Cross-Batch Memory for Embedding Learning
Supplementary Materials

1. Results on More Datasets

We further verify the effectiveness of our Cross-Batch Memory (XBM) on three more datasets. CUB-200-2011 (CUB) [11] and Cars-196 (Car) [5] are two widely used fine-grained datasets, which are relatively small. DeepFashion2 [2] is a large-scale dataset just released recently. The details and evaluation protocols of the three datasets are described as below.

CUB-200-2011 (CUB) contains 11,788 birds images of 200 classes. There are about 60 images/class. Following [11], we use 5,864 images of 100 classes for training and the remaining 5,924 images for testing.

Cars-196 (Cars) contains 16,185 images in 196 model categories, Following [5], we use the first 98 models for training with the rest for testing.

DeepFashion2 contains 216K clothes images with over 686K commercial-consumer pairs in the training set, whose size is **nearly 7 times** of In-shop. We use ground truth boxes in training and testing. We follow evaluation protocol described in [2], using 10,990 commercial images with 32,550 items as a query set, and 21,438 commercial images with 37,183 items as a gallery set.

XBM meets Pair-based DML. We show that XBM consistently improves the performance of various pair-based methods on the three datasets. For instance, by applying our XBM to the conventional contrastive loss, we achieve significant Recall@1 improvements on CUB with +4.4% and Cars with +7.8%, as shown in Table 3. On the large-scale DeepFashion2, XBM has a large improvement of +11.2% as shown in Table 5.

Comparison with the State-of-the-art. We compare our method with existing methods in Table 4. On CUB, our XBM with a contractive loss achieves the best recall@1 of 65.8% without any tricks. On Cars, except ABE [4] and A-Bier [8], our XBM augmented contrastive loss reaches the best performance using GoogleNet.

In fact, we observed that several tricks can improve the performance significantly on small-scale datasets. For example, freezing BN layer [12, 9] can increase recall@1 by more than 2% on CUB and Cars, or applying a 10 times smaller learning rate on the pretrained backbones [7]. *However, these tricks show no effect on large-scale datasets* e.g. *SOP, In-shop and VehicleID, which contains sufficient data to mitigate overfitting.* Note that to demonstrate the actual effectiveness of our XBM module, the performance of our XBM reported here was trained without such bells and whistles.

momentum m	0	0.01	0.1	0.5	0.9
SOP	**78.2**	77.4	**78.2**	76.8	75.8
In-shop	**89.3**	88.9	**89.3**	87.0	83.7
CUB	60.3	60.3	60.0	60.2	**61.8**
Cars	78.8	79.1	79.2	78.3	**80.6**

Table 1: Recall@1$(\%)$ performance of moving average update mechanism with different momentum m.

learning rate α	0	0.01	0.1	0.5	0.9
SOP	**78.2**	77.2	77.6	**78.2**	**78.2**
In-shop	**89.3**	88.7	89.0	87.8	88.4
CUB	60.3	60.2	**60.7**	60.0	59.4
Cars	78.8	78.8	**79.3**	77.9	77.8

Table 2: Recall@1$(\%)$ performance of BP update mechanism with different memory learning rates α

2. Memory Update

We develop a simple enqueue-dequeue mechanism to update the memory bank of our XBM: enqueue the latest features, and at the same time dequeue the oldest ones. In this experiment, we evaluate two alternative memory update mechanisms: moving average [13, 14] and back-propagation [6], on SOP, In-shop, CUB and Cars datasets with GoogleNet as backbone. Furthermore, we also conduct a faster updated XBM to investigate the effect of feature drift.

Moving Average Update. Update a memory embedding $\widetilde{\boldsymbol{v}}_i$ with its current feature $\boldsymbol{v}_i$ as following:

$$\widetilde{\boldsymbol{v}}_i = m\widetilde{\boldsymbol{v}}_i + (1-m)\boldsymbol{v}_i$$
$$\widetilde{\boldsymbol{v}}_i = \widetilde{\boldsymbol{v}}_i / ||\widetilde{\boldsymbol{v}}_i||_2,$$

where m is the momentum for the moving average update.

