Supplemental Material for *"Deep Metric Learning with Chance Constraints"*

Yeti Z. Gürbüz[1] Oğul Can[1, 2] A. Aydın Alatan[3]
[1]MetaDialog [2]Cerebrate AI [3]OGAM and METU

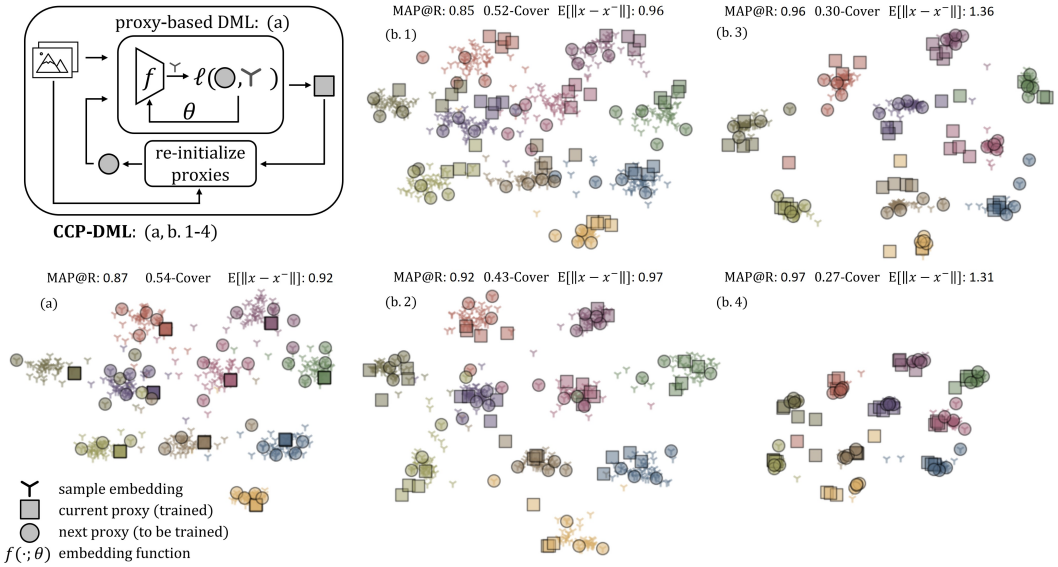

Figure 1. Illustration of our method (CCP) and the geometry of the embedding space in MNIST dataset: Boxes represent the converged proxies, while circles represent the next proxies resulting from K-Center. (a) In proxy-based DML (before our method), proxies coalesce into one. (b) With CCP (through iterations 1-4), diverse proxies are obtained, resulting in a reduced covering radius.

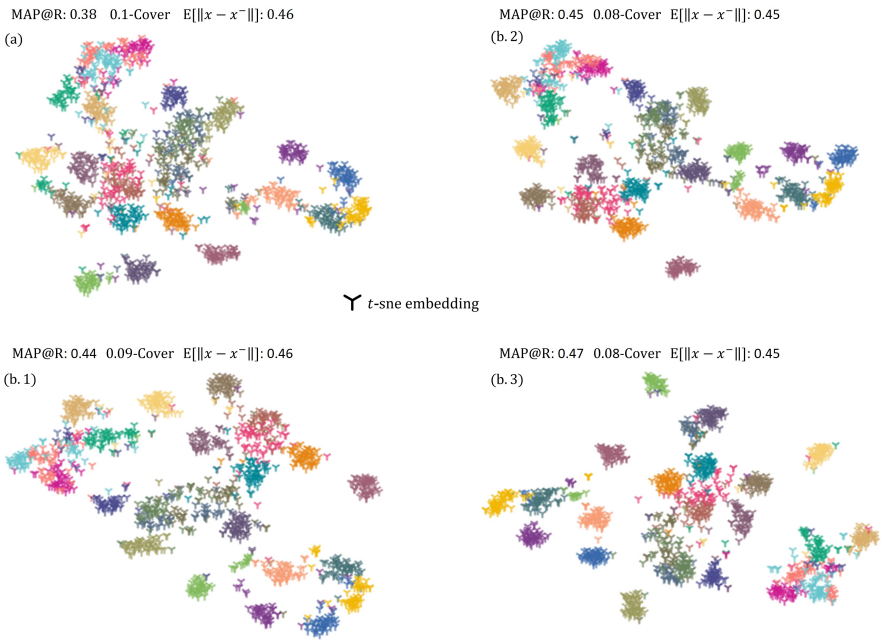

Figure 2. The geometry of the embedding space before, (a), and after, (b), our method (through iterations 1-3), relating how the generalization efforts in training domain transfer to the geometry of test domain on CUB dataset with C2-CCP. We use 2D TSNE embeddings of the validation data in the visualization, in which we report MAP@R, average covering radius and average inter-class pairwise distances.

Table 1. Evaluation on SOP and In-shop for the retrieval task. Red: the overall best. Bold: the loss specific best.

