Exploring deep learning for intelligent image retrieval

Chen, W.

Chapter 6

Feature Estimations based Correlation Distillation for Incremental Image Retrieval

In Chapter 5, we explored incremental learning for fine-grained image retrieval in which only the penultimate model is used for transferring previously learned knowledge. As incremental learning proceeds, each training session produces a specific model. Saving this stream of models will be memory-consuming. This raises a question that how to utilize the stream of models in incremental learning to transfer more previously learned information when learning on the current new data. We investigate this question by proposing a feature estimation method. Similar to the knowledge distillation framework in Chapter 5, we distill semantic correlations knowledge among the representations extracted from the new data only so as to regularize the parameters updates. In particular, for the case of learning multiple tasks sequentially, aside from the correlations distilled from the penultimate model, we estimate the representations for all prior models and further their semantic correlations by using the representations extracted from the new data. To this end, the estimated correlations are used as an additional regularization and further prevent catastrophic forgetting over all previous tasks, and it is unnecessary to save the stream of models trained on these tasks.

Keywords

Incremental learning, Fine-grained image retrieval, Correlations distillation, Feature estimation

This chapter is based on the following publication [38]:

- Chen, W., Liu, Y., Pu, N., Wang, W., Liu L., and Lew, M.S., "Feature Estimations based Correlation Distillation for Incremental Image Retrieval." IEEE Transactions on Multimedia, 2021.

6.1 Introduction

Learning is a life-long process for human beings so that we can learn continuously, devoid of forgetting previously acquired knowledge. However, this is not the case for deep neural networks, which suffer from the catastrophic forgetting problem [36]. Deep networks have been trained and validated for image retrieval on stationary datasets. As new data increase over time, the networks trained on the stationary datasets cannot be suited well for the non-stationary scenario.

The main challenge is to make the trained model adapt to new data without losing the knowledge on the seen data. Most conventional solutions for tackling this challenge suffer from obvious limitations. For example, joint training achieves optimal retrieval performance on old and new data, while it requires the presence of all the data. This is hard to meet for several scenarios where legacy data are unrecorded due to privacy issues or simply too cumbersome to collect old data. Moreover, retraining old data may lead to an imbalance issue between the quantity of old data and that of new data [229, 230].

Two incremental learning methods are developed to tackle the above limitations. First, the rehearsal based method utilizes generative adversarial nets to synthesize samples *w.r.t.* previous data distributions [231]. This method faces the difficulty of generating images with complex semantics. Second, the regularization based methods can either focus on network parameters or output activations. Parameters-based regularization methods estimate the parameter importance of previous tasks, then penalizes the drastic updates of these parameters when the model is learning a new task. Activation-based regularization methods, relying on the teacher-student framework, constrain the teacher model and the student model have similar outputs. The regularization methods have been explored for tasks such as image classification [229, 230, 232], but are less-explored for image retrieval. Recently, Parshotam *et al.* [233] regularize the representations via a normalized cross-entropy loss, training with metric learning for vehicle identification and retrieval. Chen *et al.* [37] propose regularizing both the representations and probabilities via the teacher-student framework for fine-grained image retrieval (FGIR) [234]. As depicted in Figure 6.1(a), they only use the penultimate model to transfer previously learned knowledge on old tasks.

For the case where new tasks are added sequentially, which is referred to multi-task incremental learning, only distilling on the penultimate model is insufficient to reduce forgetting on all previous tasks [235]. In fact, transferring additional knowledge learned on these tasks, *i.e.* via multi-model distillation tackles this insufficiency, as shown in Figure 6.1(b). In multi-task incremental learning, a stream of deep models is produced as new tasks are added continuously. However, it becomes too cumbersome and inefficient to store these models. Therefore, an arising question is that *how to use the model stream, not only the penultimate model, for knowledge distillation?*

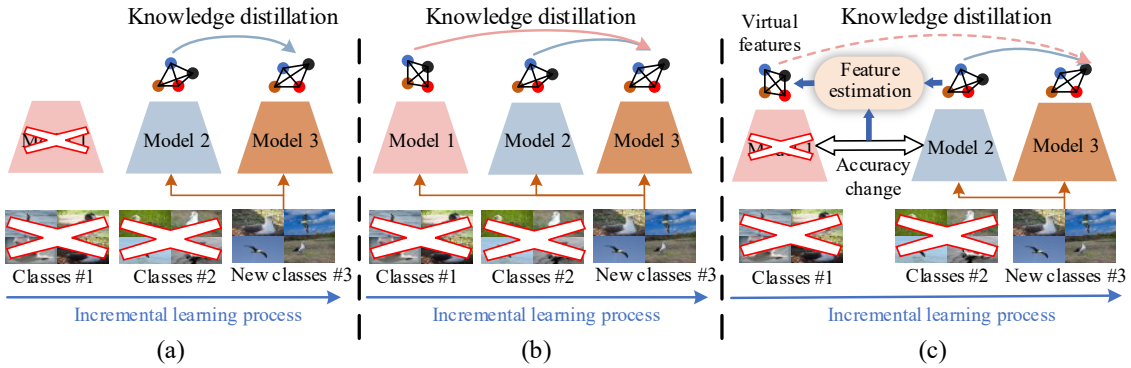

Figure 6.1: Comparison of three knowledge distillation methods. We depict three steps of distillation. (a) Single-model distillation method only stores and uses the penultimate model; (b) Multi-model distillation method has to store all old models and distills from them more knowledge devoid of forgetting; (c) Our method only stores the penultimate model while can accumulate previous knowledge learned at each model through feature estimations.

Few researchers address this problem in incremental tasks. Recently, a multi-model and multi-level knowledge distillation strategy is presented for incremental image classification [235]. However, the snapshots of all previous models still need to be saved and depend on network pruning methods to reconstruct.

In this chapter, we face the above question to improve deep model's continuous retrieval ability. Semantic correlations of features are transferred as knowledge from a teacher model to a student model when new data are used only. For multi-task incremental learning, the model stream trained on preceding tasks is unnecessarily saved. Instead, we estimate representations for these models and further their semantic correlations, using the features extracted from the current new task, as shown in Figure 6.1(c).

6.2 Related Work

Incremental image retrieval. Incremental learning can be categorized into architectural methods [230, 232], rehearsal methods [220, 231], and regularization methods [227, 235]. Most of them are used for image classification, regularizing the classification probabilities. Recently, incremental retrieval have been explored. CIHR [236] was proposed to deal with the concept drift issue for hashing retrieval in non-stationary environments. However, the selected images from previous training sessions are combined with new images to train hash tables. DIHN [218] is explored for incremental hashing retrieval where old data are used as a query set. Fine-grained incremental image retrieval is studied with only using new data [37]. However, knowledge is only transferred from the penultimate model, causing the insufficiency to remember previous knowledge when performing multi-task incremental learning. In this work, we further distill additional knowledge from the model stream

via a simple yet effective feature estimation method when only using new data in each incremental session.

Knowledge distillation. Knowledge can be distilled from the output of either the final classifier or the intermediate layers, relying on the teacher-student structures [237]. It is realized by characterizing the differences between the teacher model and the student model through metrics such as L1 distance [216], L2 distance [217], Gramian matrix [238], and KL-divergence [214]. For more details about knowledge distillation, we refer readers to a recent survey [237]. Knowledge distillation provides an effective way to retain the learned knowledge devoid of forgetting by one-teacher or a multi-teacher frameworks [235, 239]. For example, Zhou *et al.* [235] introduce using all previous models to transfer multi-level knowledge to train current new tasks. To avoid a great memory storage requirement, they prune previous models to get several "necessary" parameters during each training session.

Correlation learning has been used for multi-modal tasks to explore the relevance between different layers or data samples [240, 241, 242, 243, 244]. It focuses on the relations between feature representations rather than the features themselves. These relations enable models to explore rich contextual information of images such as [243] where three-level of correlations are integrated for optimal feature learning. Correlation learning can be combined into knowledge distillation. For example, Peng *et al.* [244] use a symmetric adjacency matrix to encode a knowledge graph with category correlations and transfer them via a semantic-visual mapping network. Similarity between activations of input pairs can also be extracted as knowledge to transfer into the student model [245]. The successful applications of correlation learning for knowledge distillation encourage its exploration for incremental learning tasks.