The embeddings in memory is updated slower when the momentum m becomes larger. We study the impact of momentum m to the performance in Table 1. We observed that training with a large momentum can *benefit a small dataset* (*e.g*. CUB or Cars), but may *impair* the performance on a large-scale dataset (*e.g*. SOP or In-shop). It is reasonable

	CUB						Cars					
Recall@K(%)	1	2	4	8	16	32	1	2	4	8	16	32
Contrastive	57.5	69.0	78.8	86.3	92.0	96.0	72.5	81.3	87.9	92.4	95.3	97.5
Contrastive w/ M	**61.9**	**72.9**	**81.2**	**88.6**	**93.5**	**96.5**	**80.3**	**87.1**	**91.9**	**95.1**	**97.3**	**98.2**
Triplet	58.1	69.6	79.7	87.5	92.8	96.2	72.4	81.7	88.1	92.7	95.5	97.3
Triplet w/ M	**60.0**	**71.1**	**80.7**	**88.0**	**93.2**	**96.4**	**78.5**	**86.4**	**91.6**	**94.8**	**96.9**	**98.4**
MS	58.2	69.8	79.9	87.3	92.8	96.0	75.7	84.6	90.1	94.4	96.9	98.4
MS w/ M	**61.8**	**72.3**	**81.5**	**88.5**	**93.0**	**96.1**	**76.5**	**84.1**	**90.0**	**93.8**	**96.3**	**98.0**

Table 3: Retrieval results of XBM augmented ('w/ M') pair-based DML methods and baseline methods with GoogleNet on CUB and Cars.

		CUB						Cars					
Recall@K (%)		1	2	4	8	16	32	1	2	4	8	16	32
Smart Mining [3]	G^{64}	49.8	62.3	74.1	83.3	-	-	64.7	76.2	84.2	90.2	-	-
HDC [10]	G^{384}	53.6	65.7	77.0	85.6	91.5	95.5	73.7	83.2	89.5	93.8	96.7	98.4
A-BIER [8]	G^{512}	57.5	68.7	78.3	86.2	91.9	95.5	82.0	89.0	93.2	96.1	97.8	98.7
ABE [4]	G^{512}	60.6	71.5	79.8	87.4	-	-	85.2	90.5	94.0	96.1	-	-
Clustering [10]	B^{64}	48.2	61.4	71.8	81.9	-	-	58.1	70.6	80.3	87.8	-	-
ProxyNCA [6]	B^{64}	49.2	61.9	67.9	72.4	-	-	73.2	82.4	86.4	87.8	-	-
HTL [1]	B^{512}	57.1	68.8	78.7	86.5	92.5	95.5	81.4	88.0	92.7	95.7	97.4	99.0
MS [12]	B^{512}	65.7	**77.0**	**86.3**	**91.2**	**95.0**	**97.3**	84.1	90.4	94.0	96.5	98.0	98.9
SoftTriple [9]	B^{512}	65.4	76.4	84.5	90.4	-	-	**84.5**	**90.7**	**94.5**	**96.9**	-	-
Contrative w/ M	G^{512}	61.9	72.9	81.2	88.6	93.5	96.5	80.3	87.1	91.9	95.1	97.3	98.2
Contrative w/ M	B^{512}	**65.8**	75.9	84.0	89.9	94.3	97.0	82.0	88.7	93.1	96.1	97.6	98.6

Table 4: Recall@$K(\%)$ performance on CUB and Cars.

Recall@K(%)		1	10	20
Match RCNN [2]		26.8	57.4	66.5
Contrative	G^{512}	29.3	51.9	60.3
Contrative w/ M	G^{512}	40.5	63.2	69.4
Contrative w/ M	B^{512}	40.9	63.3	69.6
Contrative w/ M	$\mathbf{R^{128}}$	**41.9**	**64.6**	**70.7**

Table 5: Recall@$K(\%)$ performance on DeepFashion2.

because on a small-scale dataset, the training epoch is short (*e.g*. 100 iters.), which enables a small feature drift between adjacent epochs, while a large-scale dataset has a longer epoch, and the features computed at the past epochs are highly possible to be out-of-date, and thus a large momentum may hinder the training process. Moreover, the moving average update may benefit the training by enhancing the embeddings in the memory by *aggregating its embeddings of an instance with different augmentations* when the feature drift is small.

