Exploring deep learning for intelligent image retrieval

Chen, W.

Chapter 5

On the Exploration of Incremental Learning for Fine-grained Image Retrieval

As noted, the wide popularity of mobile devices make the large image collections available to access. Deep models are usually trained for retrieval on limited categories and cannot be extended to new incoming data. To satisfy a more practical retrieval, deep models are required to learn on a stream of data sequentially. This motivates our research on what kind of knowledge is more beneficial for making a deep model learn incrementally and reduce catastrophic forgetting.

In this chapter, we consider the problem of fine-grained image retrieval in an incremental setting, when new categories are added over time. On the one hand, repeatedly training the representation on the extended dataset is time-consuming. On the other hand, fine-tuning the learned representation only with the new classes leads to catastrophic forgetting. To this end, we propose an incremental learning method to mitigate retrieval performance degradation. Without accessing any samples of the original classes, the classifier of the original network provides soft "labels" to transfer knowledge to train the adaptive network, so as to preserve the previous capability for classification. More importantly, a regularization function based on Maximum Mean Discrepancy is devised to minimize the discrepancy of new classes features from the original network and the adaptive network, respectively.

Keywords

Incremental learning, Fine-grained image retrieval, Knowledge distillation, Feature correlation, Maximum mean discrepancy

This chapter is based on the following publication [37]:

- Chen, W., Liu, Y., Wang, W., Tuytelaars, T., Bakker, E, and Lew, M.S., "On the Exploration of Incremental Learning for Fine-grained Image Retrieval." The British Machine Vision Conference (BMVC), 2020, pp. 1-10.

5.1 Introduction

In an era when the number of images is increasing, deep models for fine-grained image retrieval (FGIR) are required to be adaptable for new incoming classes. However, current image retrieval approaches are focusing mainly on static datasets and are not suited for incremental learning scenarios. To be specific, deep networks well-trained on original classes will under-perform on new incoming classes.

When new classes are added into an existing dataset, joint training on all classes allows to guarantee the performance. However, as the number of new classes increases sequentially, the repetitive re-training is time-consuming. Alternatively, fine-tuning makes the network adapt to new classes and achieve good performance on these classes. However, when the original classes become inaccessible during fine-tuning, the performance of the original classes degrades dramatically because of catastrophic forgetting, a phenomenon that occurs when a network is trained sequentially on a series of new tasks and the learning of these tasks interferes with performance on previous tasks, as shown in Figure 5.1(a).

Most of incremental learning methods are exploited for image classification, which is robust and forgiving as long as features remain within the classification boundaries. In contrast, image retrieval focuses more on the discrimination in the feature space rather than the classification decisions. Especially for FGIR, small changes on visual features may have a big impact on the retrieval performance. Additionally, we find that standard methods like Learning without Forgetting (*i.e.* LwF [212]) and Elastic Weight Consolidation (*i.e.* EWC [213]) are insufficient for this problem because the distillation is not on the actual feature space (see Section 5.4.2 and 5.4.3).

Considering the above limitations, we alleviate the problem of incremental fine-grained image retrieval. We regularize the updates of the model to simultaneously retain preservation on original classes and adaptation on new classes. Importantly, to avoid the repeated training, the samples of the original classes are not used when learning the new classes. The classifier of the original network provides soft "labels" to transfer knowledge to train the adaptive network using the distillation loss function [214]. This focuses on pair-wise similarity but can not well quantify the distance between two feature distributions. This limitation inspires us to adopt a regularization term based on Maximum Mean Discrepancy (MMD) [215] to minimize the discrepancy between the features derived from an original network and an adaptive network, respectively. Moreover, the cross-entropy loss and triplet loss are utilized to identify subtle differences among sub-categories.

The novelty of the proposed method can be summarized two-fold. First, our work extends FGIR in the context of incremental learning. This is the first work to study this problem, to the best of our knowledge. Second, we propose a deep network, which includes a knowledge distillation loss and a MMD loss, for incremental

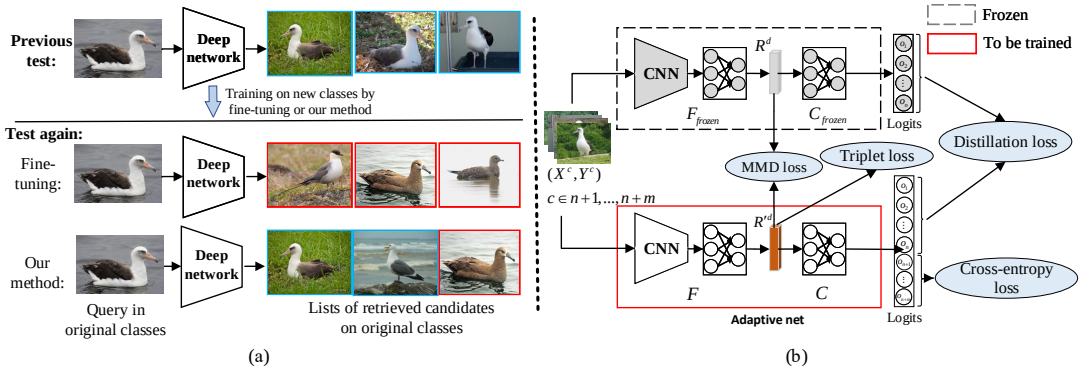


Figure 5.1: (a) Illustration of catastrophic forgetting for FGIR. Our method aims to maintain good performance on the original classes where the inaccurate returned images are in red box and correct results are in blue box. (b) Framework of our method. The only inputs for the adaptive net $\mathcal{B}$ are m new classes and labels $(\mathbf{X}^{c'}, \mathbf{Y}^{c'})$, $c' \in (n+1, ..., n+m)$. The frozen net $\mathcal{A}$ is firstly trained on n original classes and then copied as initialization for net $\mathcal{B}$.

learning without using any samples from the original classes. It achieves significant improvements over previous incremental learning methods.

5.2 Related Work

Incremental learning is the process of transferring learned knowledge from an original model to an incremental model. It has been studied in lots of tasks like image classification [212], image generation [216], object detection [217], hashing image retrieval [218], and semantic segmentation [219]. To overcome catastrophic forgetting, numerous methods have been proposed. For example, a subset of data (exemplars) of original classes are stored into an external memory, and the forgetting is thereby avoided by replaying these exemplars [220]. Recently, GANs [180] are used to synthesize samples with respect to the previous data distributions [221], which avoids the shortcomings of memory-consuming and exemplar-choosing, but generating real-like images with complex semantics is a challenging task. Alternatively, regularization methods constrain the objective functions or parameters of deep networks to preserve the previously learned knowledge. The distillation loss function [214] is used to transfer knowledge of old classes [212]. The importance weight per parameter is estimated based on the old classes, and then is used as regularization to penalize essential parameter changes when training on new incoming classes [213].