	SOP						In-shop					
	512D			128D			512D			128D		
Method	**P@1**	**P@R**	**MAP@R**	**P@1**	**P@R**	**MAP@R**	**P@1**	**P@R**	**MAP@R**	**P@1**	**P@R**	**MAP@R**
C1 [4]	68.84	43.28	40.25	64.96	39.68	36.54	80.12	53.11	50.15	76.08	49.61	46.51
C1-XBM-L [18]	78.68	54.66	51.82	75.37	49.95	47.39	88.39	61.33	58.64	**85.75**	58.13	55.37
C1-CCP-L	**79.53**	**55.11**	**52.73**	**76.24**	**50.07**	**48.07**	**88.52**	**62.54**	**59.67**	85.50	**58.51**	**55.56**
C2 [19]	74.87	49.88	46.94	71.15	45.77	42.66	86.32	62.36	59.42	83.04	58.27	55.13
C2-XBM-L [18]	76.66	51.91	49.04	73.47	48.18	45.15	87.66	63.50	60.64	84.58	59.78	56.75
C2-CCP-L	**78.95**	**55.01**	**52.19**	**75.92**	**51.14**	**48.18**	**88.52**	**63.94**	**61.07**	**86.11**	**60.16**	**57.24**
MS [17]	72.74	47.07	44.10	68.96	43.25	40.18	88.37	63.53	60.65	85.39	59.65	56.61
MS-CCP-L	**78.96**	**54.71**	**51.85**	**75.80**	**50.48**	**47.97**	**90.24**	**66.31**	**63.59**	**87.10**	**61.74**	**59.15**
Triplet [13]	75.40	50.13	47.03	70.41	44.32	41.03	86.71	63.81	60.60	82.58	58.74	55.25
Triplet-CCP-L	**77.09**	**52.42**	**49.33**	**72.21**	**46.38**	**43.11**	**89.44**	**67.23**	**64.28**	**86.00**	**62.24**	**59.04**
ProxyAnchor [5]	77.10	51.95	49.01	73.86	47.94	44.89	88.08	60.91	58.09	85.87	57.80	54.95
ProxyNCA++ [14]	76.07	51.17	48.20	72.89	47.47	44.44	87.33	60.33	57.48	84.79	57.33	54.42
SoftTriple-S [11]	78.48	53.68	50.77	74.66	48.79	45.75	88.37	62.56	59.56	85.71	58.74	55.68
HPL-PA [21]	76.97	51.97	49.07	73.84	48.10	45.11	-	-	-	-	-	-

Table 2. Evaluation on SOP and In-shop for the retrieval task. Red: the overall best. Bold: the loss specific best.

	CUB						Cars196					
	512D			128D			512D			128D		
Method	**P@1**	**P@R**	**MAP@R**	**P@1**	**P@R**	**MAP@R**	**P@1**	**P@R**	**MAP@R**	**P@1**	**P@R**	**MAP@R**
C1 [4]	63.67	33.77	23.08	56.21	29.65	19.06	77.75	33.69	23.50	64.17	26.50	16.13
C1-XBM-L [18]	65.40	35.57	24.87	57.57	30.42	19.84	83.68	37.74	27.93	**72.13**	28.55	18.83
C1-CCP-L	**68.11**	**37.85**	**27.11**	**59.56**	**32.06**	**21.27**	**83.76**	**37.78**	**28.32**	72.05	**28.74**	**18.96**
C2 [19]	67.49	37.18	26.47	59.73	31.86	21.01	81.04	34.97	24.73	69.17	27.70	17.22
C2-XBM-L [18]	68.62	37.53	26.83	60.18	32.25	21.41	82.40	36.07	25.99	70.01	28.49	18.01
C2-CCP-L	**69.73**	**38.69**	**28.02**	**62.39**	**33.49**	**22.67**	**82.89**	**36.27**	**26.27**	**72.16**	**28.98**	**18.52**
MS [17]	64.65	34.84	24.15	57.24	30.29	19.64	80.88	36.45	26.23	69.27	28.93	18.25
MS-CCP-L	**68.84**	**38.19**	**27.44**	**61.10**	**33.23**	**22.40**	**86.26**	**38.97**	**29.14**	**74.97**	**30.44**	**19.85**
Triplet [13]	64.01	34.55	23.43	55.51	29.38	18.51	78.44	33.83	23.11	64.57	26.52	15.68
Triplet-CCP-L	**65.36**	**35.42**	**24.53**	**56.65**	**30.17**	**19.31**	**81.84**	**35.61**	**25.21**	**68.75**	**28.21**	**17.43**
ProxyAnchor [5]	68.43	37.36	26.53	60.61	32.36	21.48	85.29	37.53	27.73	75.79	29.91	19.56
ProxyNCA++ [14]	65.48	35.60	24.85	58.49	31.73	20.96	82.87	36.56	26.34	72.45	29.91	19.32
SoftTriple-L [11]	68.12	36.98	26.02	57.94	30.63	19.86	84.90	37.69	27.80	73.16	29.60	19.18
HPL-PA [21]	68.25	37.57	26.72	61.31	32.81	21.90	**86.84**	38.36	28.67	**76.12**	30.13	19.83

1. Extended Empirical Study for DML

1.1. Fair (MLRC) Evaluation on DML Benchmarks

We follow the procedures proposed in [9] to provide fair and unbiased evaluation of our method. We provide the full experimental setup details in § 2.1. The evaluation results that are summarized in Fig. 3 in the main paper are tabulated in Tables 1 and 2, which demonstrate the clear superiority of our method, particularly when considering the Mean Average Precision at R (MAP@R) metric.

We should recapitulate that Precision at 1 (P@1), or Recall at 1 (R@1), is a myopic metric for assessing the quality of the embedding space geometry [9]. Therefore, solely improving P@1 does not necessarily reflect the true order of improvements brought by different methods. As observed in Tables 1 and 2, methods with similar P@1 (R@1) performances can exhibit more significant differences in MAP@R. Consequently, we firmly believe that comparing MAP@R, rather than P@1 alone, technically provides a more accurate representation of the improvements achieved by our method.