6.3 Correlations Distillation for Incremental Image Retrieval

6.3.1 Problem formulation

Given a dataset $\mathcal{D} = \{(\mathbf{X}^c, y^c)|c = 1, 2, \cdots, n\}$ with n classes, each of which c includes different amount of images $|\mathbf{X}^c|$ and they share the same ground-truth label y^c. The label is used to select a positive x_p and a negative x_n images for an anchor image x_a in each training iteration. A deep network $f(\cdot, \boldsymbol{\theta})$ learns representations $\boldsymbol{F} = f(\boldsymbol{X}, \boldsymbol{\theta})$ under the constraint of the triplet loss using hard sampling strategy, whose goal is to push away the distance $D(x_a, x_n) = ||f(x_a; \boldsymbol{\theta}) - f(x_n; \boldsymbol{\theta})||_2^2$ between x_n and x_a by a margin $\delta > 0$ compared to $D(x_a, x_p)$. Namely,

$$||f(x_a; \boldsymbol{\theta}) - f(x_p; \boldsymbol{\theta})||_2^2 + \delta < ||f(x_a; \boldsymbol{\theta}) - f(x_n; \boldsymbol{\theta})||_2^2 \tag{6.1}$$

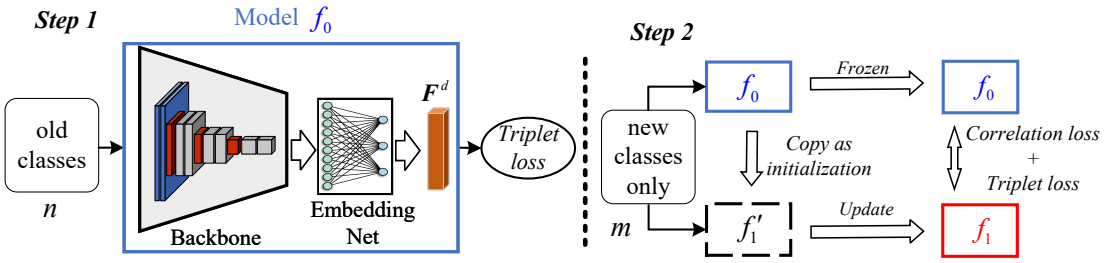

Figure 6.2: One-task incremental learning includes two training steps. **Step 1**: a model f_0 is well trained in advance on the n old classes using ranking loss only. **Step 2**: the well-trained model f_0 is frozen as a teacher network. Meanwhile, the parameters of the Backbone and the Embedding Net included in this model f_0 are copied as initialization for a temporary model $f_1^{'}$, which is updated to the final model f_1 under the constraints of correlation loss and triplet loss. At Step 2, only the m new classes are used for training.

Before incremental training, the network is well trained on the n old classes, converging at old parameters $\boldsymbol{\theta}_o$, *i.e.*,

$$\boldsymbol{\theta}_o = \underset{\boldsymbol{\theta}}{argmin}\ L_{triplet}(f_0(\mathbf{X}^c; \boldsymbol{\theta})) \tag{6.2}$$

where $L_{triplet}(x_a, x_p, x_n) = [\delta + D(x_a, x_p) - D(x_a, x_n)]_+$, as defined in Eq. 6.1. To train network f_0 incrementally, new data from m classes $\{(\mathbf{X}^{c'}, \mathbf{Y}^{c'})\}$ where $c' \in (n+1, n+2, ..., n+m)$ are added ($\{\mathbf{X}^c\} \cap \{\mathbf{X}^{c'}\} = \emptyset$) at once or sequentially, corresponding to one-task and multi-task cases, respectively.

The one-task case is depicted in Figure 6.2. At the start of training on m new classes, f_0 is copied into two copies. One is frozen as a teacher net, and another is used as a temporary initialization $f_1^{'}$ for further training ($\boldsymbol{\theta}_o = \boldsymbol{\theta}_n^{'}$, including the parameters in the Backbone and Embedding Net). We *only* use the m new classes to train to obtain f_1. Thus, the core issue of the one-task case is to make the model f_1 with new parameters $\boldsymbol{\theta}_n$ maintain a stable performance on the n old classes and achieve competitive accuracy on the m new classes. Formally, the overall objective for this scenario is:

$$L(\mathbf{X}^{c'}; \boldsymbol{\theta}_o; \boldsymbol{\theta}_n) = \underbrace{\lambda_1 L_{triplet}(\mathbf{X}^{c'}; \boldsymbol{\theta}_n)}_{\text{for plasticity}} + \underbrace{\lambda_2 L_{corr}(\mathbf{X}^{c'}; \boldsymbol{\theta}_o; \boldsymbol{\theta}_n)}_{\text{for stability}} \tag{6.3}$$

where $L_{triplet}$ makes the model perform well on new tasks while L_{corr} is the correlation loss to stabilize prior performance. $\boldsymbol{\theta}_o$ and $\boldsymbol{\theta}_n$ are the parameters for old tasks and new tasks, respectively. λ_1 and λ_2 are the plasticity and stability hyper-parameters, which tune the influence of two loss terms.

6.3.2 Correlations distillation for one-task scenario

As shown in Figure 6.2, the model $f_1^{'}$ serves as a to-be-trained student net. For the one-task incremental scenario, we propose to distill the semantic correlations as knowledge.

Specifically, the features with dimension d from the teacher model f_0 are formulated as $\boldsymbol{F}_o = f_0(\boldsymbol{X}^{c'}, \boldsymbol{\theta}_o) \in \mathbb{R}^{N\times d}$, and that from the student model f_1 are $\boldsymbol{F}_n = f_1(\boldsymbol{X}^{c'}, \boldsymbol{\theta}_n) \in \mathbb{R}^{N\times d}$. Based on the fact that semantically similar inputs produce similar patterns in a trained network [245]. Therefore, a Gram matrix with a kernel function $\mathcal{K}(\cdot)$ for $\boldsymbol{F}_o$ and $\boldsymbol{F}_n$ is defined:

$$G_o^{(i,j)} = \mathcal{K}(F_o^i, F_o^j);\ G_n^{(i,j)} = \mathcal{K}(F_n^i, F_n^j) \tag{6.4}$$

Here, we further define the function $\mathcal{K}(\cdot)$ as inner product, *i.e.*, $\mathcal{K}(F^i, F^j) = <F^i, F^j>$. Each entry (i, j) in $\boldsymbol{G} \in \mathbb{R}^{N\times N}$ represents the correlations of the same activation $(i = j)$ or these between different activations $(i \neq j)$. To compare the difference between $\boldsymbol{G}_o$ and $\boldsymbol{G}_n$, we first normalize these matrices with Softmax function $\sigma(\cdot)$, and then use KL-divergence to formulate a correlation loss L_{corr}.

$$L_{corr} = \frac{1}{N}\sum KL\big(\sigma(\boldsymbol{G}_o), \sigma(\boldsymbol{G}_n)\big) \tag{6.5}$$

6.3.3 Feature estimation for multi-task scenario

Compared to the one-task setting, the multi-task scenario is more complex where all m new classes are divided into t groups: $\boldsymbol{X}_0^{c'}$,..., $\boldsymbol{X}_t^{c'}$. For clarity, we illustrate its training process in Figure 6.3. As more new classes added sequentially, the model, correspondingly, evolutes from the initial model f_0 to the current one f_t. In practice, it may be difficult to save the stream of models. For this limit, we only save the model trained on the penultimate task $t-1$ when proceeding current task t for t^{th} new classes $\boldsymbol{X}_t^{c'}$. For example, when training on the 3^{rd} group of new classes (task t=3), the knowledge is distilled only from the penultimate models f_2, while the previous models f_0 and f_1 are not saved. Due to the lack of previous models, it causes two drawbacks: (1) the knowledge is distilled only from the penultimate model f_{t-1} to the model on the current task t, and (2) the trained model f_t may forget more on old tasks prior to $t-1$. Therefore, it is natural to raise a question that how to utilize these unsaved models trained prior to the penultimate task $t-1$ for transferring additional knowledge to supervise the training of current task t.