5.3 Proposed Approach

Problem formulation Given a fine-grained dataset which includes n class labels $(\mathbf{X}^c, \mathbf{Y}^c)$ where $c \in (1, ..., n)$, each sub-category c has a different amount of images in $\mathbf{X}^c$ and the ground-truth labels $\mathbf{Y}^c$. A deep network is trained to perform the

retrieval task for the n classes. Consider the incremental learning scenario, images from m new classes are added sequentially or at once. We take as input only the images from m new incoming classes, *i.e.* $(\mathbf{X}^{c'}, \mathbf{Y}^{c'})$, where $c' \in (n+1, ..., n+m)$, to incrementally train the deep network. In this way, it is efficient to update the network with no need of re-training the original classes again. Besides, the image instances from the original classes are not always accessible due to privacy issue or memory limit. Finally, the aim is to continually train the network, to make it preserve promising performance for all seen classes.

Overall idea As shown in Figure 5.1(b), our method includes two training stages. First, we train a network $\mathcal{A}$ on the original classes using cross-entropy and triplet loss on the output logits and representations. After $\mathcal{A}$ is well-trained, we make two copies of $\mathcal{A}$: one freezing its parameters when incrementally training, and the other adapting its parameters for the m incremental classes. We refer to this adaptive network as $\mathcal{B}$. It is initialized with parameters from $\mathcal{A}$, including the feature extraction layers F_{frozen} and classifier C_{frozen}, but extends the number of neurons in its classifier C, from which the output logits are $(o'_1, o'_2, \ldots, o'_n, o'_{n+1}, \ldots, o'_{n+m})$, and previous n neurons are copied from C_{frozen}. To overcome catastrophic forgetting, we propose to integrate two regularization strategies based on knowledge distillation and maximum mean discrepancy, respectively. Given a query image from either the original classes or newly added classes, we extract the features from the fully-connected layer for image retrieval. We introduce the details of our method below.

5.3.1 Semantic preserving loss

First, we train the model with the standard cross-entropy loss. Given the logits $(o_1, o_2, ..., o_n)$ and its class label $(y_1, y_2, \ldots, y_n)$, the loss is $H(\mathbf{y}, \mathbf{o}) = -\sum(\mathbf{y} * log(softmax(\mathbf{o})))$. Note that we only use images from the new classes during incremental training, thus the classification is performed on $(o'_{n+1}, o'_{n+2}, \ldots, o'_{n+m})$, the categorical cross-entropy loss function L_{ce} is

$$L_{ce} = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i * log\big(\frac{e^{o'_i(x)}}{\sum_{j=n+1}^{n+m} e^{o'_j(x)}}\big)\Big) \tag{5.1}$$

To identify subtle differences among categories, we use the triplet loss $L_{triplet}$ by mining training samples based on feature vectors $\mathbf{R}$.

$$L_{triplet} = \frac{1}{N}\sum_{i=1}^{N}\Big(\max(0, \lambda + S_{i,neg} - S_{i,pos})\Big) \tag{5.2}$$

where $S_{i,neg}$ and $S_{i,pos}$, based on matrix multiplication (*i.e.* $S_{i,neg} = R_i R_{neg}^{\top}$), indicate the similarity of i^{th} negative and positive pairs, respectively. λ is the margin.

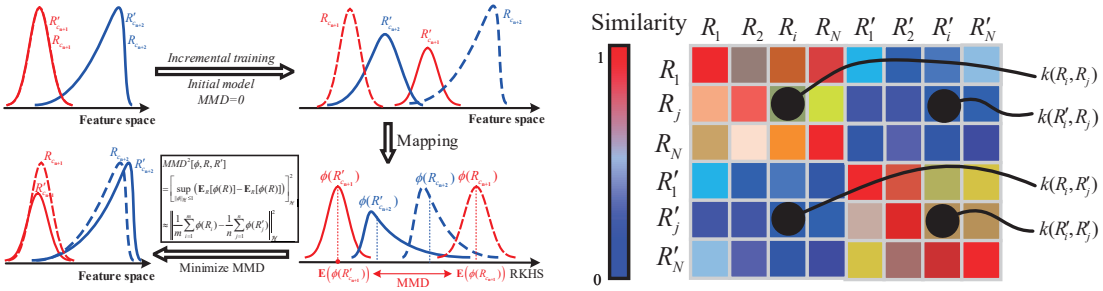


Figure 5.2: (a) The red and blue color depict the feature distributions of two categories. The dashed line indicates the distributions from the network $\mathcal{A}$, the solid line indicates that from the network $\mathcal{B}$. Since $\mathcal{A}$ is used to initialize the network $\mathcal{B}$, MMD=0 in the beginning. As training proceeds, $\mathcal{B}$ changes its output features and the MMD is expected to increase. (b) MMD for instance-to-instance similarity.

5.3.2 Knowledge distillation loss

We rewrite (F_{frozen}, C_{frozen}) as (F_f, C_f) for simplicity. Knowledge distillation loss [214] is defined to regularize the activations of the output layer in both the old and new model. To be specific, we constrain the first n values in $(o'_1, o'_2, ..., o'_n, o'_{n+1}, ..., o'_{n+m})$ as close as possible to the logits $(o_1, o_2, ..., o_n)$ from the frozen network $\mathcal{A}$. Following the method in [212], when m new classes are added at once, we compute the knowledge distillation loss by

$$L_{dist} = -\frac{1}{|\mathbf{X}^{c'}|} \sum_{x \in \mathbf{X}^{c'}}^{|\mathbf{X}^{c'}|} \sum_{k=1}^{n} \Big(p_k(x) * log[p'_k(x)] \Big) \tag{5.3}$$

where $p_k(x) = \frac{e^{o_k(x)/T}}{\sum_j^n e^{o_j(x)/T}}$ and $p'_k(x) = \frac{e^{o'_k(x)/T}}{\sum_j^n e^{o'_j(x)/T}}$, T is a temperature factor that is normally set to 2 [212]. $\mathbf{p} = \{p\}_n$ and $\mathbf{p}' = \{p'\}_n$ refer to the probabilities produced by the modified Softmax function in [214]. F_f and C_f denote the parameters of network $\mathcal{A}$. Similarly, F and C denote the parameters of network $\mathcal{B}$, as shown in Figure 5.1(b). $|\mathbf{X}^{c'}|$ indicates the number of images from the new m classes in a mini-batch. n denotes the number of the original classes. Note that n will be extended accordingly when more new classes are added.