1.2. Further Ablations

Effect of CCP in test domain. Our *proof of the concept* study in MNIST dataset (Fig. 1) empirically shows the implications of our formulation in training domain. It is important to show how such efforts in the training domain are reflected in the test domain since metric learning is expected to be generalized to new classes. We further provide the visualization of the validation data in CUB dataset in Fig. 2. We compute covering radii for 1 to n sample case in k-Center. Namely, we take k samples with minimum cover for $k \in [n]$ where n is the number of samples per class. We then take the average of these radii to compute a representative metric for the covering radius. We observe that solving single proxy-based DML results in relatively poor generalization in the test domain. On the contrary, solving the problem as the set intersection problem with alternating projections improves the embedding geometry (reduced radius without decreased inter-class pairwise distances).

Hyperparameter search. Our CCP framework introduces 3 additional hyperparameters to a typical DML problem, which are λ: regularization weight for the projection objective, $\#proxy$: number of proxies per class, and b: pool size for proxy selection in k-Center method. Among those, the selection of $\#proxy$ and b is rather resource dependent and even the setting ($\#proxy = 1$, $b = 1$) brings performance improvements as we empirically show in Figs. 4 and 5. We expect such a behaviour since CCP mechanism is able to increase the number of proxies inherently. On the other hand, we must analyze the behaviour of CCP with respect to λ in order to suggest a proper operation range. To this end, we perform Bayesian search on the λ-$\#proxy$ space by fixing $b = 16$ to see the joint effect of two in CUB with C2-CCP. We provide the results in Fig. 3. We observe that absence of λ degrades the performance. Similarly, large values of λ causes over-regularization. We obtain interval of $[10^{-1}, 10^{-5}]$ that works well for λ.

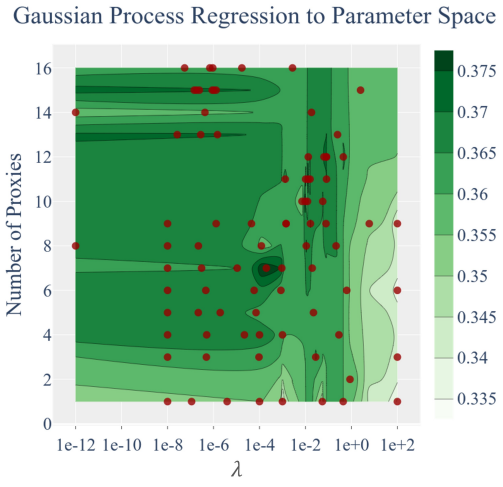

Figure 3. Bayesian search on λ-$\#proxy$ space

For the number of proxies, we observe increasing the proxy per class improves performance. On the other hand, the increase saturates as it can also be observed from Fig. 5. As the result of Bayesian parameter search, we take λ=$2 \cdot 10^{-4}$ and $\#proxy$=8 with pool size b=12 in our evaluations against other methods for CUB and Cars. For SOP and In-shop, we reduce $\#proxy$=4 and b=7 owing to relatively less number of samples per class in the dataset.

Effect of batch size. Batch size plays important role in DML methods to perform well. Therefore, we analyze the robustness to the batch size especially for the cases where increasing the batch size is prohibitive. We train baseline contrastive loss and CCP contrastive loss for the batch sizes of 16, 32, 64 and 128. The training setup is the same as in the state-of-the-art comparison (§ 2.1). In each batch we use 4 samples per class. We provide the results in Fig. 4. We observe that baseline contrastive loss has increasing performance as the batch size increases whereas our method's performance with small batch size is on par with the large batch size. Thus, CPP has reduced batch size complexity.

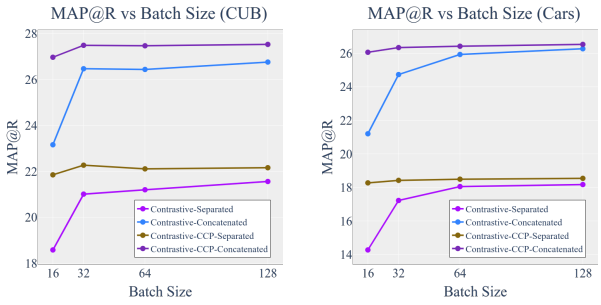

Figure 4. Analysis of batch size dependence of the performance on CUB (left) and Cars (right) dataset with C2-CCP.

Computational analysis. Our method outlined in Algorithm 1 puts marginal computation and memory overhead on top of the baseline approaches.

For the computation, we have proxy initialization and weight update steps at the beginning of the each problem instance. In overall, in our system with RTX 2080 Ti GPU and i7 CPU, that additional computation adds on the average 5-10 ms per step (batch update). In particular, for batch size of 32, we typically have rate of 105 ms/batch with Contrastive-CCP whereas vanilla has 97 ms/batch rate. In In-shop and SOP dataset, we have the same rates however for CCP, we have 200 to 400 ms computation overhead due to sampling for proxy initialization. We do not have such overhead in Cars and CUB owing to the much less number of classes. With that being said, we have such 400 ms overhead in In-shop and SOP only at the begin-

ning of new problem instance, which has no significant effect in long run thanks to rather infrequent happening of proxy re-initialization. Due to alternating problems, our method takes more steps to converge than their baseline counterparts. We provide the optimization steps per proxy-based problem instance for several losses in Fig. 4 from which relative convergence can be compared owing to each problem instance being a proxy-based method. Nevertheless, we also provide Tab. 3 to compare the convergence of the methods for SOP and In-shop datasets. The reported numbers are the rounded averages of the 4 models. We observe 10%-35% increase in the optimization steps for the pairwise losses.