Hereafter, for better understanding, we introduce the multi-task scenario by defining an adaptive model f_t for the current task t, a frozen model f_{t-1} trained on penultimate task $t-1$, and unsaved models $f_{t-2}, ..., f_0$ for earlier tasks $t-2, ..., 0$, as shown in Figure 6.4. Since the frozen model f_{t-1} is initialized from the previous unsaved model f_{t-2} at the start of training on task $t-1$, the feature distributions

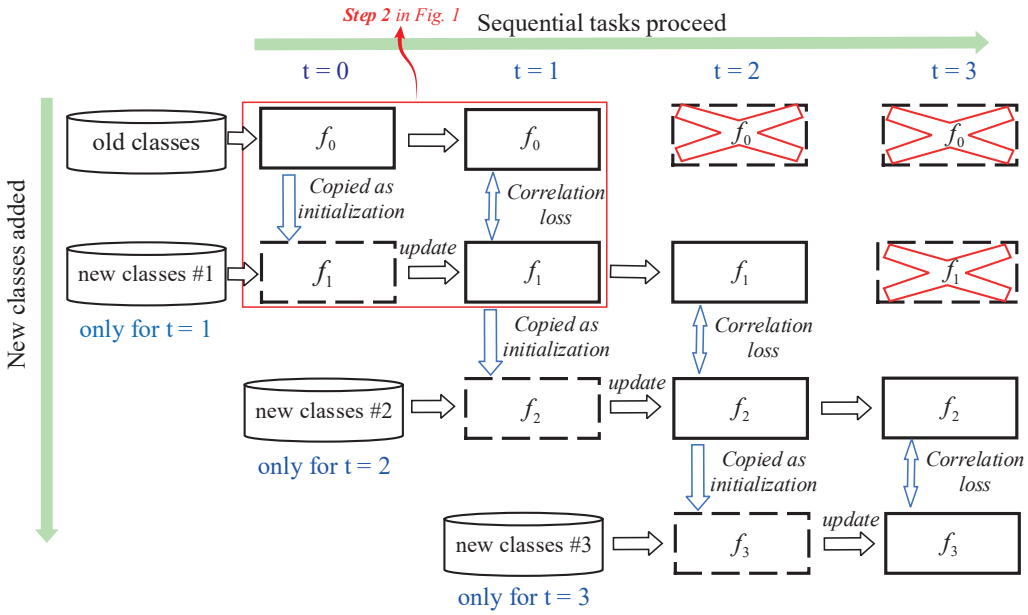

Figure 6.3: Illustration of multi-task incremental learning when three groups of new classes are added sequentially. For each round when new classes are added, the model trained on a previous task is frozen its parameters as a teacher net and is also copied as initializations of the model for new classes. Each round can be viewed as one-task incremental learning. For simplicity, the triplet loss is ignored.

of these two models have some inherent relations, which can be reflected through their accuracy (*e.g.*, mAP). This accuracy evolution along with training the models stream gives a hint for feature estimation.

a. Accuracy drops and accuracy gains

We propose a simple yet effective method to estimate the feature distributions for all unsaved models, which serve as an additional regularization term for training on current task t ($t \geq 2$). For this purpose, we first focus on the accuracy change during training from task $t-2$ to task $t-1$. Parameters of the penultimate model f_{t-1} are copied from those of the model f_{t-2}. Before training on task $t-1$, the accuracy on its old tasks and the new classes $\boldsymbol{X}_{t-1}^{c'}$ are recorded as Acc_o^b and Acc_n^b, respectively. Naturally, Acc_n^b is far from accurate since the penultimate model f_{t-1} is not trained specifically for new data. After training on task $t-1$, the accuracy on these old tasks and new classes $\boldsymbol{X}_{t-1}^{c'}$ are recorded as Acc_o^a and Acc_n^a, respectively. Intuitively, the model f_{t-1} acquires new knowledge on new classes $\boldsymbol{X}_{t-1}^{c'}$, and the accuracy increases from Acc_n^b to Acc_n^a (*i.e.*, accuracy gains). In contrast, model f_{t-1} may degrade accuracy from Acc_o^b to Acc_o^a (*i.e.*, accuracy drops) because this model is driven towards the new data.

The accuracy drops and accuracy gains, related to the stability-plasticity trade-off, are criteria that correspond to old tasks and new tasks, respectively. For instance, if a model has larger stability on previous tasks, both the accuracy drops and accuracy gains are small. In contrast, if the stability is too weak, the model suffers obvious

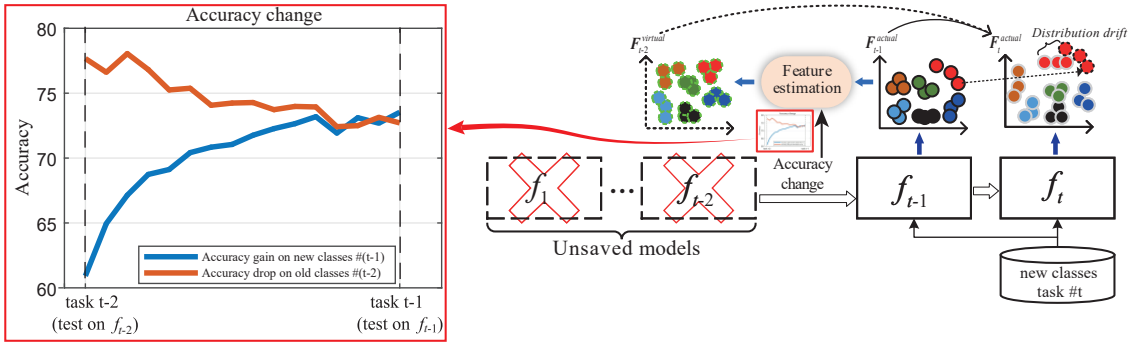

Figure 6.4: Illustration of feature estimation when performing the task t. The virtual feature distribution of unsaved model f_{t-2} can be estimated by that of frozen model f_{t-1} under multi-task incremental learning.

accuracy drops and forgetting on previous tasks. Inspired by [246], we define the accuracy changes using the accuracy drops and accuracy gains:

$$\alpha_{drop} = \frac{(Acc_o^a - Acc_o^b)}{Acc_o^b}, \alpha_{gain} = \frac{(Acc_n^a - Acc_n^b)}{Acc_n^b} \tag{6.6}$$

As the training proceeds from task $t-2$ to task $t-1$, their accuracy changes on old classes (the brown-color line in Figure 6.4) and new classes (the black-color line). Rather than saving these models, we only record their accuracy drops $\alpha_{drop}|_{(t-2)\to(t-1)}$ and accuracy gains $\alpha_{gain}|_{(t-2)\to(t-1)}$, which are meta-data of these models and provide implicit information to estimate the feature distribution drifts. Here, the subscript "$(t-2) \to (t-1)$" means the knowledge is distilled from task $t-2$ to penultimate task $t-1$.

b. Distribution drifts estimation

Estimating feature distribution drifts was explored in [247] where the attribute vectors are learned based on the source set and target set, then the learned vectors are used to estimate new features. In this work, we estimate feature drifts via the change of model accuracy. We only save the penultimate model f_{t-1} when training on current task t, see Figure 6.4. The recorded accuracy change from model f_{t-2} to model f_{t-1} has been reflected through the drifts of their feature distributions. Based on this, we use the accuracy change (α_{drop}, α_{gain}) and the available features from the model f_{t-1} to estimate the feature drifts which are used to further compute virtual features for model f_{t-2}. To be specific, when feeding t^{th} group of new classes $\boldsymbol{X}_t^{c'}$ into the model f_{t-1} and the adaptive model f_t, we obtain their corresponding actual features $\boldsymbol{F}_{t-1}^{actual} = f_{t-1}(\boldsymbol{X}_t^{c'})$ and $\boldsymbol{F}_t^{actual} = f_t(\boldsymbol{X}_t^{c'})$. Since the accuracy drops and accuracy gains from model f_{t-2} to model f_{t-1} have been obtained, we estimate their feature distribution drifts using a simple yet effective method:

$$\begin{aligned} &\boldsymbol{\Delta}|_{(t-2)\rightarrow(t-1)} \approx \boldsymbol{\alpha} \cdot \boldsymbol{F}_{t-1}^{actual} \\ &s.t.\ \boldsymbol{\alpha} = Cat(\alpha_1, ..., \alpha_i, ..., \alpha_N),\ \alpha_i \in \mathbb{R}^d, \boldsymbol{\alpha} \in \mathbb{R}^{N \times d} \\ &\alpha_i \sim U(\,\alpha_{drop}|_{(t-2)\rightarrow(t-1)},\, \alpha_{gain}|_{(t-2)\rightarrow(t-1)}) \end{aligned} \tag{6.7}$$

where $Cat(\cdot)$ means vector concatenation operation. Each raw vector α_i is randomly sampled from the uniform distribution $U(\cdot, \cdot)$ according to α_{drop} and α_{gain}. Thereby, $\boldsymbol{\alpha}$ has the same dimension with the features $\boldsymbol{F}$. In theory, the expectation of each sampling in $\boldsymbol{\alpha}$ is close to $0.5 \times (\alpha_{drop} + \alpha_{gain})$.