5.3.3 Maximum mean discrepancy loss

Knowledge distillation loss focuses on constraining classification boundaries to mitigate the forgetting issue. However, for FGIR, it is more important to reduce the difference between feature distributions. For this, we adopt maximum mean discrepancy (MMD) [215] to capture the correlation of feature distributions between network $\mathcal{A}$ and $\mathcal{B}$. MMD has been used to bridge source and target distributions

such as in domain adaptation [222]. However, our work is the first to impose MMD to regularize the forgetting issue for FGIR.

Given the features R^d (d is feature dimension) from network $\mathcal{A}$ and $\mathcal{B}$, MMD measures the distance between the means of two feature distributions after mapping them into a reproducing kernel Hilbert space (RKHS). In Figure 5.2(a), we illustrate how MMD mitigates the catastrophic forgetting issue. Note that, in the Hilbert space $\mathcal{H}$, norm operation can be equal to the inner product [215]. Finally, the squared MMD distance is:

$$\begin{aligned}
&\mathrm{MMD}^2(\mathbf{R},\mathbf{R}') = ||\frac{1}{N}\sum_{i=1}^{N}\phi(R_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(R'_j)||^2_{\mathcal{H}} \\
&= \frac{1}{N^2}<\sum_{i=1}^{N}\phi(R_i) - \sum_{j=1}^{N}\phi(R'_j), \sum_{i=1}^{N}\phi(R_i) - \sum_{j=1}^{N}\phi(R'_j)>_{\mathcal{H}} \\
&= \frac{1}{N^2}\Big[\sum_{i=1}^{N}\sum_{j=1}^{N}<\phi(R_i),\phi(R_j)>_{\mathcal{H}} + \sum_{i=1}^{N}\sum_{j=1}^{N}<\phi(R'_i),\phi(R'_j)>_{\mathcal{H}} - 2\sum_{i=1}^{N}\sum_{j=1}^{N}<\phi(R_i),\phi(R'_j)>_{\mathcal{H}}\Big] \\
&\quad s.t.\ R = F_f(x),\ R' = F(x)
\end{aligned} \tag{5.4}$$

where N is batch size, and $\phi(\cdot)$ denotes the mapping function. However, it is hard to determine $\phi(\cdot)$. In RKHS, the kernel trick is used to replace the inner product in Eq. 5.4, *i.e.* $<\phi(R),\phi(R')> = k(R,R')$. Considering all the features in a mini-batch, $\mathbf{R} = \{R\}_N$ and $\mathbf{R}' = \{R'\}_N$, we define the MMD loss L_{mmd} as:

$$L_{mmd} = \mathrm{MMD}(\mathbf{R},\mathbf{R}') = \frac{1}{N}\Big[\sum_{i=1}^{N}\sum_{j=1}^{N}k(R_i,R_j) - 2\sum_{i=1}^{N}\sum_{j=1}^{N}k(R_i,R'_j) + \sum_{i=1}^{N}\sum_{j=1}^{N}k(R'_i,R'_j)\Big]^{\frac{1}{2}} \tag{5.5}$$

where $\mathrm{k(R,R')}=\exp(-(||\mathrm{R}-\mathrm{R}'||_2^2)/(2\sigma_{\mathrm{m}}^2))$, σ_m means m variances in the Gaussian kernel.

Discussion. Knowledge distillation loss focuses on constraining pair-wise similarity. However, MMD loss measures the distance between each feature vector, as depicted in Figure 5.2(b). Finally, it captures the distance of two feature distributions from the frozen net and the adaptive net. Thus, MMD loss is more powerful to quantize the correlation of two models.

Overall, the objective function in our method for incremental FGIR learning is:

$$L = \alpha L_{dist} + \beta L_{mmd} + (L_{ce} + L_{triplet}) \tag{5.6}$$

5.4 Experiments

5.4.1 Datasets and experimental settings

Datasets. We evaluate our method on the Stanford-Dogs [223] and Caltech-UCSD Birds-200 (CUB-200) [224] datasets. For the former, we use the official train/test splits. When training incrementally, we split the first 60 sub-categories (in the order of official classes) as the original classes and images from the remaining 60 sub-categories are added at once or sequentially. For the latter, we choose 60% of images from each sub-category as training set and 40% as testing set. Afterwards, we split the first 100 sub-categories (in the order of official classes) as the original classes and the remaining 100 sub-categories as new classes. The details are shown in Table 5.1.

Table 5.1: Statistics of the datasets used in our experiments.

Datasets	Training set (#Image/#Class)			Testing set (#Image/#Class)		
	Original cls.	New cls.	Total	Original cls.	New cls.	Total
Stanford-Dogs	6000/60	6000/60	12000/120	4651/60	3929/60	8580/120
CUB-200	3504/100	3544/100	7048/200	2360/100	2380/100	4740/200

Experimental settings. We use the Recall@K [131] (K is the number of retrieved samples), mean Average Precision (mAP), the precision-recall (PR) curve and feature distribution visualizations for evaluation. We adopt the Google Inception [62] to extract image features. During training, the parameters in Inception are updated using the Adam optimizer [225] with a learning rate of 1×10^{-6}, while parameters in fully-connected layers and classifier are updated with a learning rate of 1×10^{-5}. We follow the sampling strategy in [226] and each incremental process is trained 800 epochs. Following the practice in [131, 226], the output 512-D features (R^d) from fully-connected layers are used for retrieval. We set the hyper-parameters $\alpha = \beta = 1$ in Eq. 5.6, and the margin $\lambda = 0.5$ in Eq. 5.2.