Table 3. Total steps of training in SOP and In-shop

Method	SOP	In-shop
C2	62K	114K
C2-XBM	81K	93K
C2-CCP	69K	127K
MS	67K	98K
MS-CCP	91K	131K
Triplet	93K	73K
Triplet-CCP	124K	115K
ProxyAnchor	54K	87K
ProxyNCA++	88K	103K
SoftTriple	48K	82K

For the memory, we store the weighs of the previously converged model in the memory as well as the variables for proxies. For the model, approximately 40-45 mb additional GPU memory is used and for the proxies 16.6 mb and 5.9 mb memory is used in SOP and In-shop dataset (75 kb in CUB and Cars).

In summary, CCP brings $\approx$ 8% increase in back-propagation computation time and only $\approx$ 60 MB increase in memory for the largest model. CCP takes 10% - 35% more steps to converge than their baseline counterparts due to alternating problems. On the other hand, our method with small batch size performs on par with the large batch size thanks to alternating proxies (Fig. 4). To this manner, marginal increase in computation is seemingly a fair trade-off in improving the performance along with robustness to batch size.

2. Empirical Study Details

In the following sections, we outline the complete details of our experimental setup, enabling easy reproducibility.

We use our own framework implemented in Tensorflow [1] library in the experiments.

Fairness in evaluation. Independent works [2, 9, 12] reveal that conventional training and evaluation procedures in DML may fail to properly assess the true order of performance that the methods bring. The consensus for unbiased comparability is evaluation of the methods with their best version under the same experimental settings unless the compared methods demand any particular architecture or experimental setup. Our empirical study is completely aligned with the literature's claims for unbiased evaluation.

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: CCP-DML Framework

2.1. Experimental Setup

Datasets. We perform our experiments on 4 widely-used benchmark datasets: Stanford Online Products (SOP) [10], In-shop [8], Cars196 [7] and, CUB-200-2011 (CUB) [16].

SOP [10] 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.

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 (5,924 images) is used for testing.

Training Splits. We split datasets into disjoint training, validation and test sets according to [9]. 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 $^3/_4$. For the ablation studies, we split training set into 3 splits instead of 1 and train a single model on the $^2/_3$ of the set while using $^1/_3$ for the validation.

Data augmentation follows [9]. 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.

Evaluation metrics. We consider precision at 1 (P@1), precision (P@R) 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 [9]. Thus, we use MAP@R to monitor training.

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

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

P@R$_i = {}^{r_i}/_{R_i}$. Average over all queries gives P@R metric, *i.e.*, P@R $= \frac{1}{n} \sum_{i \in [n]}$ P@R$_i$, where n is the number of queries.

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

Training procedure. For the optimization procedure, we use Adam [6] optimizer for mini-batch gradient descent with a mini-batch size of 32 (4 samples per class), 10^{-5} learning rate, 10^{-4} weight decay, default moment parameters, β_1=.9 and β_2=.99. We evaluate validation MAP@R 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 60 steps and recover the parameters with the best validation performance. Following [9], we train 4 models for each $3/4$ partition of the train set. For the ablation studies, we train a single model on the $2/3$ partition.

Embedding vectors. Embedding dimension is fixed to 128. During training and evaluation, the embedding vectors are L2 normalized using the transformation proposed in Section 4.4. We follow the evaluation method proposed in [9] 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.

Losses with CCP. We evaluate our method with *C1-CCP*: Contrastive loss [3], *C2-CCP*: Contrastive loss with positive margin [19], *MS-CCP*: Multi-similarity (MS) loss [17], *Triplet-CCP*: Triplet loss [13]. We should note that ProxyAnchor [5] is indeed proxy-based MS loss except for missing a margin term. Similarly, ProxyNCA [14] is $\log \Sigma \exp$-approximation of proxy-based Triplet with hard negative mining and for single proxy case SoftTriple [11] is equivalent to ProxyNCA. Therefore, our experiments cover wide range of the DML losses.

Compared methods. We compare our method against proxy-based SoftTriple [11], ProxyAnchor [5] and ProxyNCA++ [14] methods as well as XBM [18].

Fairness. We note that like the compared methods (*i.e.*, loss functions, proxy-based methods), our method's improvement claims do not demand any particular architecture or experimental setup. Therefore, to evaluate the improvements purely coming from the proposed ideas, we implemented the best version of the compared methods in our framework and evaluate on the same architecture and experimental settings. In this manner, we stick to BN-Inception with global average pooling architecture to directly compare our method with the benchmarked losses in [9]. To eliminate any framework related performance differences, we re-implemented the methods within our framework and produce the consistent results with [9]. Our experimental setting is fair and unbiased since:

i) The compared methods are either invented loss functions or proxy-based approaches, which do not demand a particular setting to show the effectiveness of the proposed ideas.

ii) We use the same experimental setting for each method (*e.g.*, image size, architecture, embedding size, batch size, data augmentation).

iii) We implement and re-evaluate all the compared methods on our framework (*i.e.*, train and evaluate).

iv) We reproduce consistent results reported in [9] to eliminate any framework related performance bias.

v) We use the same train and test split as the conventional methods, but we do not exploit test data during training. We use $1/4$ split of train data for the validation set.

Hyperparameters. For the hyperparameter selection, we exploit the recent work [9] that has 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 loss parameters when applied to 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.

ProxyNCA++: We set its temperature parameter to 0.1 for all datasets. We use 1 sample per class in batch setting (*i.e.*, 32 classes 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.

