1} ADiag(x^{(\varepsilon)}) \big),$$

which includes $(m{+}1)$ by n matrix inversion, $H \coloneqq ADiag(x^{(\varepsilon)}) A^\intercal$. We now show that $H^{\text{-}1}$ can be obtained without explicit matrix inversion.

H can be expressed as:

$$H = \begin{bmatrix} {}^1\!/_n I & {}^1\!/_n - \rho \\ {}^1\!/_n - \rho^\intercal & \mu \end{bmatrix}$$

H is Hermitian and positive definite. Using block matrix inversion formula for such matrices (Corrolary 4.1 of [20]), we obtain the inverse as;

$$H^{\text{-}1} = \begin{bmatrix} nI + k^{\text{-}1} \hat{\rho} \hat{\rho}^\intercal & -k^{\text{-}1} \hat{\rho} \\ -k^{\text{-}1} \hat{\rho}^\intercal & k^{\text{-}1} \end{bmatrix}$$

where $k = 1 - \mu - n\rho^{(\varepsilon)\intercal} \rho^{(\varepsilon)}$ and $\hat{\rho} = 1 - n\rho^{(\varepsilon)}$.

Given $\frac{\partial \mathcal{L}}{\partial x^{(\varepsilon)}}$, *i.e.*, $(\frac{\partial \mathcal{L}}{\partial \rho^{(\varepsilon)}}, \frac{\partial \mathcal{L}}{\partial \pi^{(\varepsilon)}})$, the rest of the proof to obtain $\frac{\partial \mathcal{L}}{\partial c}$ follows from right multiplying the Jacobian, *i.e.*, $\frac{\partial \mathcal{L}}{\partial c} = \frac{\partial x^{(\varepsilon)}}{\partial c} \frac{\partial \mathcal{L}}{\partial x^{(\varepsilon)}}$ and rearranging the terms. □

Table 6: Comparing our pooling method with OT-based pooling

Item↓ Method→	Ensemble of SWD Monge Maps [25]	OT Monge Map [15, 21]	Ours (GSP)
optimization problem	$s^\dagger$-many 1D OT	$\underset{\substack{\pi \geq 0 \\ \Sigma_i \pi_{ij} = 1/n \\ \Sigma_j \pi_{ij} = 1/m}}{\text{argmin}} \Sigma_{ij} c_{ij} \pi_{ij}$	$\underset{\substack{\rho, \pi \geq 0 \\ \rho_j + \Sigma_i \pi_{ij} = 1/n \\ \Sigma_{ij} \pi_{ij} = \mu}}{\text{argmin}} \Sigma_{ij} c_{ij} \pi_{ij}$
image representation	$[g_1 \mid g_2 \mid \cdots \mid g_s]^\dagger$	$\sqrt{m}[f_1 \mid f_2 \mid \cdots \mid f_n]\pi^\intercal$	$\Sigma_i \frac{1 - n\rho_i}{n\mu} f_i$
dimension	$m \times s$	$m \times d$	d
feature selection	✗	✗	✓
gradient computation	auto-diff	auto-diff	closed form expression
matrix-inverse-free gradient	✓	✗	✓

† s: number of slices, g_i: projection of $\{f_j\}_j$ to slice-i with sorted j according to Monge Map.

B. Optimal Transport Based Operators

In this section, we briefly discuss optimal transport (OT) based aggregation and selection operators. We provide their formulations to show how our formulation differs from them.

B.1. Feature Aggregation

Given a cost map $c_{ij} = \|\omega_i - f_j\|_2$ which is an m (number of prototypes) by n (number of features) matrix representing the closeness of prototypes ω_i and features f_j, [21] consider the following OT problem:

$$\pi^* = \underset{\substack{\pi \geqslant 0 \\ \Sigma_i \pi_{ij} = 1/n \\ \Sigma_j \pi_{ij} = 1/m}}{\arg\min} \sum_{ij} c_{ij} \pi_{ij} \tag{P4}$$

and defines their aggregated feature as:

$$g = \sqrt{m}[f_1 \mid f_2 \mid \cdots \mid f_n]\pi^{\intercal} \ . \tag{B.1}$$

Namely, g is an ensemble of m vectors each of which is the weighted aggregation of $\{f_i\}_{i\in[n]}$ with the weights proportional to the assignment weights to the corresponding prototype. The same aggregation scheme is also discovered within the context of linear *Wasserstein* embeddings via *Monge maps* and is shown to be a barycentric projection of the feature set with the transport plan to approximate *Monge map* [15]. Similar to them, ensemble of Monge maps corresponding to sliced-Wasserstein distances (SWD) are further employed in set aggregation [25]. In such ensemble representations, there is no feature selection mechanism and thus, all the features are somehow included in the image representation.

For instance, if we further sum those m vectors of g in Eq. (B.1) to obtain a single global representation, we end up with global average pooling: $g^{\intercal}\mathbf{1}_m = \sqrt{m}[f_1 \mid f_2 \mid \cdots \mid f_n]\pi^{\intercal}\mathbf{1}_m = \sqrt{m}/n[f_1 \mid f_2 \mid \cdots \mid f_n]\mathbf{1}_n = \sqrt{m}/n \Sigma_i f_i$.