Back-Propagation (BP) Update. Besides substituting the memory features of instances sampled into current mini-batch, BP method updates the memory features of each instance based on its gradients computed at back-propagation (BP):

$$\widetilde{\boldsymbol{v}}_i = \widetilde{\boldsymbol{v}}_i + \alpha \frac{\partial \widetilde{\mathcal{L}}}{\partial \widetilde{\boldsymbol{v}}_i}$$
$$\widetilde{\boldsymbol{v}}_i = \widetilde{\boldsymbol{v}}_i / ||\widetilde{\boldsymbol{v}}_i||_2,$$

where α is the learning rate of memory features.

In BP update, the memory embeddings are optimized along with the model, and serve as proxies in proxy-based DML methods [6, 9]. Obviously, BP requires much more memory and computational cost to compute and save the gradients compared to our XBM. However, it cannot obtain clear performance improvements in all datasets as shown in Table 2. This suggests that the embeddings drift slowly, and the past mini-batches can largely represent the distribution of current embedding space.

Recall@$k(\%)$	1	10	100	1000
update $\times 1$	77.4	89.6	95.4	98.4
update $\times 10$	77.4	89.8	95.5	98.5

Table 6: Recall@$k(\%)$ performance on SOP with update $\times 1$ and $\times 10$ configurations.

Faster Update. Generally, we update the XBM with **one mini-batch** at each iteration ($\times 1$). To further evaluate the side effect of feature drift in our XBM, at each iteration, ten mini-batches ($\times 10$) are enqueued into the XBM memory queue. This accelerates XBM update 10 times faster, which makes the feature drift smaller. As shown in Table 6, the $\times 10$ update cannot bring clear performance gain, which suggests that the natural feature drift is slow enough and cannot hinder the performance.

3. Feature Drift on General Tasks

"Slow drift" phenomena not only exist in pair-based DML, but also happen in other machine learning tasks, *e.g.* image recognition. Figure 1 illustrates that the normalized

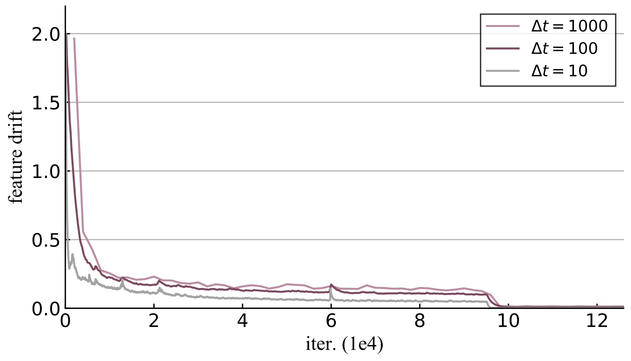

Figure 1: **Feature drift** with different steps of ResNet50 on ImageNet trained *from scratch*. The embeddings drift within a relatively small distance even under a large interval, *e.g.* $\Delta t = 1000$.

embeddings at global pooling layer of ResNet50 drift at a slow rate when trained with cross entropy loss on ImageNet dataset.

4. Hyperparameters

In Table 7, we list all the key hyperparameters applied in our experiments. Our XBM achieves outstanding performance on large-scale datasets and comparable results on small-scale datasets without any training trick or large training iterations.

	init. lr.	lr.×0.1	total iter.	$R_{\mathbb{M}}(\%)$	α	m
SOP	3e-4	24k	34k	1	0	0
In-shop	3e-4	24k	34k	0.2	0	0
VehicleID	1e-4	30k	50k	0.5	0	0
DeepFashion2	3e-4	20k	36k	1	0	0
CUB	3e-5	-	1.4k	-	0.2	0.9
Cars	1e-4	1.4k	2k	-	0.1	0.9

Table 7: **Hyperparameters** used in training memory augmented models to compare with state-of-the-art.