SoftTriple: SoftTriple has 4 parameters λ, γ, τ, and δ. We set $(\lambda, \gamma, \tau, \delta)$ to $(20, 0.1, 0.2, 0.01)$, $(17.69, 19.18, 0.0669, 0.3588)$, $(20, 0.1, 0.2, 0.01)$, $(100, 47.9, 0.2, 0.3145)$ for CUB, Cars196, In-shop and SOP, respectively. 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.

XBM: We evaluate XBM with C1 and C2 since in the original paper, contrastive loss is reported to be the best performing baseline with XBM. We set the memory size of XBM to the total number of proxies (*i.e.*, $proxy\_per\_class \times \#classes$) to compare the methodology by disentangling the effect of proxy number. With that being said, we also evaluate XBM with the memory sizes suggested in the original paper. In this manner we use two memory sizes for XBM for each dataset: (S, L) where S and L denote the number of batches in the memory. For CUB and Cars196, CCP uses 1(8) proxies per class for $S(L)$. Thus, we set (S, L) to $(3, 25)$ for CUB and Cars196. For In-shop and SOP, CCP uses 1(4) proxies per class for $S(L)$. Thus, we set (S, L) to $(100, 400), (400, 1400)$ for In-shop and SOP, respectively. We perform 1K steps of training with the baseline loss prior to integrate XBM loss in order to ensure *slow drift* assumption.

CCP: For the hyperparameters of our method, we use 8 proxies per class and λ=2 10^{-4} for CUB and Cars datasets, as the result of the parameter search; and use pool size, b=12, for greedy k-Center method. We select pool size based on our empirical studies on the effect pool size and number of proxies. Due to computation limitations, we use 4 proxy per class, λ=2 10^{-4} and b= 7 for SOP and In-shop dataset. We perform no warm-up or do not use learning rate multiplier for the proxies.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 4

[2] Istvan Fehervari, Avinash Ravichandran, and Srikar Appalaraju. Unbiased evaluation of deep metric learning algorithms. *arXiv preprint arXiv:1911.12528*, 2019. 4

[3] 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. 5

[4] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Discriminative deep metric learning for face verification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1875–1882, 2014. 2

[5] 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. 2, 5

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[7] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014. 4

[8] 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. 4

[9] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *European Conference on Computer Vision*, pages 681–699. Springer, 2020. 2, 4, 5

[10] 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*, 2016. 4

[11] Qi Qian, Lei Shang, Baigui Sun, Juhua Hu, Hao Li, and Rong Jin. Softtriple loss: Deep metric learning without triplet sampling. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 2, 5

[12] 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. 4

[13] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 2, 5

[14] 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. 2, 5

[15] Aad W Van Der Vaart, Aad van der Vaart, Adrianus Willem van der Vaart, and Jon Wellner. *Weak convergence and empirical processes: with applications to statistics*. Springer Science & Business Media, 2013. 9

[16] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset, 2011. 4

[17] 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. 2, 5

[18] 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. 2, 5

[19] 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. 2, 5

[20] Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012. 8

[21] Zhibo Yang, Muhammet Bastan, Xinliang Zhu, Douglas Gray, and Dimitris Samaras. Hierarchical proxy-based loss for deep metric learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1859–1868, 2022. 2

A. Appendix

A.1. Proof for Lemma 4.1

Lemma 4.1. Generalized contrastive loss defined as $\ell(z_i, z_j; \theta) \coloneqq (y_{ij}(\|x_i - x_j\|_{f_\theta} - \beta) + \alpha)_+$ is $\sqrt{2}\omega^L$-Lipschitz in x_i and x_j for all y_i, y_j, θ for the embedding function $f(\cdot; \theta)$ being L-layer CNN (with ReLU, max-pool, average-pool) with a fully connected layer at the end, where ω is the maximum sum of the input weights per neuron.

Proof. We first show that $f(x; \theta)$ is Lipschitz continuous.

We consider $x \in \mathbb{R}^d$ as an input to a layer and $\hat{x} \in \mathbb{R}^{d'}$ as the corresponding output. We express i^{th} component of $\hat{x}$ as $\hat{x}_i = \sum_j w_{i,j} x_{s_i(j)}$ where $s_i = \{s_i(j) \in [d]\}$ is the set of components contributing to $\hat{x}_i$ and $w_{i,j} \in \theta$ is the layer weights. For instance, for a fully connected layer $s_i(j) = j$; for a 3x3 convolutional layer, s_i corresponds to 3x3 window of depth $\#channels$ centered at i. We now consider two inputs x, x' and their outputs $\hat{x}, \hat{x}'$. We write:

$$\frac{\|\hat{x} - \hat{x}'\|_2^2}{\|x - x'\|_2^2} = \frac{\sum_{i \in [d']} |\hat{x}_i - \hat{x}'_i|^2}{\|x - x'\|_2^2} = \frac{\sum_{i \in [d']} |\sum_j w_{i,j} x_{s_i(j)} - \sum_j w_{i,j} x'_{s_i(j)}|^2}{\|x - x'\|_2^2}$$
$$\leqslant \frac{\sum_{i \in [d']} \sum_j |w_{i,j}|^2 |x_{s_i(j)} - x'_{s_i(j)}|^2}{\|x - x'\|_2^2}$$

Rearranging terms, we express:

$$\sum_{i \in [d']} \sum_j |w_{i,j}|^2 |x_{s_i(j)} - x'_{s_i(j)}|^2 = \sum_{k \in [d]} \sum_{i,j : s_i(j) = k} |w_{i,j}|^2 |x_k - x'_k|^2$$