5.4.2 One-step incremental learning for FGIR

We report the results for multiple classes added at once. The process includes two stages. First, we use the cross-entropy and triplet loss to train the network $\mathcal{A}$ on the original classes (100 classes for the CUB-200 dataset, 60 classes for the Stanford-Dogs dataset), denoted as $\mathcal{A}$(1-100) and $\mathcal{A}$(1-60), respectively. Second, only images of new classes are added at once to train network $\mathcal{B}$, denoted as $\mathcal{B}$(101-200) for CUB-200 and $\mathcal{B}$(61-120) for Stanford-Dogs. DIHN [218] has been explored the incremental learning for hashing-based image retrieval. However, its main difference with ours is to depend on the usage of old data as query set to avoid forgetting in their assumption. Considering no previous works for the fine-grained incremental image retrieval, we apply Learning without Forgetting (LwF) [212], Elastic Weight

Consolidation (EWC) [213], ALASSO [227], and the incremental learning for semantic segmentation (dubbed L2 loss) [219] for comparison. LwF, EWC, and ALASSO distill knowledge on classifier and network parameters, which are insufficient for incremental FGIR. L2 loss in [219] is more similar with ours where the knowledge is distilled on the classifier and intermediate feature space. Note that cross-entropy and triplet loss (*i.e.* semantic preserving loss) are combined with these three algorithms for fair comparison. The Recall@K results for the CUB-200 dataset are reported in Table 5.2.

Table 5.2: Recall@K (%) of incremental FGIR on the CUB-200 dataset when new classes are added at once. The best performance in the original class and the new class are in boldface.

Configurations	Original classes			New classes		
Recall@K(%)	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-100) (initial model)	79.41	85.64	89.63	-	-	-
+$\mathcal{B}$(101-200) w feature extraction	-	-	-	47.02	57.44	67.86
+$\mathcal{B}$(101-200) w fine-tuning	53.90	64.56	73.56	76.18	82.56	87.39
+$\mathcal{B}$(101-200) w LwF (*i.e.* L_{dist})	54.92	66.40	75.42	75.76	82.69	86.93
+$\mathcal{B}$(101-200) w ALASSO	56.91	66.65	76.57	72.48	79.50	85.67
+$\mathcal{B}$(101-200) w EWC	62.03	72.16	80.08	73.32	80.92	86.01
+$\mathcal{B}$(101-200) w L2 loss	66.48	75.68	82.67	**77.44**	**83.78**	**88.07**
+$\mathcal{B}$(101-200) w Our method	**74.41**	**82.57**	**88.52**	73.11	80.84	86.64
$\mathcal{A}$(1-200) (reference model)	77.33	85.08	89.03	76.64	83.53	89.12

In Table 5.2, the "w feature extraction" depicts when $\mathcal{A}$ directly extracts features on the new classes without re-training. The "w fine-tuning" depicts using L_{ce} and $L_{triplet}$ to train $\mathcal{A}$ on the new classes but without using L_{dist}. Overall, the network $\mathcal{B}$ suffers from the catastrophic forgetting issue and has lower performance on the original classes, whereas our method outperforms the others. As for the new classes, other three algorithms outperform ours. For example, " w L2 loss" method achieves on Recall@1 by 4.33% compared to ours (77.44%→73.11%). However, it suffers from significant performance degradation on the original classes with Recall@1 dropping by 12.93% compared to the initial model (79.41%→66.48%). For our method, the Recall@1 on the original classes is 74.41% (dropped by 5.00% from 79.41% of the initial model); the Recall@1 on the new classes is 73.11% compared to the reference model from $\mathcal{A}$(1-200) (*i.e.* Recall@1=76.64%). Similarly, the Recall@K results for the Stanford-Dogs dataset are reported in Table 5.3. We observe similar trends as the results we shown in main paper, when our method achieves good performance on the original classes and new classes with Recall@1= 76.67% and Recall@1=81.88%, respectively. Compared to the initial model on the original classes, our method has dropped Recall@1 performance by 4.00% (80.67%→76.67%).

We report the PR curves and mAP results in Figure 5.3(a), 5.3(b), and 5.3(c), respectively. Overall, when tested on the new classes, all methods share similar trends. When tested on the original classes, our method has better performance although it still has gap to reference performance. For mAP results, the reference

Table 5.3: Recall@K (%) of incremental FGIR on the Stanford-Dogs dataset when new classes are added at once. The best performance are in boldface.

Configurations	Original classes			New classes		
Recall@K(%)	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-60) (initial model)	80.67	87.27	92.20	-	-	-
+$\mathcal{B}$(61-120) w feature extraction	-	-	-	75.64	83.91	90.48
+$\mathcal{B}$(61-120) w fine-tuning	61.43	72.80	81.70	78.93	86.99	91.55
+$\mathcal{B}$(61-120) w LwF (*i.e.* L_{dist})	61.77	72.72	81.70	78.52	86.38	91.12
+$\mathcal{B}$(61-120) w EWC	62.24	73.30	82.82	78.90	86.59	91.19
+$\mathcal{B}$(61-120) w ALASSO	62.61	74.49	82.98	78.14	85.98	91.02
+$\mathcal{B}$(61-120) w L2 loss	72.07	81.44	87.47	**82.21**	88.75	92.52
+$\mathcal{B}$(61-120) w Our method	**76.67**	**85.10**	**91.14**	81.88	**88.98**	**93.36**
$\mathcal{A}$(1-120) (reference model)	79.29	86.86	91.61	82.57	88.75	93.13

results are the same as in Table 5.2. We utilize the well-trained network $\mathcal{A}$ at epoch=700 as initial model to train $\mathcal{B}$ on the new classes until convergence, we test the mAP of network $\mathcal{B}$ on the original classes. As the curves show, the network trends to degrade its accuracy on the original classes during incremental training. Similarly, we report the precision-recall curves and mAP results in Figure 5.4. We can observe these curves share with the similar trends with those from the CUB-200 dataset. Overall, our method can effectively address the catastrophic forgetting issue on the original classes while achieve ideal performance on the new classes.

Furthermore, we explore the influence of the number of the added new classes. Specifically, on the CUB-200 dataset, we choose 100 classes and 25 classes as new categories. The results are reported in Table 5.4. Likewise, for the Stanford-Dogs dataset, we choose 60 new classes and 5 classes for incremental learning, whose results are reported in Table 5.5. For CUB-Brids, we observe that larger newly-added classes lead to heavier forgetting. For example, when only 25 new classes are used, the Recall@1 drops from 79.41% to 76.65%, compared to the one drops from 79.41% to 74.41% where 100 new classes are added. Note that the reference models are trained jointly on all classes and tested on the original and new classes separately. For Stanford-Dogs, we observe these two datasets share with similar trends that larger new coming classes lead to heavier forgetting issue. For the Stanford-Dogs dataset, when only 5 new classes are added, the Recall@1 drops from 80.67% to 79.75%, compared to the one drops from 80.67% to 76.67% when 60 new classes are added.