5. Proof of Lemma 1

Proof. The gradient of the accurate loss and the approximated loss can be computed as below:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_i}\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{\theta}} = \boldsymbol{v}_j \frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{\theta}}$$
$$\frac{\partial \widetilde{\mathcal{L}}}{\partial \boldsymbol{\theta}} = \frac{\partial \widetilde{\mathcal{L}}}{\partial \boldsymbol{v}_i}\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{\theta}} = \widetilde{\boldsymbol{v}}_j \frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{\theta}}$$

Then, the gradient error is:

$$\begin{aligned} \left|\left|\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} - \frac{\partial \widetilde{\mathcal{L}}}{\partial \boldsymbol{\theta}}\right|\right|_2^2 &= \left|\left|(\boldsymbol{v}_j - \widetilde{\boldsymbol{v}}_j)\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{\theta}}\right|\right|_2^2 \\ &\leq \left|\left|\boldsymbol{v}_j - \widetilde{\boldsymbol{v}}_j\right|\right|_2^2 \left|\left|\frac{\partial \boldsymbol{v}_i}{\partial \boldsymbol{\theta}}\right|\right|_2^2 \\ &\leq \left|\left|\frac{\partial f(\boldsymbol{x}_i; \boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}}\right|\right|_2^2 \epsilon \\ &\leq C\epsilon. \end{aligned} \tag{1}$$

□

Empirically, we observed that $\left|\left|\frac{\partial f(\boldsymbol{x}_i;\boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}}\right|\right|_2^2$ is usually less than 1 so that the gradient error can be strictly controlled by the small feature drift.

6. Visualization.

We visualize a number of samples to investigate the performance of our XBM, including the mined negatives in training and the retrieved examples in testing. All examples are *randomly* selected from SOP, In-shop and VehicleID by following some rules.

Figure 2 demonstrates the hard negatives mined from memory with over 0.5 similarities. The negatives in each row are uniformly sampled from a sequence sorted by similarities. These results clearly demonstrate that the proposed XBM can provide **diverse**, visually related, and even fine-grained samples to construct informative negative pairs.

Furthermore, we select the anchors having a similarity with the hardest samples over 0.8, and show the top 10 negatives in Figure 3. Some of the presented negatives are extremely similar or even exactly the same items with corresponding anchor images.

References

[1] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R Scott. Deep metric learning with hierarchical triplet loss. In *ECCV*, 2018. 2

[2] Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo. A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. *CVPR*, 2019. 1, 2

[3] Ben Harwood, Vijay Kumar B G, Gustavo Carneiro, Ian Reid, and Tom Drummond. Smart mining for deep metric learning. In *ICCV*, 2017. 2

[4] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *ECCV*, 2018. 1, 2

[5] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshops*, 2013. 1

[6] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *ICCV*, 2017. 1, 2

[7] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016. 1

[8] M. Opitz, G. Waltner, H. Possegger, and H. Bischof. Deep Metric Learning with BIER: Boosting Independent Embeddings Robustly. *PAMI*, 2018. 1, 2

[9] Qi Qian, Lei Shang, Baigui Sun, and Juhua Hu. Softtriple loss: Deep metric learning without triplet sampling. *ICCV*, 2019. 1, 2

[10] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. In *CVPR*, 2017. 2

[11] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Master's thesis, 2011. 1

[12] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *CVPR*, 2019. 1, 2

[13] Zhirong Wu, Alexei A Efros, and Stella Yu. Improving generalization via scalable neighborhood component analysis. In *ECCV*, 2018. 1

[14] Zhun Zhong, Liang Zheng, Zhiming Luo, Shaozi Li, and Yi Yang. Invariance matters: Exemplar memory for domain adaptive person re-identication. In *CVPR*, 2019. 1