If $\sum_{i,j : s_i(j) = k} |w_{i,j}| \leqslant \omega$ for all k and for all layers, *i.e.*, the absolute sum of the input weights per neuron is bounded by ω, we can write $\sum_{k \in [d]} \sum_{i,j : s_i(j) = k} |w_{i,j}|^2 |x_k - x'_k|^2 \leqslant \omega^2 \sum_{k \in [d]} |x_k - x'_k|^2 \leqslant \omega^2 \|x - x'\|_2^2$, hence,

$$\frac{\|\hat{x} - \hat{x}'\|_2}{\|x - x'\|_2} \leqslant \omega.$$

For max-pooling and average-pooling layers, the inequality holds with $\omega = 1$; since, we can express max-pooling as a convolution where only one weight is 1 and the rest is 0; and similarly, we can express average-pooling as a convolution where the weights sum up to 1.

For ReLU activation, we consider the fact that $|\max\{0, u\} - \max\{0, v\}| \leqslant |u - v|$ to write:

$$\frac{\|ReLU(x) - ReLU(x')\|_2}{\|x - x'\|_2} \leqslant 1.$$

Therefore, L-layer CNN $f(x; \theta)$ is ω^L-Lipschitz.

We now consider $\ell(z_i, z_j; \theta) = \max\{0, y_{ij}(\|f(x_i; \theta) - f(x_j; \theta)\|_2 - \beta) + \alpha\}$ as $g(h(f(x_i; \theta), f(x_j; \theta)))$ where $g(h) = \max\{0, y_{ij}(h - \beta) + \alpha\}$ is 1-Lipschitz, and $h(f, f') = \|f - f'\|_2$ is $\sqrt{2}$-Lipschitz and 1-Lipschitz in f for fixed f'. Thus, for y_i, y_j, θ fixed, $\ell(z_i, z_j; \theta) \coloneqq (y_{ij}(\|x_i - x_j\|_{f_\theta} - \beta) + \alpha)_+$ is ω^L-Lipschitz in x_i and in x_j; and $\sqrt{2}\omega^L$-Lipschitz in both, for all y_i, y_j, θ. □

Note that it is easy to show that the normalization proposed in Section 4.4:

$$\hat{v} = \begin{cases} v & \text{for } \|v\|_2 \leqslant 1 \\ v/\|v\|_2 & \text{for } \|v\|_2 \geqslant 1 \end{cases}$$

is 2-Lipschitz. Therefore, our loss is still Lipschitz continuous with normalized embeddings in our framework.

A.2. Proof for Proposition 4.1

Proposition 4.1. Given $\mathcal{S}=\{z_i\}_{i\in[m]} \overset{i.i.d.}{\sim} p_{\mathcal{Z}}$ such that $\forall k\in\mathcal{Y}$ $\{x_i|y_i=k\}$ is $\delta_{\mathcal{S}}$-cover[2] of $\mathcal{X}$, $\ell(z_i,z_j;\theta)$ is ζ-Lipschitz in x_i, x_j for all y_i, y_j and θ, and bounded by L; then with probability at least $1-\gamma$,

$$\left|\mathbb{E}_{z_i,z_j}[\ell(z_i,z_j;\theta)] - \tfrac{1}{m}\sum_{i\in[m]}\mathbb{E}_{z_j}[\ell(z_i,z_j;\theta)]\right| \leqslant \mathcal{O}(\zeta\,\delta_{\mathcal{S}}) + \mathcal{O}(L\sqrt{\log\tfrac{1}{\gamma}/m}).$$

Proof. We start with defining $\hat{\mathcal{L}}(z;\theta) \coloneqq \mathbb{E}_{z'\sim p_{\mathcal{Z}}}[\ell(z,z';\theta)]$. Note that

$$\begin{aligned}\|\hat{\mathcal{L}}(z_1;\theta) - \hat{\mathcal{L}}(z_2;\theta)\|_2 &= |\,\mathbb{E}_{z'\sim p_{\mathcal{Z}}}[\ell(z_1,z';\theta)] - \mathbb{E}_{z'\sim p_{\mathcal{Z}}}[\ell(z_2,z';\theta)]| \\ &\leqslant \mathbb{E}_{z'\sim p_{\mathcal{Z}}}[|\ell(z_1,z';\theta) - \ell(z_2,z';\theta)|].\end{aligned}$$

Therefore, $\ell(z,z';\theta)$ being ζ-Lipschitz in x for fixed x', y, y' and θ, and bounded by L implies $\hat{\mathcal{L}}(z;\theta)$ is also ζ-Lipschitz in x for all y, θ and bounded by L. Hence, we have

$$|\hat{\mathcal{L}}(z_i;\theta) - \hat{\mathcal{L}}(z;\theta)| \leqslant \zeta\,\delta_{\mathcal{S}} \quad \forall z_i, z : z_i\in\mathcal{S}, z\in\mathcal{Z}, \|z_i - z\|_2 \leqslant \delta_{\mathcal{S}}$$