Briefly, those optimal transport based set aggregators, [15, 21, 25] map a set of features to another set of features without discarding any and do not provide a natural way to aggregate the class-discriminative subset of the features. Such representation schemes are useful for structural matching. Albeit enabling $\ell 2$ metric as a similarity measure for the sets, their ensemble based representation results in very high dimensional embeddings. On the contrary, our formulation effectively enables learning to select discriminative features and maps a set of features to a single feature that is of the same dimension and is distilled from nuisance information. We summarize the comparison of our pooling method with the optimal transport based counterparts in Table 6.

B.2. Top-k Selection

Given n-many scalars as $x = [x_i]_{i\in[n]}$ and m-many scalars as $y = [y_i]_{i\in[m]}$ with y is in an increasing family, *i.e.*, $y_1 < y_2 < \ldots$, [43] considers the following OT:

$$\pi^* = \underset{\substack{\pi \geqslant 0 \\ \Sigma_i \pi_{ij} = q_j \\ \Sigma_j \pi_{ij} = p_i}}{\arg\min} \sum_{ij} c_{ij} \pi_{ij} \tag{P5}$$

where $c_{ij} = |y_i - x_j|$ and p is m-dimensional probability simplex, *i.e.*, $p \in \{p \in \mathbb{R}^m_{\leq 0} \mid \Sigma_i p_i = 1\}$. Then, top-$k$ selection is formulated with the setting $q = 1/n\mathbf{1}_n$, $y = [0, 1]$ and $p = [\frac{k}{n} \ \frac{n-k}{n}]^{\intercal}$. Similarly, sorted top-$k$ selection is formulated with the setting $y = [k]$ and $p = [\frac{1}{n} \cdots \frac{1}{n} \ \frac{n-k}{n}]^{\intercal}$. Solving the same problem (P5), [4] formulate top-k ranking by letting q and p be arbitrary probability simplex and y be in an increasing family.

Such top-k formulations are suitable for selecting/ranking scalars. In our problem, the aim is to select a subset of features that are closest to the prototypes which are representatives for the discriminative information. Namely, we have a problem of subset selection from set-to-set distances. If we had our problem in the form of set-to-vector, then we would be able to formulate the problem using (P5). However, there is no natural extension of the methods in [4, 43] to our problem. Therefore, we rigorously develop an OT based formulation to express a discriminative subset selection operation analytically in a differentiable form.

Our formulation in (P1) differs from the typical optimal transport problem exploited in (P5). In optimal transport, one matches two distributions and transports all the mass from one to the other. Differently, we transport μ portion of the uniformly distributed masses to the prototypes that have no restriction on their mass distribution. In our formulation, we have a portion constraint instead of a target distribution constraint, and we use an additional decision variable, ρ, accounting for residual masses. If we do not have ρ and set $\mu = 1$, then the problem becomes a specific case of an optimal transport barycenter problem with 1 distribution.

Our problem can be expressed in a compact form by absorbing ρ into π with zero costs associated in the formulation, as in the proof (Appendix A.3). We choose to explicitly define ρ in the problem (P1) to show its role. We believe its residual mass role is more understandable this way. The benefits of our formulation include that we can perform feature selection with matrix inversion free Jacobian and we can change the role of the prototypes as background representatives simply by using ρ to weight the features instead of $1/n$-ρ in Eq. (4.1). Our specific formulation further allows us to tailor a zero-shot regularization loss built on the learned prototypes within our pooling layer.

C. Implementations with Pseudo Codes

Algorithm 1 Deep Metric Learning Loss with GSP and ZSR

input: $(X, Y) = (\{x_i\}, \{y_i\})_{i \in b}$ // a batch of image-label pairs
$F \leftarrow \text{Backbone}(X)$ // a CNN backbone such as *BN-Inception, ResNet*
$(X_p, Z) \leftarrow \{\text{GSP}(f)\}_{f \in F}$ // get pooled features and attribute predictions, see Algorithm 2
$\mathcal{L}_{ZSR} \leftarrow \text{ZSR}(Z, Y)$ // compute ZSR loss, see Algorithm 4
$\mathcal{L}_{DML} \leftarrow \text{LossDML}(X_p, Y)$ // a DML loss such as *contrastive, triplet, XBM, LIBC, ...*
$\mathcal{L} \leftarrow (1\text{-}\lambda)\mathcal{L}_{DML} + \lambda\mathcal{L}_{ZSR}$ // we set λ=0.1
return $\mathcal{L}$