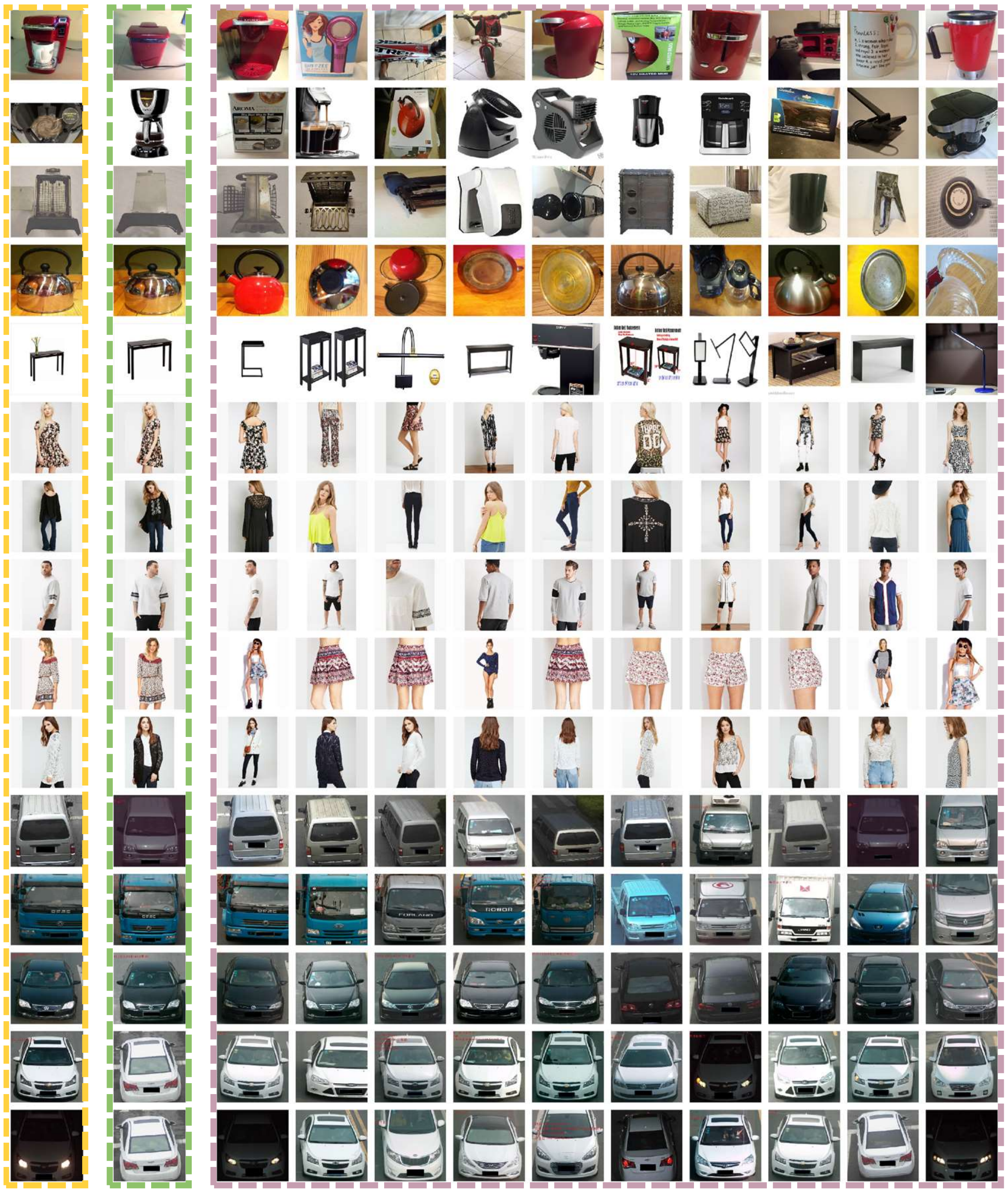

Figure 2: Given an anchor image (yellow), we present examples of a positive (green), and multiple negatives mined from memory (purple) uniformly sorted from hard to simple. The demonstrated anchors are *randomly* chosen from training datasets.