From Theorem 14 of [20], we can partition $\mathcal{Z}$ into $K = \min_t\{|t| : t \text{ is } \frac{\delta_{\mathcal{S}}}{2}\text{-cover of } \mathcal{Z}\}$ disjoint sets, denoted as $\{\mathcal{R}_i\}_{i\in[K]}$, such that $\forall i : z_i\in\delta_{\mathcal{S}}$; both z_i, z being $\in\mathcal{R}_i$ implies $|\hat{\mathcal{L}}(z_i;\theta) - \hat{\mathcal{L}}(z;\theta)| \leqslant \zeta\,\delta_{\mathcal{S}}$. Hence, from Theorem 3 of [20], with probability at least $1-\gamma$, we have:

$$\begin{aligned}\left|\mathbb{E}_{z,z'\sim p_{\mathcal{Z}}}[\ell(z,z';\theta)] - \tfrac{1}{m}\sum_{i\in[m]}\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell(z_i,z;\theta)]\right| &= \left|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\hat{\mathcal{L}}(z;\theta)] - \tfrac{1}{m}\sum_{i\in[m]}\hat{\mathcal{L}}(z_i;\theta)\right| \\ &\leqslant \zeta\,\delta_{\mathcal{S}} + L\sqrt{\frac{2\,K\,\log 2 + 2\,\log 1/\gamma}{m}}\end{aligned}$$

Note that K is dependent on δ_s and satisfies $\lim_{m\to\infty}\frac{K}{m}\to 0$ ensuring that the right hand side goes to zero as more samples are exploited and the covering radius is improved. Thus, asymptotically the following holds:

$$\left|\mathbb{E}_{z_i,z_j}[\ell(z_i,z_j;\theta)] - \tfrac{1}{m}\sum_{i\in[m]}\mathbb{E}_{z_j}[\ell(z_i,z_j;\theta)]\right| \leqslant \mathcal{O}(\delta_s) + \mathcal{O}(\sqrt{\log\tfrac{1}{\gamma}/m}) \text{ with probability at least } 1-\gamma\,.$$

□

A.3. Proof for Proposition 4.2

Proposition 4.2. Given $\{z_i\}_{i\in[n]} \overset{i.i.d.}{\sim} p_{\mathcal{Z}}$ and a set $s\subset[n]$. If $s = \cup_k s'_k$ with s'_k is the δ_s-cover of $\{i\in[n] \mid y_i = k\}$ (*i.e.*, the samples in class k), $\ell(z_i,z_j;\theta)$ is ζ-Lipschitz in x_i, x_j for all y_i, y_j and θ, and bounded by L, $e(\mathcal{A}_{s\times[n]})$ training error; then with probability at least $1-\gamma$ we have:

$$\left|\tfrac{1}{n^2}\sum_{i,j\in[n]\times[n]}\ell(z_i,z_j;\mathcal{A}_{s\times[n]}) - \tfrac{1}{|s|\,n}\sum_{i,j\in s\times[n]}\ell(z_i,z_j;\mathcal{A}_{s\times[n]})\right| \leqslant \mathcal{O}(\zeta\,\delta_s) + \mathcal{O}(e(\mathcal{A}_{s\times[n]})) + \mathcal{O}(L\sqrt{\log\tfrac{1}{\gamma}/n})$$

Proof. We are given a condition on s that we can partition $\mathcal{Z}$ into $m = |s|$ disjoint sets such that any sample from the dataset $(x_i,c), i\in[n]$, has a corresponding sample from s, $(x'_j,c), j\in s$ within δ_s ball. Thus, we start with partitioning $\mathcal{Z}$ into s disjoint sets as $\mathcal{Z} = \cup_i\mathcal{S}_i$ with $\mathcal{S}_i\cap\mathcal{S}_j = \emptyset$, $\forall i\neq j$.

We define $\ell_{[n]}(z) = \frac{1}{n}\sum_{i\in[n]}\ell(z,z_i,\mathcal{A}_{s\times[n]})$ and $\ell_s(z) = \frac{1}{m}\sum_{i\in s}\ell(z,z_i,\mathcal{A}_{s\times[n]})$ for the sake of clarity. Hence, we are interested in bounding $|\frac{1}{n}\sum_{[n]}\ell_{[n]}(z_i) - \frac{1}{m}\sum_s\ell_{[n]}(z_i)|$. We proceed with using triangle inequality to write:

$$\begin{aligned}&\left|\tfrac{1}{n}\sum_{i\in[n]}\ell_{[n]}(z_i) - \tfrac{1}{m}\sum_{i\in s}\ell_{[n]}(z_i)\right| \\ &\qquad\leqslant \underbrace{\left|\tfrac{1}{n}\sum_{i\in[n]}\ell_{[n]}(z_i) - \sum_{i\in s}\tfrac{n_i}{n}\ell_{[n]}(z_i)\right|}_{(T1)} + \underbrace{\left|\sum_{i\in s}\tfrac{n_i}{n}\ell_{[n]}(z_i) - \tfrac{1}{m}\sum_{i\in s}\ell_{[n]}(z_i)\right|}_{(T2)}\end{aligned}$$

[2] $\mathcal{S}\subset\mathcal{S}'$ is $\delta_{\mathcal{S}}$-cover of $\mathcal{S}'$ if $\forall z'\in\mathcal{S}'$, $\exists z\in\mathcal{S}$ such that $\|z-z'\|_2\leqslant\delta_{\mathcal{S}}$.

For term $(T1)$ we write:

$$(T1) \leqslant \frac{1}{n} \sum_{i\in[m]} \sum_{z_j\in\mathcal{S}_i} |\ell_{[n]}(z_{s(i)}) - \ell_{[n]}(z_j)| \overset{(1)}{\leqslant} \zeta\, \delta_s$$

where in (1), we use ζ-Lipschitz of the loss function and the condition $|z_{s(i)} - z_j| \leqslant \delta_s$, $\forall z_j \in \mathcal{S}_i$.