We visualize the feature distributions using t-SNE [195] under two experimental settings: with and without MMD loss in Figure 5.5, which demonstrate the MMD loss reduces the distance between distributions and effectiveness for mitigating the forgetting issue.

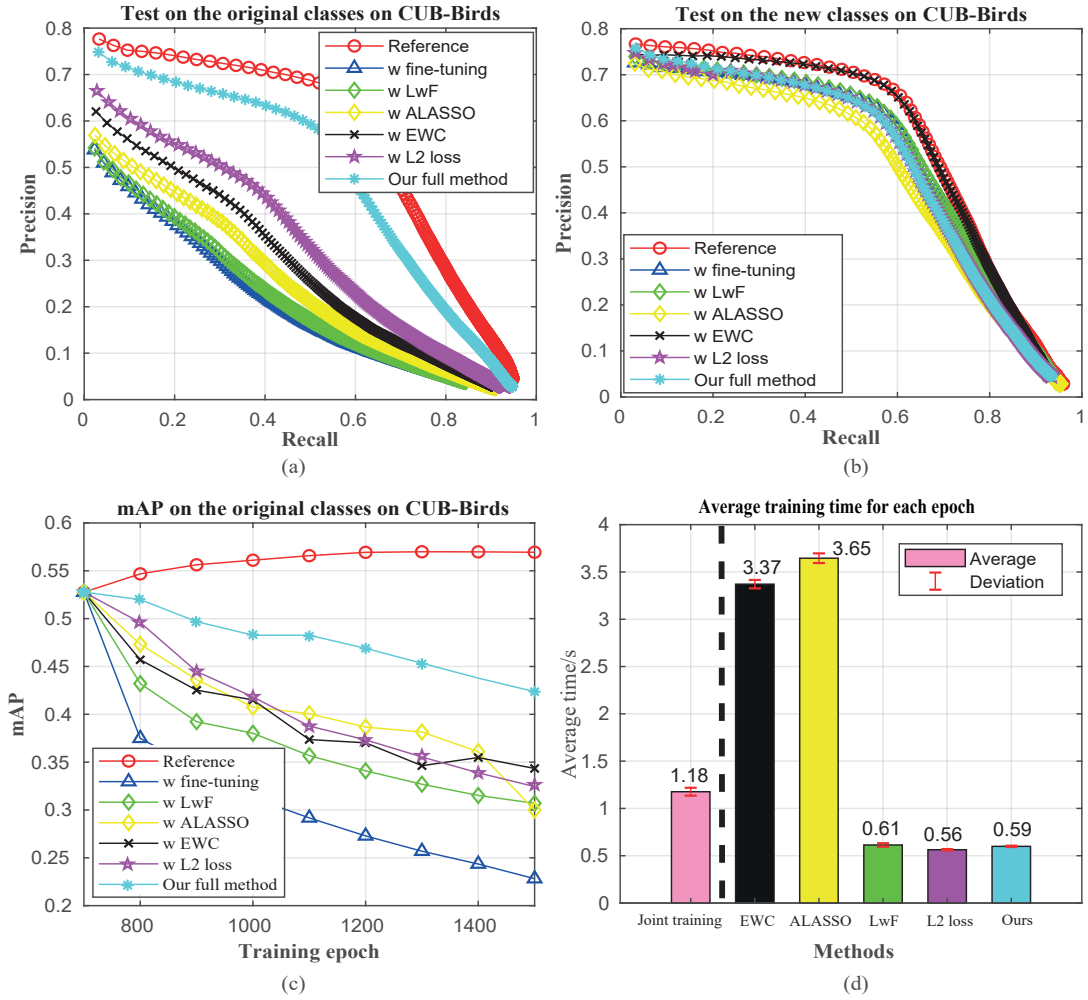


Figure 5.3: Performance evaluation on the CUB-200 dataset. (a)-(b) denote the PR curves tested on the original classes and new classes. (c) depicts the mAP results for different methods as the training proceeds. We only show the results tested on the original classes. (d) training time comparison during each epoch.

5.4.3 Multi-step incremental learning for FGIR

We split all new classes into 4 groups and added each sequentially. The training procedures are as follows: the initial model $\mathcal{A}$ is pre-trained on the original classes (1-100), and used as an initial model to train on newly-added classes (101-125) until

Table 5.4: Recall@K (%) on CUB-200 when 25 or 100 new classes are added at once. Correspondingly, † indicates the results are tested on different new classes.

Configurations	Original classes			New classes†		
Recall@K(%)	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-100) (initial model)	79.41	85.64	89.63	-	-	-
+$\mathcal{B}$(101-125) w Our method	76.65	83.47	88.86	73.13	82.31	88.44
+$\mathcal{B}$(101-200) w Our method	74.41	82.57	88.52	73.11	80.84	86.64
$\mathcal{A}$(1-125) (reference model)	77.84	83.94	87.80	79.25	85.54	91.96
$\mathcal{A}$(1-200) (reference model)	77.33	85.08	89.03	76.64	83.53	89.12