It is assumed that the features change uniformly during sequential training and the changes can be reflected through the accuracy drops and accuracy gains. With this hypothesis, the feature drifts $\boldsymbol{\Delta}|_{(t-2)\rightarrow(t-1)}$ can be evaluated according to the actual features $\boldsymbol{F}_{t-1}^{actual}$. With the feature drifts, inspired by [247], the virtual feature distributions for unsaved model f_{t-2} are estimated:

$$\boldsymbol{F}_{t-2}^{virtual} = \boldsymbol{F}_{t-1}^{actual} + k\,\boldsymbol{\Delta}|_{(t-2)\rightarrow(t-1)} \tag{6.8}$$

where k is a scaling factor, we set $k = 1$. The reason why we can estimate the virtual features $\boldsymbol{F}_{t-2}^{virtual}$ from $\boldsymbol{F}_{t-1}^{actual}$ is because the parameters of model f_{t-1} are initialized from model f_{t-2} at the start of training f_{t-1}.

Similarly, we can further approximate the virtual feature $\boldsymbol{F}_{t-3}^{virtual}$ for model f_{t-3} according to the already-estimated $\boldsymbol{F}_{t-2}^{virtual}$, its accuracy drops $\alpha_{drop}|_{(t-3)\rightarrow(t-2)}$ and accuracy gains $\alpha_{gain}|_{(t-3)\rightarrow(t-2)}$ from task $t-3$ to task $t-2$. Normally, with a recursive scheme, the virtual features of all previous unsaved models can be estimated using their recorded accuracy drops, accuracy gains, and already-estimated virtual features. Finally, the features of first model f_0 are estimated as:

$$\begin{aligned} \boldsymbol{F}_0^{virtual} = &(1 + k\,\boldsymbol{\alpha}|_{(t-2)\rightarrow(t-1)})(1 + k\,\boldsymbol{\alpha}|_{(t-3)\rightarrow(t-2)}) \\ &...(1 + k\,\boldsymbol{\alpha}|_{(0)\rightarrow(1)})\boldsymbol{F}_{t-1}^{actual} \end{aligned} \tag{6.9}$$

c. Importance for estimated features

The estimated features for all previous unsaved models serve as additional regularization terms. Thus, more Gram matrices $\boldsymbol{G}^{virtual}$ are computed based on these estimated features, as illustrated in Figure 6.5. To this end, the additional correlation loss, such as $L_{corr}^{(t-2)\rightarrow t}$, based on the estimated features is formulated as:

$$L_{corr}^{(t-2)\rightarrow t} = \frac{1}{N} \sum \Big(KL\big(\sigma(\boldsymbol{G}_{t-2}^{virtual}), \sigma(\boldsymbol{G}_{t}^{actual})\big)\Big) \tag{6.10}$$

When more new classes are added sequentially, more Gram matrices are computed through the recursively-estimated features. However, these Gram matrices cannot be treated identically when used for regularizing the current task t since the accumulated errors may make the recursively estimated features more and more unreliable. For this limitation, the estimations for earlier tasks are assigned with a smaller importance. Naturally, the importance is related to the indices of old tasks. Finally, we formulate the correlation loss terms with different importance factors:

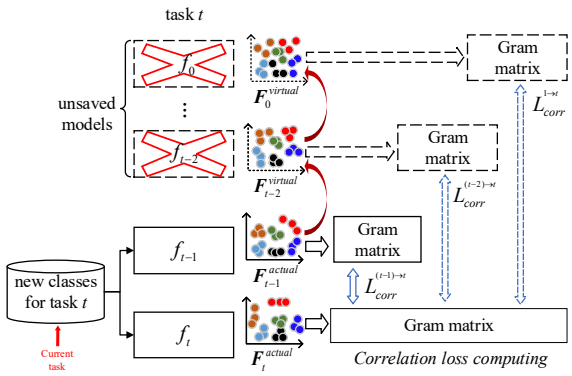

Figure 6.5: Feature estimations based knowledge distillation. The red arrows denote feature estimation process. The dash arrows indicate the features are virtually estimated from the actual features. For instance, the superscript "$(t-2) \rightarrow t$" refers to the Gram matrix of task $(t-2)$ is used as supervision for training current task t, which is formally defined in Eq. 6.5.

$$L_{corr} = L_{corr}^{(t-1)\rightarrow t} + \underbrace{\frac{1}{(t-1)} L_{corr}^{(t-2)\rightarrow t} + \frac{0.1}{(t-2)} L_{corr}^{(t-3)\rightarrow t} + ... + \frac{(0.1)^{t-2}}{1} L_{corr}^{1\rightarrow t}}_{\text{Feature estimation for prior sequential tasks(t}\geq 2)} \tag{6.11}$$

For one-task incremental scenario (t=1), Eq. 6.11 can be re-written as Eq. 6.5. If more tasks are performed ($t \geq 2$), each semantic correlation loss based on the estimated virtual features are constrained with importance factors ($\frac{1}{(t-1)}$, $\frac{0.1}{(t-2)}$,...). Substituting the term Eq. 6.11 into Eq. 6.3, we obtain the overall objective function for incremental FGIR.

6.4 Experiments

6.4.1 Datasets and experimental setup

We evaluate the method on two datasets: Caltech-UCSD Birds-200 (CUB-200) [224] and Stanford-Dogs-120 (Dogs-120) [223]. We choose 60% images from each category as training sets and 40% as testing sets. Afterwards, we split the first 100 categories (60 for the Dogs dataset) as the old classes (*i.e.*, n=100 or 60) and the remaining 100 (60 for the Dogs dataset) categories as new classes (*i.e.*, m=100 or 60). For the multi-task case, these new classes are divided into several groups evenly. All splits are in the order of official classes. In the following text, we use the class index of each dataset to denote a group of new classes. For example, "*classes (101-125)*" in italic means that the m=25 new classes from the index 101 to 125 are used for training, the corresponding trained model is f_1*(101-125)*. The details of datasets are followed the splitting methodology in Table 5.1 in Chapter 5.

Implementation details. We utilize Google Inception as a backbone net. The whole process includes two stages: initial training and incremental training. In the first stage, the initial model f_0 is trained on the n old classes by using the Adam optimizer with a learning rate of 1×10^{-6}, its embedding net is updated with a learning rate of 1×10^{-5}. In the second stage, we train a new model f_1 based on the converged f_0 on the m new classes using Eq. 6.3, with the same learning rate in the first stage. The model f_0 trained on the n old *classes (1-100)* or *(1-60)* is wrote as f_0*(1-100)* or f_0*(1-60)*. Likewise, the model f_1 is represented by the added m new classes, such as f_1*(101-200)* or f_1*(61-120)*. Following the practice in [131, 226], the output 512-d features (F^d in Figure 6.2) are used for retrieval[1]. We set the plasticity factor $\lambda_1 = 1$ and stability factor $\lambda_2 = 10$ in Eq. 6.3 for the following experiments.

Evaluation metrics. We use the Recall@1 [131, 248] and mean Average Precision (mAP) as retrieval metrics, and use average incremental accuracy [232, 249] and average forgetting [246] to evaluate incremental learning.