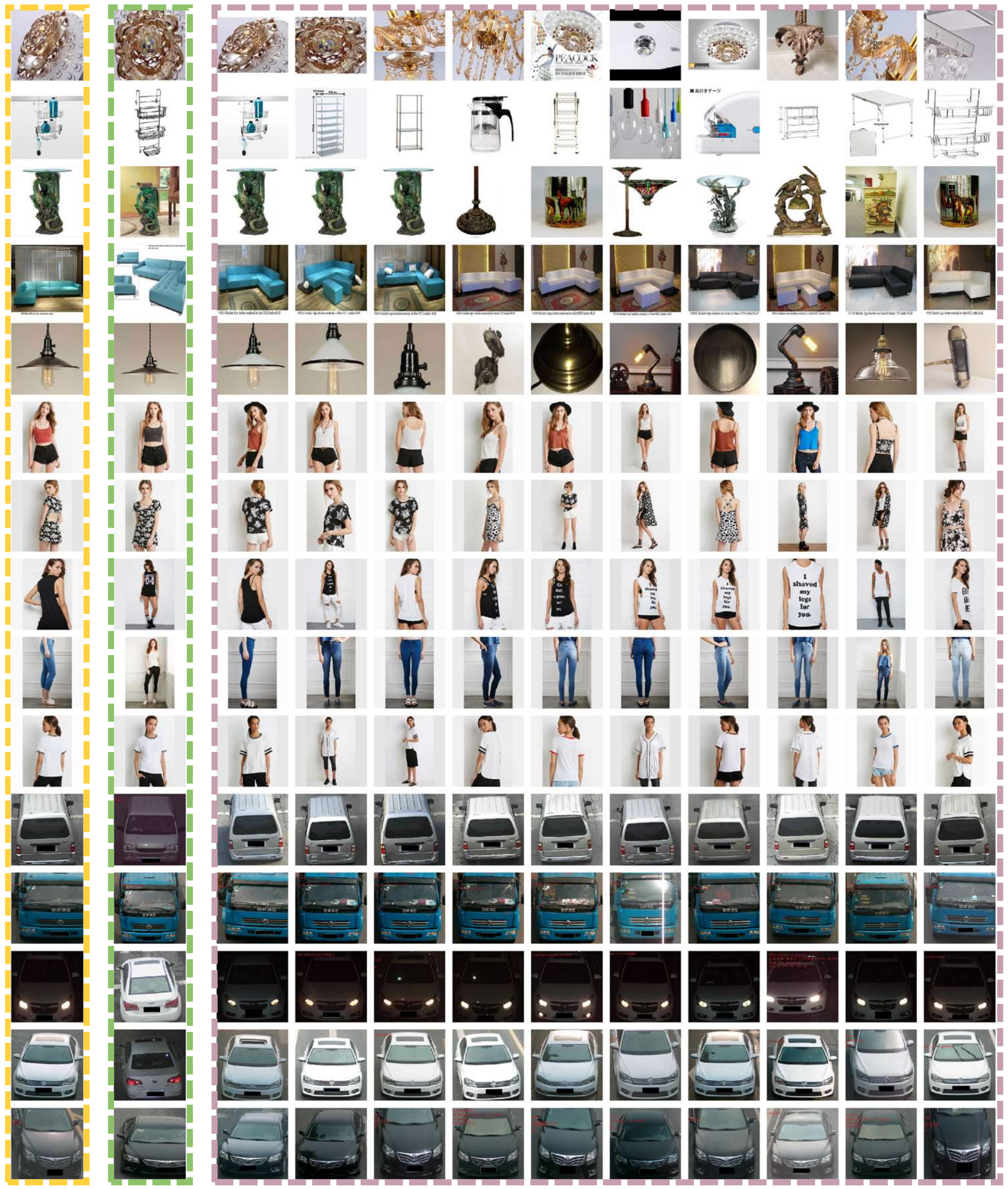

Figure 3: Given an anchor image (yellow), we present examples of a positive (green), and top 10 negatives mined from memory (purple). The examples are *randomly* chosen from anchors whose top 1 negative has over 0.8 similarity.