Using triangle inequality, we bound term $(T2)$ as:

$$\Big|\sum_{i\in s} \tfrac{n_i}{n}\ell_{[n]}(z_i) - \tfrac{1}{m}\sum_{i\in s}\ell_{[n]}(z_i)\Big| \leqslant \Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_s(z)] - \mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_{[n]}(z)]\Big|^{(T2.1)}$$
$$+ \Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_{[n]}(z)] - \sum_{i\in s}\tfrac{n_i}{n}\ell_{[n]}(z_i)\Big|^{(T2.2)} + \Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_s(z)] - \tfrac{1}{n}\sum_{i\in[n]}\ell_s(z_i)\Big|^{(T2.3)}$$

where we use $\frac{1}{m}\sum_s \ell_{[n]}(z_i) = \frac{1}{n}\sum_{[n]}\ell_s(z_i)$ in $(T2.3)$.

For $(T2.1)$ we have:

$$(T2.1) \leqslant \Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\tfrac{1}{m}\sum_{i\in s}\ell(z_i,z) - \tfrac{1}{n}\sum_{i\in[n]}\ell(z_i,z)]\Big|$$

where we abuse the notation for the sake of clarity and drop parameter $\mathcal{A}_{s\times[n]}$ dependency from the loss. Rearranging the terms, we have:

$$(T2.1) \leqslant \Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\tfrac{1}{m}\sum_{i\in[m]}\tfrac{n-m\,n_i}{n}\ell(z_{s(i)},z)]\Big| + \Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\tfrac{1}{n}\sum_{i\in[m]}\sum_{j\in\mathcal{S}_i}\ell(z_{s(i)},z) - \ell(z_j,z)]\Big|$$

where similar to $(T1)$, the second summand is upper bounded by $\zeta\,\delta_s$. Using triangle inequality for the first summand, we write:

$$\Big|\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\tfrac{1}{m}\sum_{i\in[m]}\tfrac{n-m\,n_i}{n}\ell(z_{s(i)},z)]\Big| \leqslant (T2.3) + e(\mathcal{A}_{s\times[n]})$$

Hence, we have:

$$(T2.1) \leqslant \zeta\,\delta_s + (T2.3) + e(\mathcal{A}_{s\times[n]})$$

where from Hoeffding's Bound, $(T2.3) \leqslant L\sqrt{\log\frac{1}{\gamma}/2n}$ with probability at least $1-\gamma$:

Finally, we express $(T2.2)$ as:

$$(T2.2) = \Big|\sum_{i\in[m]}\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_{[n]}(z)\mid z\in\mathcal{S}_i]p(z\in\mathcal{S}_i) - \sum_{i\in s}\tfrac{n_i}{n}\ell_{[n]}(z_i)\Big|$$
$$\leqslant \Big|\sum_{i\in[m]}\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_{[n]}(z)\mid z\in\mathcal{S}_i]\tfrac{n_i}{n} - \sum_{i\in s}\tfrac{n_i}{n}\ell_{[n]}(z_i)\Big|$$
$$+ \Big|\sum_{i\in[m]}\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_{[n]}(z)\mid z\in\mathcal{S}_i]p(z\in\mathcal{S}_i) - \sum_{i\in[m]}\mathbb{E}_{z\sim p_{\mathcal{Z}}}[\ell_{[n]}(z)\mid z\in\mathcal{S}_i]\tfrac{n_i}{n}\Big|$$

Rearranging the terms we have:

$$(T2.2) \leqslant \sum_{i\in[m]}\tfrac{n_i}{n}\max_{z\in\mathcal{S}_i}|\ell_{[n]}(z) - \ell_{[n]}(z_{s(i)})| + \max_{z\in\mathcal{Z}}|\ell_{[n]}(z)|\sum_{i\in[m]}\Big|\tfrac{n_i}{n} - p(z\in\mathcal{S}_i)\Big|$$

where the first summand is bounded above by $\zeta\,(\delta_s + \varepsilon(n))$ owing to loss being ζ-Lipschitz. Here, we denote $\varepsilon(n)$ as the covering radius of $\mathcal{Z}$, *i.e.*, the dataset, $\{x_i, y_i\}_{[n]}$ is $\varepsilon(n)$-cover of $\mathcal{X}\times\mathcal{Y}$. We note that $(n_i)_{i\in[m]}$ is an $i.i.d.$ multinomial random variable with parameters n and $(p_{\mathcal{Z}}(z\in\mathcal{S}_i))_{i\in[m]}$. Thus, by the Breteganolle-Huber-Carol inequality (Proposition A6.6 of [15]), we have :

$$(T2.2) \leqslant \zeta\,(\delta_s + \varepsilon(n)) + L\sqrt{\tfrac{2m\log 2 + 2\log 1/\gamma}{n}}$$

Finally, with probability at least $1-\gamma$, we end up with:

$$\Big|\tfrac{1}{n}\sum_{i\in[n]}\ell_{[n]}(z_i) - \tfrac{1}{m}\sum_{i\in s}\ell_{[n]}(z_i)\Big| \leqslant \zeta\,(3\,\delta_s + \varepsilon(n)) + e(\mathcal{A}_{s\times[n]}) + L\,(\sqrt{\log\tfrac{1}{\gamma}/2n} + \sqrt{\tfrac{2m\log 2 + 2\log 1/\gamma}{n}})$$

□

Corollary 4.2.1. Generalization performance of the proxy-based methods can be limited by the maximum of distances between the proxies and the corresponding class samples in the dataset.

Proof. The covering radius for each class subset is the maximum distance between the corresponding class samples and the class proxy. We at least know that the generalization error is bounded above with a term proportional to that distance. □