Table 6.1: Recall@1 and mAP (%) of incremental FGIR trained for the one-task scenario, "Initial model f_0*(1-100)*" indicates model trained on the first 100 classes on the CUB-200 datasets. "Reference model" indicates the model f_0*(1-200)* trained on all classes of CUB-200. The best performance is reported in boldface.

Dataset	**Caltech-UCSD Birds-200**					
Configuration and Results (%)	Old *classes (1-100)*		New *classes (101-200)*		Average	
	Recall@1	mAP	Recall@1	mAP	Recall@1	mAP
Initial model f_0*(1-100)*	79.24	55.78	46.93	19.54	63.09	37.66
⇒ Model f_1 w fine-tuning	70.21	42.57	**75.13**	**48.90**	72.67	45.74
⇒ Model f_1 w EWC [213]	73.32	45.73	72.84	44.14	73.08	44.94
⇒ Model f_1 w ALASSO [227]	72.88	43.87	72.94	45.50	72.91	44.69
⇒ Model f_1 w NCE$_{EWC}$ [233]	72.63	43.80	73.07	45.15	72.85	44.48
⇒ Model f_1 w L2 loss [219]	75.93	50.23	74.12	47.47	75.03	48.85
⇒ Model f_1 w MMD loss [37]	77.03	51.10	74.12	45.05	75.58	48.08
⇒ Model f_1 w Our method	**77.71**	**52.25**	75.00	46.51	**76.36**	**49.38**
Reference model f_0*(1-200)*	78.18	52.17	79.24	50.99	78.71	51.58

6.4.2 One-task scenario evaluation

Baselines. CIHR [236] and DIHN [218] have been explored for incremental hashing retrieval. The main difference with ours is that they used old data for training, while we use new data *only*. For a fair comparison, we take [37] as a baseline in which the feature-level regularization (*i.e.*, maximum mean discrepancy (MMD) loss) is used. We also compare to the popular algorithms including EWC[2], ALASSO[3], NCE loss[4], and L2 loss. Specifically, EWC [213] and ALASSO [227] are the network parameters

[1]Code available at: https://github.com/cw1091293482/Deep-Incremental-Image-Retrieval
[2]https://github.com/joansj/hat/tree/master/src/approaches
[3]https://github.com/dmpark04/alasso
[4]https://github.com/ProsusAI/continual-object-instances

Table 6.2: Recall@1 and mAP (%) of incremental FGIR trained for the one-task scenario, "Initial model f_0*(1-60)*" indicates model trained on the first 60 classes on the Stanford-Dogs datasets. "Reference model" indicates the model f_0*(1-120)* trained on all classes of Stanford-Dogs. The best performance is reported in boldface.

Dataset	**Stanford-Dogs-120**					
Configuration and Results (%)	Old *classes (1-60)*		New *classes (61-120)*		Average	
	Recall@1	mAP	Recall@1	mAP	Recall@1	mAP
Initial model f_0*(1-60)*	81.27	66.05	69.28	34.13	75.28	50.09
$\Rightarrow$ Model f_1 w fine-tuning	73.96	45.24	**83.69**	**67.25**	78.83	56.25
$\Rightarrow$ Model f_1 w EWC [213]	74.76	46.92	81.45	62.69	78.11	54.81
$\Rightarrow$ Model f_1 w ALASSO [227]	75.92	48.35	81.50	63.40	78.71	55.88
$\Rightarrow$ Model f_1 w NCE$_{EWC}$ [233]	75.12	47.88	81.62	62.99	78.37	55.44
$\Rightarrow$ Model f_1 w L2 loss [219]	78.99	56.57	83.23	66.63	81.11	61.60
$\Rightarrow$ Model f_1 w MMD loss [37]	79.49	**59.43**	83.35	65.21	81.42	**62.32**
$\Rightarrow$ Model f_1 w Our method	**79.92**	58.37	83.48	66.01	**81.70**	62.19
Reference model f_0*(1-120)*	80.37	62.48	83.10	66.78	81.74	64.63

regularization methods. To deploy these methods, we further train a classifier on the top of the embedding net. NCE loss [233] regularizes the inner product of an anchor-positive pair and anchor-negative pairs via a normalized cross-entropy loss. This method is combined into EWC algorithm. We follow this protocol by mining 9 hard negative samples (termed as NCE$_{EWC}$). L2 loss [217] focuses on minimizing the Euclidean distance between the features from the teacher-student models. For a fair comparison, the above four methods are trained with triplet loss $L_{triplet}$, having the same hyper-parameter $\lambda_1 = 1$. In terms of the plasticity factor λ_2, we tune this factor for four methods in incremental FGIR until we get their optimal performance. As a result, the corresponding plasticity factors are tuned as 8000, 0.2, 10, and 0.1, respectively. Moreover, the "Reference" by joint learning serves as an *upper-bound* performance. The fine-tuning method is also used as a reference for the new tasks since there is no knowledge distillation regularization.

One-task incremental learning (m=100 or m=60) is similar to transfer learning, while incremental training further emphasizes reducing forgetting on the n old classes. The results are reported in Tables 6.1 and 6.2. Note that only model f_0 is available, thereby it is unnecessary to estimate virtual features.

Naturally, the initial model f_0 trained on the n old classes performs poorly on the m new unseen classes. Take the CUB-200 dataset as an example, mAP is 19.54% when the initial model f_0*(1-100)* is tested on the m new classes without any re-training. Using the initial model f_0, we further re-train on the m new classes using different incremental algorithms to obtain the model f_1, whose performance is distinct on the old and new classes, as shown in Table 6.1. The fine-tuning method achieves the best accuracy on the new classes, it improves the accuracy (19.54%$\rightarrow$48.90% in mAP) on the new classes on the CUB-200 dataset but degrades accuracy (*i.e.*, forgetting)

on the old classes (55.78%→42.57% in mAP). Similar trends can be observed on the Stanford-Dogs dataset in Table 6.2.

For other algorithms, the models trained by network parameters regularization methods such as EWC and ALASSO show a similar trend that they reduce forgetting on the n old classes, but their performance on the m new classes is less competitive compared to the fine-tuning method. NCE_{EWC} regularises metric learning via cross-entropy loss on the feature embeddings. We find this method has some limited benefits. For example, it improves on the Stanford-Dogs dataset in terms of the average performance. L2 loss and MMD loss regularize the features directly. For L2 loss, it regularizes the model f_1 to forget less on the old classes of two datasets. For instance, on the CUB-200 dataset, it reduces the degradation by 3.31% of Recall@1 (79.24%→75.93%) and 5.55% of mAP (55.78%→50.23%), see Table 6.2 and Table 6.1 for details.

MMD loss is more similar to our method in which feature correlations are also considered [37]. Compared to MMD loss, our method, in most cases, suffers less accuracy degradation on two datasets. For instance, our method degrades the Recall@1 on the n old classes by 1.53% (79.24%→77.71%) and 1.35% (81.27%→79.92%) on CUB-200 and Stanford-Dogs, respectively, whereas the MMD loss degrades the Recall@1 on the old classes by 2.21% (79.24%→77.03%) and 1.78% (81.27%→79.49%) on two datasets. Moreover, in terms of the performance on the m new classes, our method also achieves closer accuracy to that of the fine-tuning method.