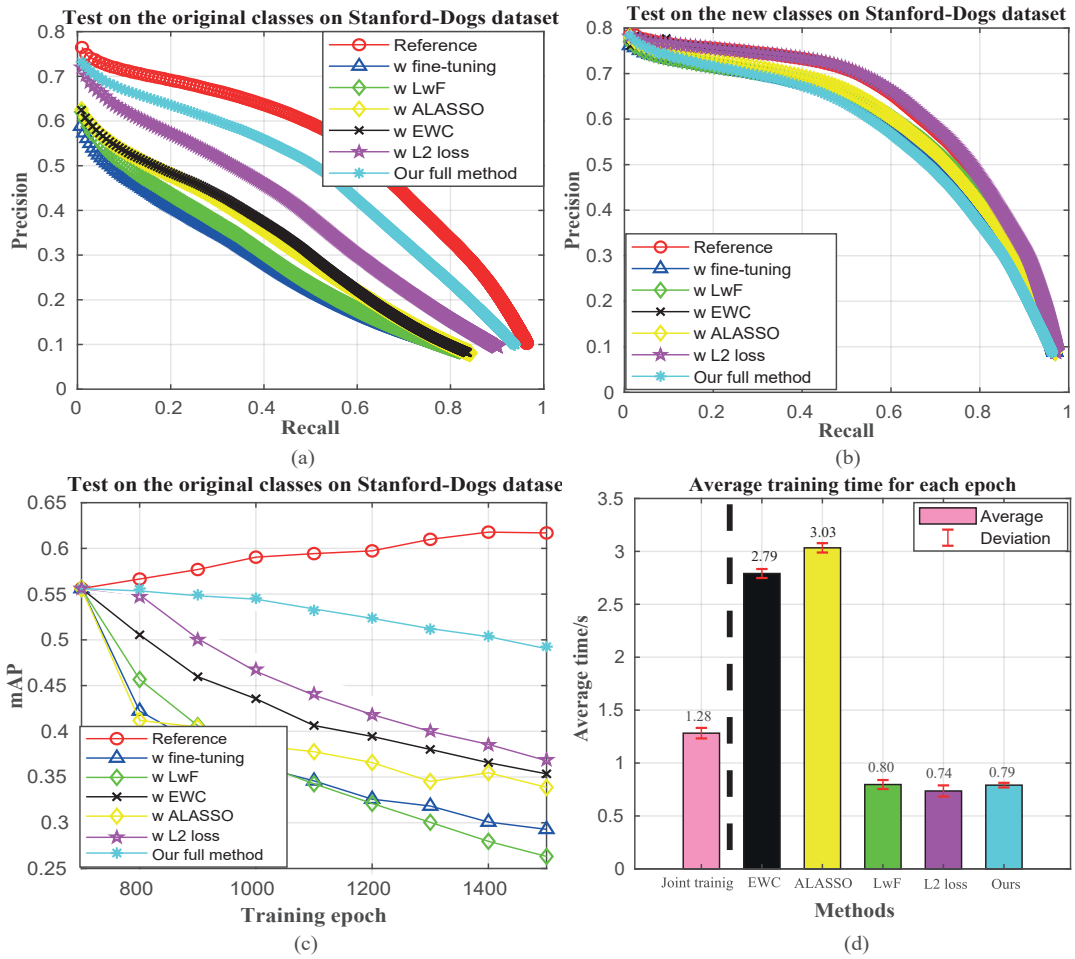
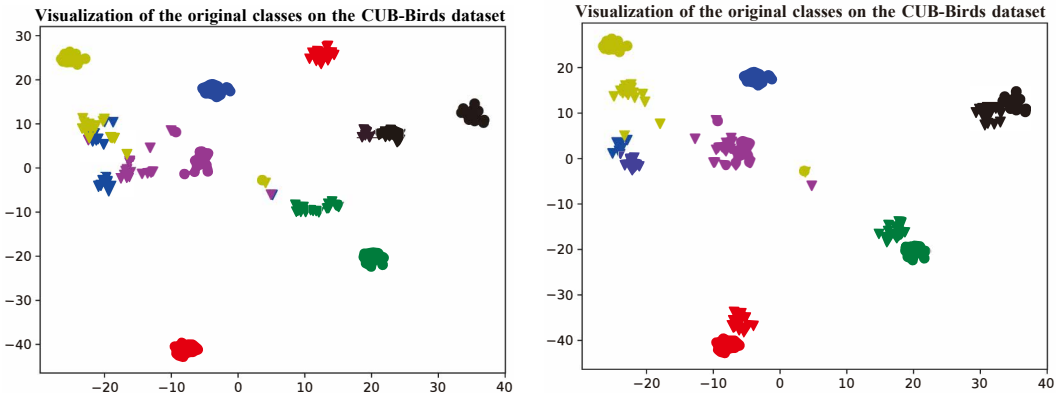


Figure 5.4: Performance evaluation on the Stanford-Dogs dataset. Figure (a)-(b) denote the precision-recall curves tested on the original classes and new classes. The larger the area under each curve, the better performance of the method. Figure (c) depicts the mAP results for different methods as the training proceeds. We only show the results tested on the original classes. Being closer to the reference curve (red one) indicates less performance degradation, *i.e.*, the method can maintain its previous performance on the original classes.

convergence to produce a new model $\mathcal{B}$(101-125). Afterwards, the newly-trained model $\mathcal{B}$(101-125) is used as an initial model to train on other new classes (126-150)

Table 5.5: Recall@K (%) on the Stanford-Dogs dataset when 5 or 60 new classes are added at once. Similar to the settings in Table 5.4, † indicates the results are tested on different new classes.

Configurations	Original classes			New classes†		
Recall@K(%)	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-60) (initial model)	80.67	87.27	92.20	-	-	-
+$\mathcal{B}$(61-65) w Our full method	79.75	87.23	91.92	97.45	98.55	99.27
+$\mathcal{B}$(61-120) w Our full method	76.67	85.10	91.14	81.88	88.98	93.36
$\mathcal{A}$(1-65) (reference model)	79.62	86.15	90.91	96.73	97.82	98.55
$\mathcal{A}$(1-120) (reference model)	79.29	86.86	91.61	82.57	88.75	93.13



(a) (b)

Figure 5.5: t-SNE visualization for feature distribution of 6 categories. The circle indicates the features from reference model, which has the same distribution in two cases. The triangle denotes the feature from models trained with/without MMD loss. (a): model trained without MMD loss; (b): model trained with MMD loss.

to produce $\mathcal{B}$(101-125)(126-150). This process is repeated until 4 groups of classes are added sequentially.

We compare to three representative methods (we choose EWC rather than ALASSO since EWC obtains higher performance on the CUB-200 dataset) and report the results in Table 5.9. The reference performances are achieved by jointly training all the classes, and then tested on each group (including the original classes). Overall, the model suffers from the catastrophic forgetting issue when sequentially training. However, our method achieves a minimal performance degradation. For instance, when 4 groups have been added, the model $\mathcal{B}$(101-125)(126-150)(151-175)(176-200) is tested on the original classes(1-100). The "L2 loss" algorithm Recall@1 drops 79.41%→67.37%→58.14%→53.86% → 45.85%, the average degradation is 8.39%. Our method Recall@1 drops 79.41%→76.65%→73.77%→70.47%→66.40%. The average performance degrades by 3.25%, which indicates that our method significantly mitigates the forgetting problem. Furthermore, our method has good performance on new classes, which are closer to the reference performance. When the model $\mathcal{B}$(101-125)(126-150)(151-175)(176-200) is tested on new classes (176-200), the results are achieved with Recall@1=85.21%, Recall@2=89.92% and Recall@4=93.28%, respectively, while the reference results are Recall@1=83.70%, Recall@2=90.25% and Recall@4=93.78%.