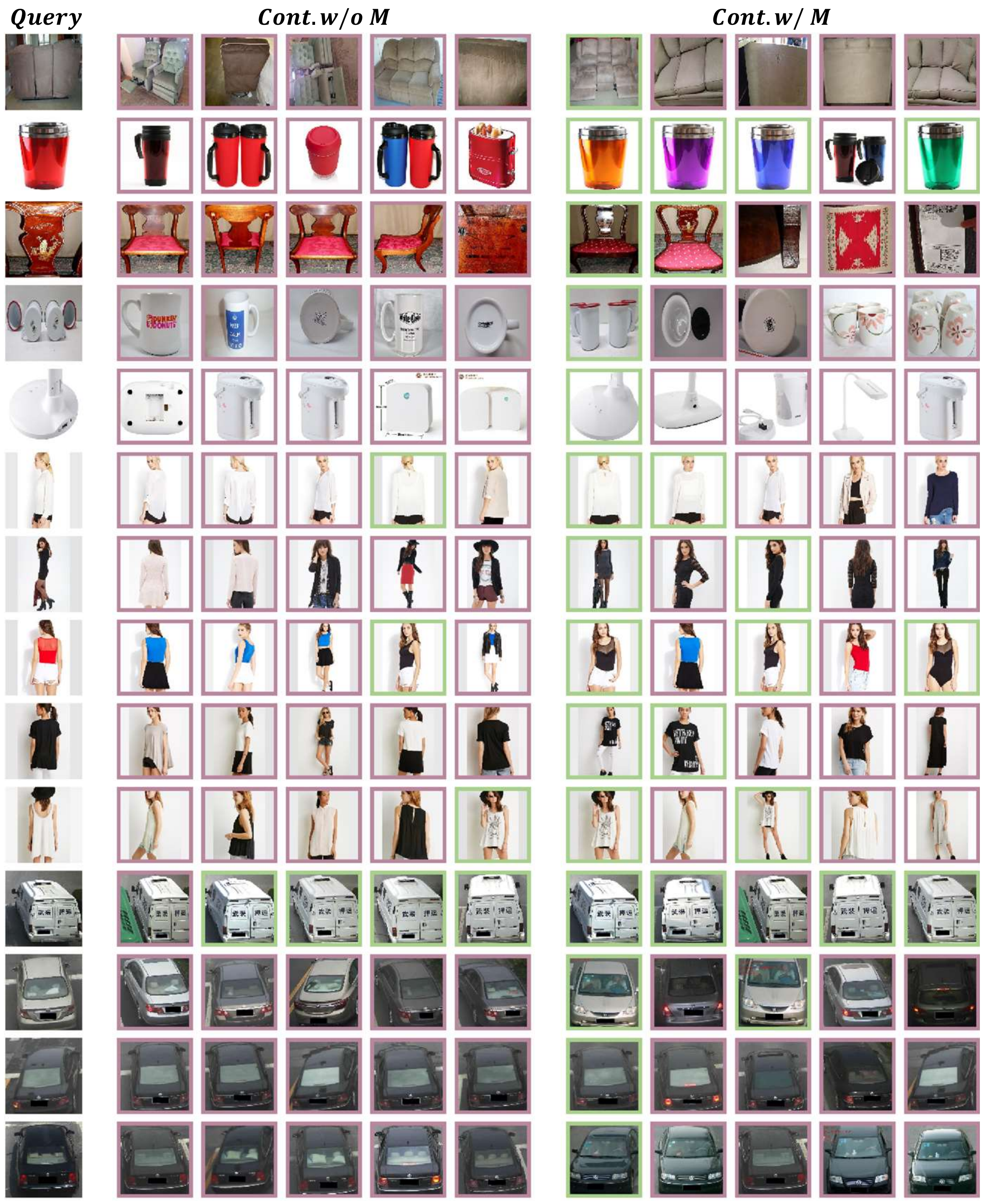

Figure 4: Top 5 retrieved images w/o and w/ memory module. We randomly select the examples with the correct top 1 predictions given by our XBM but incorrect by the baseline model. Correct results are highlighted with green, while incorrect purple.