Algorithm 2 GSP(f)

trainable parameters: $\omega = \{\omega_i\}_{i \in [m]}$ // m-many prototypes

input: $f = \{f_i\}_{i \in [n]}$ // feature map, $n = w\,h$ (*i.e.* width×height)
$\bar{\omega}_i \leftarrow {}^{\omega_i}/_{\max\{1, \|\omega_i\|_2\}}\ \forall i \in [m]$
$\bar{f}_j \leftarrow {}^{f_j}/_{\max\{1, \|f_j\|_2\}}\ \forall j \in [n]$
$c_{ij} \leftarrow \|\bar{\omega}_i - \bar{f}_j\|_2$ // cost map, $c = \{c_{ij}\}_{(i,j) \in [m] \times [n]}$
$\rho, \pi \leftarrow \text{WeightTransport}(c)$ // see Algorithm 3
$f \leftarrow \frac{1\text{-}n\rho}{\mu} \odot f$ // re-weight features, $\odot$: element-wise multiplication
$x_p \leftarrow (\frac{1}{n} \sum_{i \in [n]} f_i^p)^{1/p}$ // pooled feature, GSP for p=1, GeMean+GSP for p>1
$z_i \leftarrow \frac{1}{\mu} \sum_{j \in [n]} \pi_{ij}\ \forall i \in [m]$ // *attribute* predictions, $z = \{z_i\}_{i \in [m]}$
return x_p, z

Algorithm 3 WeightTransport(c)

hyperparameters: μ : transport ratio, ε : entropy regularization weight, k : number of iterations

forward: gets cost map, c, returns residual weights, ρ, and transport plan π

input: $c = \{c_{ij}\}_{(i,j) \in [m] \times [n]}$ // cost map of m-many prototypes and n-many features
$\kappa \leftarrow \exp(\text{-}\varepsilon c)$, $t \leftarrow 1$ // exp is element-wise
repeat k times
$\rho \leftarrow {}^1/_n (1 + t\,\kappa^\intercal \mathbf{1}_m)^{-1}$ // $A^\intercal \mathbf{1}_m$: sum A along rows, $(\cdot)^{-1}$ is element-wise
$t \leftarrow \mu(\mathbf{1}_m^\intercal \kappa\,\rho)^{-1}$
return ρ, $t\,\kappa\,Diag(\rho)$ // $\pi \leftarrow t\,\kappa\,Diag(\rho)$

backward: gets the solution (ρ, π) and the gradients $(\frac{\partial \mathcal{L}}{\partial \rho}, \frac{\partial \mathcal{L}}{\partial \pi})$, returns $\frac{\partial \mathcal{L}}{\partial c}$

input: ρ, π, $\frac{\partial \mathcal{L}}{\partial \rho}$, $\frac{\partial \mathcal{L}}{\partial \pi}$ // results of forward pass and the loss gradient w.r.t. them
$q \leftarrow \rho \odot \frac{\partial \mathcal{L}}{\partial \rho} + (\pi \odot \frac{\partial \mathcal{L}}{\partial \pi})^\intercal \mathbf{1}_m$ // $A^\intercal \mathbf{1}_m$: sum A along rows, $\odot$: element-wise multiplication
$\eta \leftarrow (\rho \odot \frac{\partial \mathcal{L}}{\partial \rho})^\intercal \mathbf{1}_n$ - $n\,q^\intercal \rho$
$\frac{\partial \mathcal{L}}{\partial c} \leftarrow \text{-}\varepsilon\Big(\pi \odot \frac{\partial \mathcal{L}}{\partial \pi} - n\pi Diag\big(q - \frac{\eta}{1\text{-}\mu\text{-}n\rho^\intercal \rho}\big)\rho\Big)$
return $\frac{\partial \mathcal{L}}{\partial c}$

Algorithm 4 ZSR(Z, Y)

trainable parameters: $\Upsilon = \{\upsilon_i\}_{i\in[\#\text{classes}]}$ // a semantic embedding vector for each class label

input: $Z=\{z_i\}_{i\in b}$, $Y=\{y_i\}_{i\in b}$ // a batch, b, of attribute prediction vectors, z_i, and their labels, y_i

$(b_1, b_2) \leftarrow$ split b into two class-disjoint halves s.t. $\{y_i\}_{i\in b_1} \cap \{y_i\}_{i\in b_2} = \emptyset$

$\Upsilon_k \leftarrow [\upsilon_{y_i}]_{i\in b_k}$ for k=1, 2 // label embedding matrix for batch-k, *i.e.* prediction targets

$Z_k \leftarrow [z_i]_{i\in b_k}$ for k=1, 2 // attribute prediction matrix for batch-k, *i.e.* prediction inputs

$A_k \leftarrow \Upsilon_k(Z_k^\intercal Z_k + \epsilon I)^{-1} Z_k^\intercal$ for k=1, 2 // fit label embedding predictor for batch-k, ϵ=0.05

$\hat{\Upsilon}_1 \leftarrow A_2 Z_1$, $\hat{\Upsilon}_2 \leftarrow A_1 Z_2$ // use predictor for b_k to predict the label embeddings of $b_{k'}$

$\hat{\Upsilon} \leftarrow [\hat{\Upsilon}_1 \mid \hat{\Upsilon}_2]$ // concatenate predictions

$S \leftarrow \text{SoftMax}(\hat{\Upsilon}^\intercal \Upsilon)$ // similarity scores between predictions and label embeddings

$\mathcal{L}_{ZSR} \leftarrow \text{CrossEntropy}(S, Y)$

return $\mathcal{L}_{ZSR}$