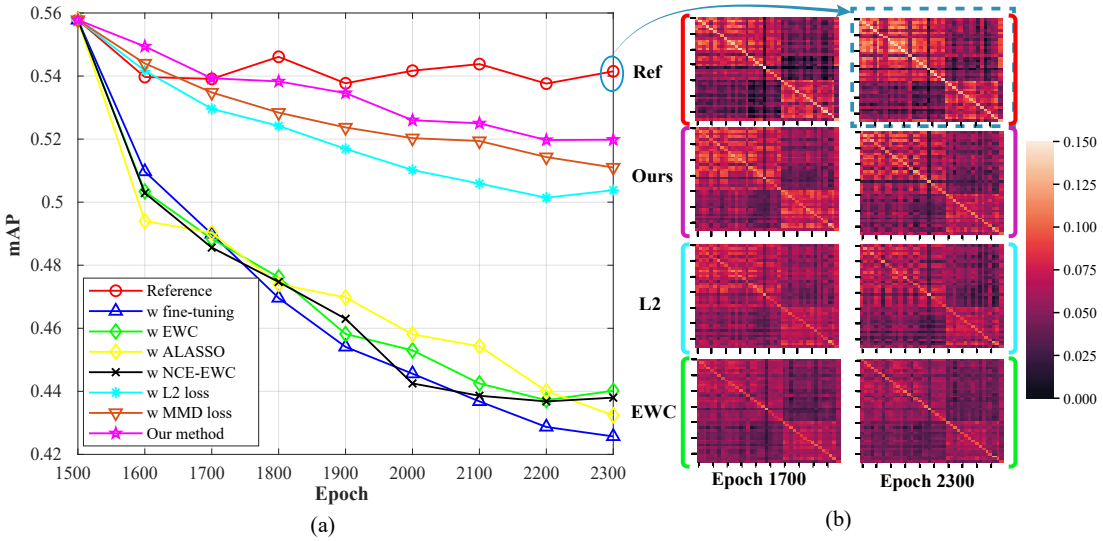

Figure 6.6: (a) mAP evolution of old classes *(1-100)* tested on the CUB-200 dataset under one-task scenario. (b) The Gram matrices of four representative methods (best viewed in color). More brightness indicates higher semantic correlations between two samples. The reference performance is obtained by joint training. Our method retains most semantics (higher brightness) compared to EWC and L2 loss.

Besides, we report the mAP evolution during incremental training in Figure 6.6(a). The activation regularization methods (*e.g.*, L2 loss) outperform the network param-

eters regularization methods (*e.g.*, EWC). Moreover, we visualize the Gram matrices of three methods. As the training proceeds, their differences with respect to the reference Gram matrices are maximized. Namely, the bright area in the three methods becomes ambiguous. However, our method retains most semantics of old classes (more brightness) than the other two continual learning strategies even at the last training epoch.

6.4.3 Multi-task scenario evaluation

Multi-task scenario refers to the case that m new classes are divided evenly into several groups and added sequentially. For the CUB-200 dataset, the remaining 100 new classes are split into 4 disjoint groups, with 25 classes per group; For the Stanford-Dogs dataset, we also get 4 groups with 15 classes per group. Thus, there are 4 steps incremental training for each dataset. For each step, the model is trained only on the images from a new class group (*e.g.*, *classes (126-150)* of the CUB dataset) and is tested separately in prior groups (*e.g.*, *classes (1-100)* and *classes (101-125)*) to evaluate the forgetting rate of this step. Note that incremental performance is *insensitive* to the arrival order and choice of new classes since the tasks do not depend on softmax-based probabilities [250].

Accuracy change range. We estimate the features of previous models (using Eqs. 6.7 and 6.8) based on the accuracy change defined in Eq. 6.6. Concretely, we use mAP to calculate the accuracy range. For instance, on the CUB-200 dataset, model f_0*(1-100)* takes as input the first group of new classes (see Figure 6.3) and produces an incrementally-trained model f_1*(101-125)*. In terms of mAP, it degrades from 54.20% to 52.44% on the $n = 100$ old classes while increases from 29.82% to 52.27% on the $m = 25$ new classes. These recorded mAPs are used to calculate the accuracy change range $(\alpha_{drop}, \alpha_{gain})$ using Eq. 6.6. Finally, the mAP change range is (-0.0325, 0.7528) during task $t = 1$ and is used to estimate the features for model f_0 when training the next task $t = 2$, without storing this model. The estimated features serve as an extra regularization for training task $t = 2$ in which the knowledge is mainly transferred from the model f_1*(101-125)* to f_2*(126-150)*. This process is performed repeatedly until all new class groups are added. The earlier feature estimation procedure becomes less reliable as more groups of new classes are added. We solve this issue by decreasing importance factors in Eq. 6.11. For multi-task scenario, we keep the plasticity factor λ_1 and stability factor λ_2 in Eq. 6.3 as 1 and 10, respectively.

We adopt forgetting measurement [246] to quantify the forgetting ratio. Specifically, the forgetting ratio for a particular task is defined as the difference between the maximum accuracy gained throughout the incremental training process in the past and the accuracy the currently-trained model has, then all t tasks forgetting ratios are averaged:

$$forgetting = \frac{1}{t-1}\sum_{j=1}^{t-1}\Big(\max_{l\in\{1,...,t-1\}} Acc_{l,j} - Acc_{t,j}\Big), \forall j < t \tag{6.12}$$

where $Acc_{t,j}$ denotes the accuracy of j^{th} group of new classes evaluated by the model trained on the task t. Concretely, we employ the mAP metric as Acc for evaluation. When the model has been incrementally trained up to task t, we measure and then average all previous forgetting ratios $(1, 2, ..., t-1)$ using Eq. 6.12 as final forgetting evaluation.

The average forgetting ratios are depicted in Figure 6.7. Note that we use the task index to indicate the group of new classes being added. For example, "$t = 2$" on the CUB-200 dataset means the model is training on the 2^{nd} group of new classes and then tested on *classes (1-100)* and *classes (101-125)* separately. Obviously, all methods suffer catastrophic forgetting on two datasets. In particular, fine-tuning on a new task leads to significant forgetting on the old tasks. EWC and ALASSO cannot reduce the forgetting issue ideally in the multi-task scenario. By contrast, activation regularization methods perform better on two datasets. Particularly, MMD loss and our method, by distilling feature correlations, can significantly reduce the forgetting ratio compared to the L2-regularized feature alignment method. Our method can further largely mitigate the forgetting ratio when feature estimation is considered into correlations distillation. Finally, our method has the least forgetting ratio (up to 10%) on these two datasets.

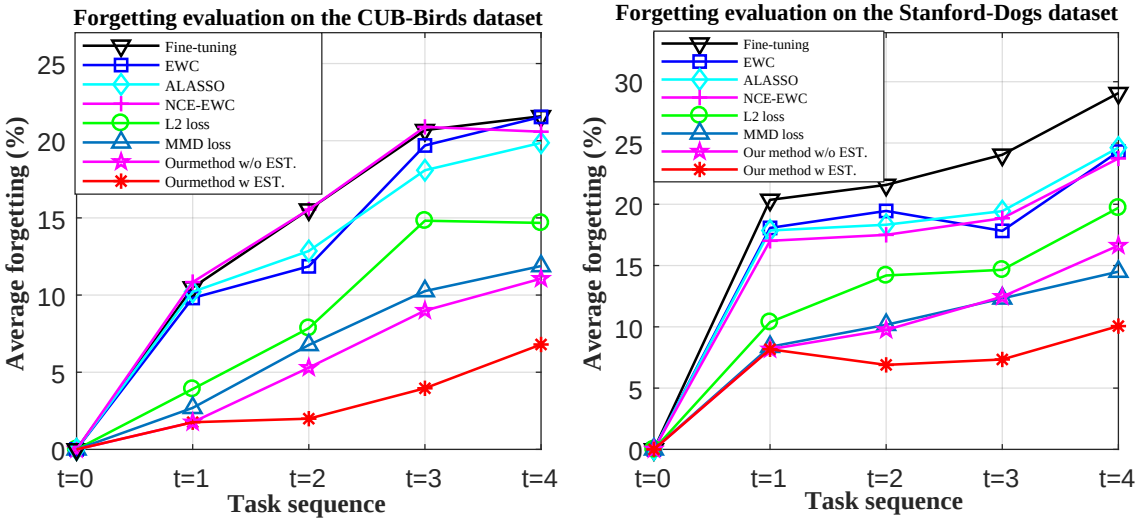

Figure 6.7: Average forgetting evaluation. "w/o EST." indicates that feature ESTimation strategy is not included in our method (see Eq. 6.11). The forgetting is measured on previous old classes after training on current new classes. The forgetting ratios over all previous tasks are averaged to show. The higher value indicates the more severe forgetting.