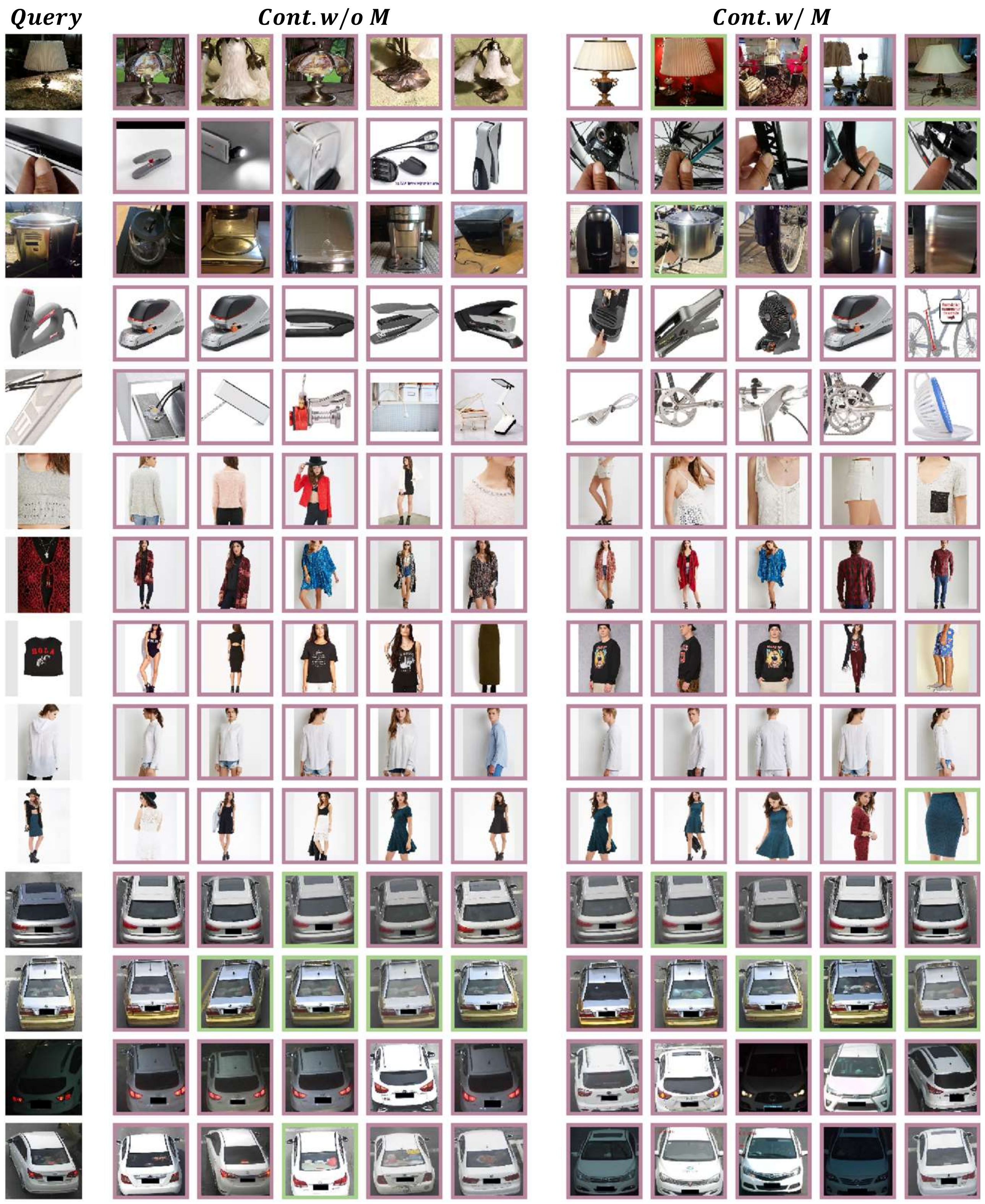

Figure 5: Top 5 retrieved images w/o and w/ memory module. We randomly select the examples with the wrong top 1 predictions by both the baseline model and our XBM. Correct results are highlighted with green, while incorrect purple.