Similarly, we report the results on the Stanford-Dogs dataset in Table 5.10 when new classes are added sequentially. We observe similar trends as those for the CUB-200 dataset. Compared to the other two methods, the proposed method has ideal retrieval performance on the newly added classes and the original classes.

Table 5.6: Average top-1 accuracy of incremental learning for image classification on CIFAR-100 dataset [228].

Method	Number of the added new classes				
	20	40	60	80	100
L2 loss	77.3	47.5	40.5	36.6	32.8
EWC	77.3	60.5	50.9	43.3	39.5
LwF	77.3	62.5	52.9	46.2	41.0
Ours	77.3	64.6	55.8	49.2	43.3

Table 5.7: Ablation study for different components of loss function

Configurations	Original classes			New classes		
Recall@K(%)	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-100) (initial model)	79.41	85.64	89.63	-	-	-
+$\mathcal{B}$(101-200) w $L_{ce} + L_{triplet}$	53.90	64.56	73.56	76.18	82.56	87.39
+$\mathcal{B}$(101-200) w $L_{ce} + L_{triplet} + L_{dist}$	54.92	66.40	75.42	75.76	82.69	86.93
+$\mathcal{B}$(101-200) w $L_{ce} + L_{triplet} + L_{mmd}$	73.36	81.25	87.43	73.40	81.60	86.64
+$\mathcal{B}$(101-200) w $L_{ce} + L_{triplet} + L_{dist} + L_{mmd}$	74.41	82.57	88.52	73.11	80.84	86.64
$\mathcal{A}$(1-200) (reference model)	77.33	85.08	89.03	76.64	83.53	89.12

5.4.4 Validation with image classification

We evaluate the effectiveness of our method on CIFAR-100 [228] which is the popular benchmark for class-incremental learning in image classification. We split 100 classes into a sequence of 5 tasks, and each task includes 20 classes. In Table 5.6, the results indicate the average top-1 accuracy of the classes from seen tasks. In the last column, the test set evaluates the classes from all the five tasks. Note that, the 20 classes in the first task (the second column) achieve the same performance, as it has no incremental learning yet. We observe that our method outperforms other methods across the tasks. It suggests our method generalizes well to various applications. Notably, our improvement for image retrieval is more significant than that for image classification. The reason is that the proposed MMD loss is imposed on the feature representation, which largely benefits the metric learning for image retrieval. This also explains why our method is focused mainly on image retrieval.

Table 5.8: Sensitivity analysis of the hyper-parameters α, β. The better trade-off performance of the hyper-parameters are in bold face.

Configurations	Original classes			New classes		
Recall@K (%)	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-100) (initial model)	79.41	85.64	89.63	-	-	-
+$\mathcal{B}$(101-200) ($\alpha = 0.1, \beta = 0.1$)	56.53	66.31	75.59	77.52	83.82	88.15
+$\mathcal{B}$(101-200) ($\alpha = 0.1, \beta = 1$)	73.31	82.00	87.14	72.77	80.92	87.14
+$\mathcal{B}$(101-200) ($\alpha = 0.1, \beta = 10$)	79.58	85.76	90.47	49.50	61.51	70.59
+$\mathcal{B}$(101-200) ($\alpha = 1, \beta = 0.1$)	55.81	67.25	75.59	77.02	83.91	87.90
+$\mathcal{B}$(101-200) ($\alpha = 1, \beta = 1$)	**74.41**	**82.57**	**88.52**	**73.11**	**80.84**	**86.64**
+$\mathcal{B}$(101-200) ($\alpha = 1, \beta = 10$)	79.41	86.31	90.51	48.82	61.09	71.05
$\mathcal{A}$(1-200) (reference model)	77.33	85.08	89.03	76.64	83.53	89.12

5.4.5 Training time comparison

We compare the average training time on the CUB-200 dataset and the Stanford-Dogs dataset when 100 and 60 new classes are added at once. The results are shown in Figure 5.3(d) and 5.4(d), respectively. Note that all models in five methods are starting from the same initial model trained on the original 100 classes as initialization. The reference time is from joint training where the initial model is trained on all classes. The other four methods are incrementally learning the new classes only. We observe that our method saves more time by 50% as expected. EWC and ALASSO algorithms take more time than reference because the gradients computation during back-propagation process is time-consuming.

5.4.6 Components analysis

Ablation study. We have done an ablation study on the CUB-200 dataset when multiple classes are added at once. Note that the component "L_{ce}+$L_{triplet}$" comprises our baseline performance, thus we analyze the different loss items in Eq. 5.6. We can observe the influence of difference components for the original and new classes. The results are shown in Table 5.7.

Hyper-parameters sensitivity analysis. We explore the sensitivity of hyper-parameters α, β in Eq. 5.6, which affect significantly the trade-off performance. We conduct this experiment on the CUB-200 dataset. As shown in Table 5.8, we find that the incrementally-trained model is more sensitive to β than α. For instance, when α is set as 0.1, but β changes from 0.1 to 1, model $\mathcal{B}$ performs better on the new classes and significantly retains its previous performance. However, this obvious trend cannot be observed when β is set as 0.1, but α changes from 0.1 to 1 where the model $\mathcal{B}$ performs almost the same on the original and new classes. Finally, if α=β=1, the incrementally-trained model $\mathcal{B}$ keeps a better trade-off performance between the original and the new classes.

5.5 Chapter Conclusions

In this chapter, for the first time, we have exploited incremental learning for fine-grained image retrieval in several scenarios for increasing numbers of image categories when only images of new classes are used. To overcome the catastrophic forgetting, we adopted the distillation loss function to constrain the classifier in the original network and the incremental classifier in the adaptive network. Moreover, we introduced a regularization function, based on Maximum Mean Discrepancy (MMD), to minimize the discrepancy between features of newly added classes from the original and the adaptive network. Comprehensive and empirical experiments on two fine-grained datasets show the effectiveness of our method that is superior over existing methods. In the future, it is promising to investigate incremental learning between different fine-grained datasets for image retrieval.

Table 5.9: Recall@K (%) results on the CUB-200 dataset when new classes are added sequentially. "Added new (101-125)" indicates the first 25 classes (101-125) are used as the first part to train the network.