After all new tasks are added sequentially (*i.e.* $t = 4$), we get the final model

f_4(176-200) or *f_4(106-120)* for this task. We measure the accuracy of each prior task (*i.e.*, class group) using the final model. We take Recall@1 as a metric for demonstration, as shown in Figure 6.8, including the performance for the previous tasks and the last new task. In this experiment, we use the performance of joint training as reference upper bound. In terms of Recall rate tested on the last new class group (*i.e.*, *classes (176-200)* and *classes (106-120)*), we find all six incremental learning algorithms and the fine-tuning method (without any knowledge distillation) have similar performance, close to the upper bound, especially for the Stanford-Dogs dataset. However, in terms of Recall on previous tasks, feature correlations used as knowledge can lead to a better-performing performance than other counterparts, closer to the upper bound, which means that our method suffers less forgetting on these preceding tasks. For instance, when tested the final model *f_4(176-200)* on the old *classes (1-100)* of the CUB-200 dataset, our method achieves around 73% of Recall@1, 7% lower than the upper bound (80%), whereas other methods achieve less than 70%.

We have demonstrated that our method can reduce the catastrophic forgetting on the previous tasks effectively. Also, the performance of the new task is essential to evaluate. As the incremental training proceeds, we report the Recall@1 on the new task during each incremental step in Figure 6.9. That is, we record the accuracy of new classes every time these classes are added. The results illustrate the evolution of performance on new classes. Obviously, we observe that all methods have similar Recall evolution and their performance is close to each other, especially for the Stanford-Dogs dataset.

We evaluate the case when more tasks are added sequentially on the CUB-200 dataset. Concretely, the remaining m=100 new classes are divided into 10 groups evenly. We focus on activation regularization algorithms and compare with L2 and MMD loss regularized methods. After the final model *f_{10}(191-200)* is trained at the end of the task sequence (*i.e.*, new *classes (191-200)*), we test this model on the original *classes (1-100)*, which suffer the most severe forgetting. The results are reported in Figure 6.10. Obviously, on the original *classes (1-100)*, correlations distillation with feature estimation method reduces the forgetting on *classes (1-100)* effectively.

6.4.4 Ablation study

a. Efficacy of feature estimation

Feature estimation is introduced in Eq. 6.11 to reduce forgetting in the multi-task scenario. Here, we explore the efficacy of feature estimation. For this purpose, we consider a vanilla correlations distillation only from task (t–1) to task t, *i.e.*, without using the feature estimation. Therefore, the loss for training is $L = \lambda_1 L_{triplet} + \lambda_2 L_{corr}^{(t-1)\rightarrow t}$.

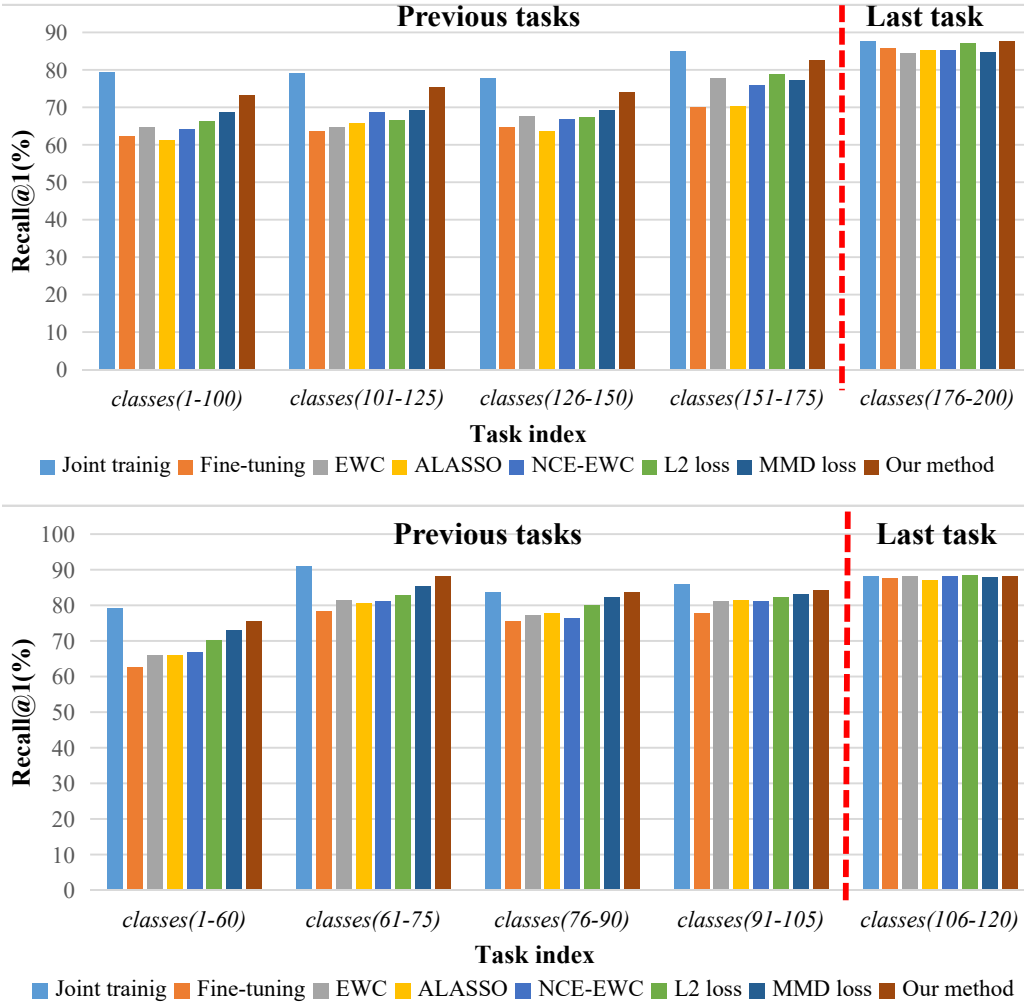

Figure 6.8: The Recall@1 evaluation of each task (class group) at the end of the 4-step incremental learning. For instance, the model f_4*(176-200)* incrementally-trained on 4^{th} new *classes (176-200)* at task $t = 4$ and is tested on all previously seen class groups. (a) Tested on the CUB-200 dataset; (b) Tested on the Stanford-Dogs dataset.

We follow previous experimental protocols and conduct this study on the CUB-200 dataset. We depict the Recall@1 and mAP evolution in Figure 6.11. Note that it is unnecessary to estimate feature drifts when task $t = 1$. When more new classes are added, distilling as knowledge feature correlations like MMD loss and our vanilla distillation method is more effective than L2 loss for reducing performance degradation. Also, vanilla distillation without feature estimation has a higher performance than MMD loss. When feature estimation strategy is used, additional regularization from unsaved models can effectively retain more previously-learned knowledge, thereby leading to less forgetting on the original *classes (1-100)*.

b. Influence of hyper-parameter

We show the efficacy of feature estimation in Figure 6.11. However, it seems that the estimated features in Eq. 6.11 act as augmented components for reducing catas-

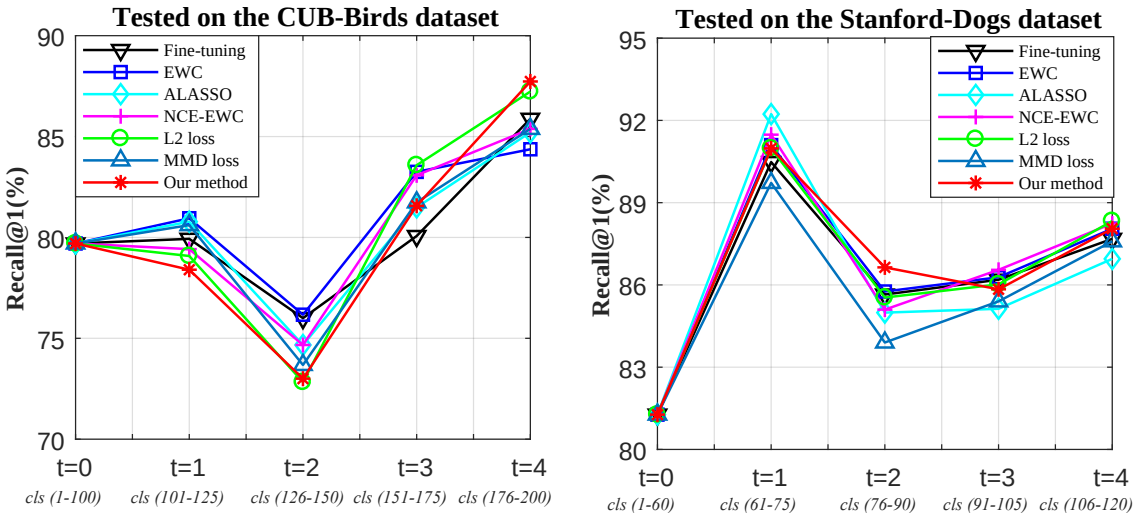

Figure 6.9: The Recall@1 evolution tested on each new incoming class group during incremental learning. "*cls (1-100)*" indicates "*classes (1-100)*".