Configurations		Original (1-100)			Added new (101-125)			Added new (126-150)			Added new (151-175)			Added new (176-200)		
Recall@K(%)		K=1	K=2	K=4	K=1	K=2	K=4	K=1	K=2	K=4	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-100) (initial model)		79.41	85.64	89.63	-	-	-	-	-	-	-	-	-	-	-	-
with fine-tuning	+$\mathcal{B}$(101-125)	54.58	64.07	73.09	79.76	86.39	90.14	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)	41.53	53.18	63.55	61.56	73.64	84.01	70.33	80.00	88.00	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)	39.24	49.79	61.06	50.34	63.44	73.47	57.33	69.00	78.50	80.40	85.43	88.11	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)(176-200)	35.68	45.97	57.42	48.13	64.29	76.53	51.83	64.17	75.83	66.16	75.54	80.90	83.70	89.75	92.27
LwF algorithm [212]	+$\mathcal{B}$(101-125)	57.50	68.05	75.68	79.59	85.88	88.95	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)	42.46	54.03	64.66	62.59	74.83	82.31	70.17	79.67	86.00	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)	40.21	51.57	61.27	47.79	63.10	75.68	56.83	67.33	78.17	81.57	87.27	90.79	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)(176-200)	33.31	44.75	55.38	49.83	63.78	75.85	48.00	60.33	72.33	67.17	75.88	82.91	83.70	89.41	92.94
EWC algorithm [213]	+$\mathcal{B}$(101-125)	61.23	70.85	80.04	80.95	86.39	90.82	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)	46.65	56.40	67.54	65.48	77.72	84.01	72.33	80.67	86.67	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)	43.60	54.79	64.70	61.50	72.45	80.44	66.50	75.50	82.67	81.08	85.26	87.77	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)(176-200)	36.82	47.54	59.66	57.99	67.01	76.87	50.67	64.67	77.67	64.15	74.87	81.24	82.02	86.39	90.42
L2 loss algorithm [219]	+$\mathcal{B}$(101-125)	67.37	76.27	83.31	80.61	85.54	89.46	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)	58.14	68.31	76.78	72.11	80.44	87.41	73.33	82.17	88.67	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)	53.86	62.03	71.91	60.37	71.43	80.27	66.33	76.67	84.67	81.24	87.27	90.95	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)(176-200)	45.85	56.61	67.75	57.65	71.77	80.95	59.33	70.50	79.13	73.70	83.08	88.94	84.20	89.24	92.10
Our method	+$\mathcal{B}$(101-125)	76.65	83.47	88.86	73.13	82.31	88.44	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)	73.77	81.36	87.80	74.32	83.33	89.29	74.50	83.00	87.83	-	-	-	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)	70.47	78.77	85.97	70.41	80.78	88.78	72.00	79.17	86.83	78.89	86.77	90.26	-	-	-
	+$\mathcal{B}$(101-125)(126-150)(151-175)(176-200)	66.40	75.93	83.14	70.07	80.27	86.22	69.00	78.33	85.50	73.87	83.92	88.78	85.21	89.92	93.28
$\mathcal{A}$(1-200) (reference model)		77.33	85.08	89.03	76.87	84.86	90.48	73.00	80.00	87.67	83.25	88.94	92.29	83.70	90.25	93.78

Table 5.10: Recall@K (%) results on the Stanford-Dogs dataset when new classes are added sequentially. "Added new (61-75)" indicates we use first 15 classes (61-75) as the first incremental part to train the network.

Configurations		Original (1-60)			Added new (61-75)			Added new (76-90)			Added new (91-105)			Added new (106-120)		
Recall@K(%)		K=1	K=2	K=4	K=1	K=2	K=4	K=1	K=2	K=4	K=1	K=2	K=4	K=1	K=2	K=4
$\mathcal{A}$(1-60) (initial model)		80.67	87.27	92.20	-	-	-	-	-	-	-	-	-	-	-	-
with fine-tuning	+$\mathcal{B}$(61-75)	52.61	65.49	75.17	88.97	92.86	94.74	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)	40.81	53.32	64.78	69.92	79.95	86.72	79.74	88.17	93.21	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)	39.39	52.25	64.12	66.54	77.69	84.46	68.35	80.83	87.73	78.76	87.17	91.95	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)(106-120)	36.79	48.38	60.87	58.02	69.17	79.32	62.21	77.66	86.31	67.79	76.90	85.31	82.81	90.62	92.83
LwF algorithm [212]	+$\mathcal{B}$(61-75)	50.87	62.76	73.40	88.35	92.48	94.36	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)	42.06	53.62	65.10	71.18	82.46	88.10	77.33	87.19	92.44	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)	37.58	50.48	63.00	60.65	73.68	82.33	70.10	81.38	87.40	80.62	87.17	92.48	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)(106-120)	38.46	50.63	62.59	59.90	72.68	81.20	63.86	77.22	85.54	68.41	77.70	85.66	81.34	88.69	92.74
EWC algorithm [213]	+$\mathcal{B}$(61-75)	55.84	67.64	77.57	89.10	92.61	94.36	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)	45.32	58.29	68.85	79.82	85.21	90.35	81.38	88.72	93.32	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)	37.60	49.71	61.88	67.04	79.04	85.71	67.47	79.52	88.39	81.33	86.99	91.15	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)(106-120)	34.08	45.60	58.40	63.53	75.19	83.71	63.42	77.66	86.31	70.00	79.12	85.84	81.99	87.96	92.10
L2 loss algorithm [219]	+$\mathcal{B}$(61-75)	65.30	75.83	83.51	90.85	94.74	95.61	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)	55.97	67.36	77.04	84.46	90.73	92.73	80.94	89.38	93.54	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)	50.38	62.87	73.64	72.68	82.21	88.85	75.68	84.67	91.57	83.72	90.00	93.81	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)(106-120)	46.01	58.74	69.64	67.79	78.07	85.71	72.51	84.45	90.03	74.87	83.98	89.82	86.21	91.36	94.39
Our method	+$\mathcal{B}$(61-75)	76.07	84.88	90.11	91.85	95.36	96.87	-	-	-	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)	70.67	80.48	87.87	89.10	93.11	95.99	84.23	89.92	93.43	-	-	-	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)	67.75	79.17	86.45	86.09	91.98	95.49	81.60	90.25	93.76	84.25	89.03	93.45	-	-	-
	+$\mathcal{B}$(61-75)(76-90)(91-105)(106-120)	65.47	76.52	85.08	83.21	89.35	93.73	79.19	87.84	93.32	82.83	89.20	94.42	87.13	92.10	94.39
$\mathcal{A}$(1-120) (reference model)		79.29	86.86	91.61	92.61	94.99	96.37	82.48	90.80	93.76	83.72	91.33	95.58	86.12	93.11	95.96