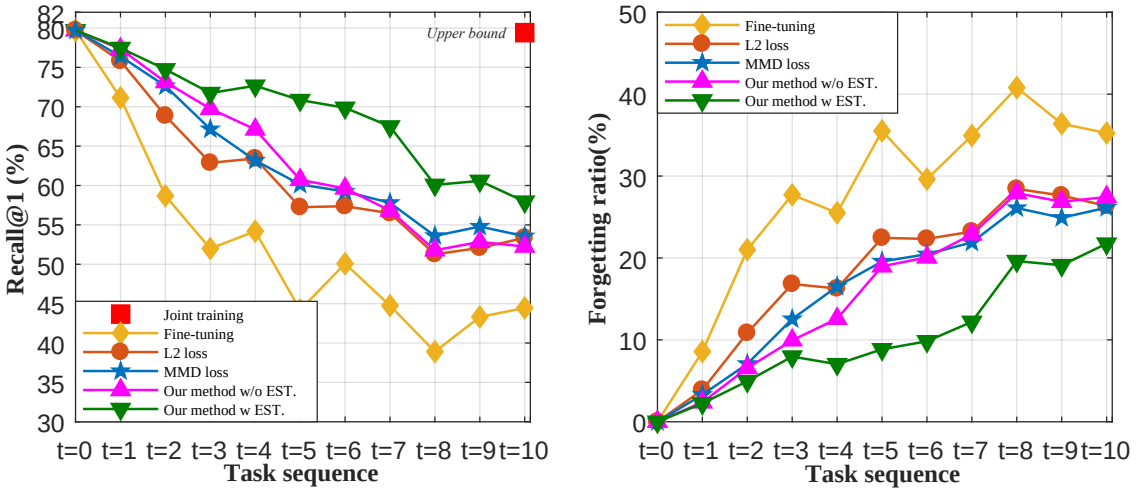

Figure 6.10: 10-task performance comparison on the old *classes (1-100)*. The testing model is trained at the end of 10 tasks sequence on CUB-200. (a) Evolution of Recall@1; (b) Forgetting ratio evaluated on Recall@1.

trophic forgetting. In other words, the forgetting ratio reducing on the old classes might be realized by the hyper-parameter. To this end, we explore the influence of hyper-parameter. Following previous experimental protocols, we consider two-step incremental training on the CUB-200 dataset where only new *classes (101-125)* and *classes (126-150)* are sequentially added. We do not consider task t=1 is because there is no feature estimation in this task. When new *classes (126-150)* are adding at task $t=2$, the deep network is trained, using Eqs. 6.3 and 6.11, under four conditions: **case (a)** without feature estimation, **case (b)** with hyper-parameter augmented, **case (c)** with feature estimation, and **case (d)** with two-model distil-

lation. The case (a) is viewed as a baseline where the correlations are distilled only from the penultimate model f_1*(101-125)* to the to-be-trained model f_2*(126-150)* by using their actual features. The case (d) is a complete method, similar to [235] in which the previous models f_0*(1-100)* and f_1*(101-125)* are both saved for regularizing the training of current task $t = 2$. In contrast, it is unnecessary for our method (case (c)) to save the model f_0*(1-100)*.

The results are reported in Table 6.3. Naturally, the complete method in the case (d) produces an optimal performance on the old classes because all models are available. In terms of the baseline method, due to no distillation regularization, the trained model f_2*(126-150)* has the best performance on the new classes. For instance, its mAP reaches the maximal 52.45%. However, this model degrades performance heavily on the old classes to a minimal mAP (48.09%). In contrast, when the hyper-parameter of the baseline is augmented from λ_2 to $\lambda_2(1 + \frac{1}{(t-1)})$. The trained model f_2*(126-150)* reduces forgetting on the old classes but limits the learning on the new classes. In particular, compared to the baseline method, the mAP of the case (b) on the old *classes (1-100)* reaches a maximal 50.71%, while it has the lowest Recall@1 (75.00%) and mAP (50.87%) on the new *classes (126-150)*. Therefore, Simply increasing the hyper-parameter of the stability term λ_2 in Eq. 6.3 cannot tackle well the stability-plasticity dilemma on the old tasks and new task because no extra knowledge is transferred. By contrast, training by using the feature estimation method can achieve competitive accuracy, taking both the old classes and new classes into account. Specifically, the model trained using the feature estimation method has a similar performance to the "two-model distillation" method on the old classes (76.19% → 76.91% of Recall@1). Meanwhile, the performance on the new classes is close to that of the baseline method (76.33% → 76.83% of Recall@1).

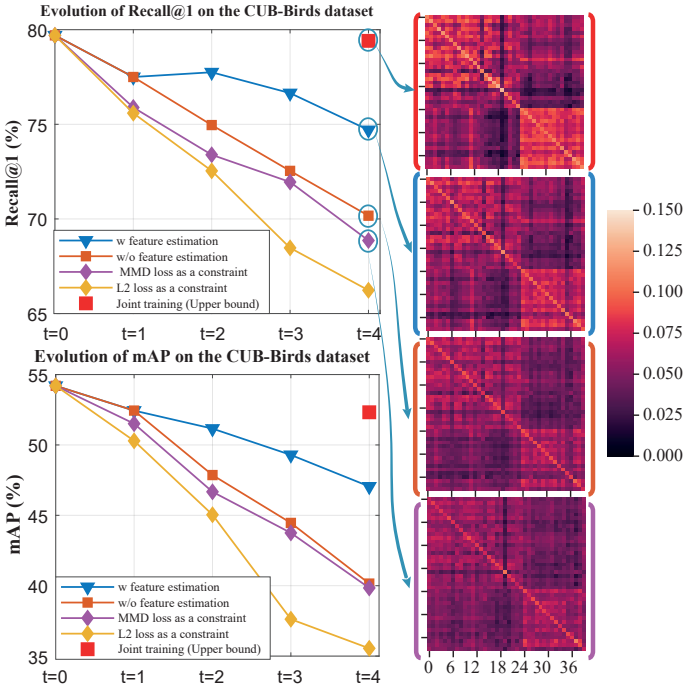

Figure 6.11: Efficacy exploration for (a) Recall@1 and (b) mAP evolution *only* tested on the original *classes (1-100)*. We show the correlation matrices at the end of incremental training. This visualization further indicates that learning with feature estimation makes its performance closer to the upper bound.

Table 6.3: Hyper-parameter analysis (%) on CUB-200 where training task $t = 2$. We set $\lambda_1 = 1$ and $\lambda_2 = 10$. L_{actual} means that the loss term is computed by using actual features, whereas $L_{virtual}$ denotes the one computed by using estimated features.

Configurations		Old *classes (1-100)*		New *classes (126-150)*	
Conditions	The form of loss function $L =$	Recall@1	mAP	Recall@1	mAP
Case (a)	$\lambda_1 L_{triplet} + \lambda_2 \left(L_{actual}^{(t-1)\to t} \right)$	74.75	48.09	76.83	52.45
Case (b)	$\lambda_1 L_{triplet} + \lambda_2 \left(L_{actual}^{(t-1)\to t} + \frac{1}{(t-1)} L_{actual}^{(t-1)\to t} \right)$	76.69	50.71	75.00	50.87
Case (c)	$\lambda_1 L_{triplet} + \lambda_2 \left(L_{actual}^{(t-1)\to t} + \frac{1}{(t-1)} L_{virtual}^{(t-2)\to t} \right)$	76.19	50.45	76.33	51.79
Case (d)	$\lambda_1 L_{triplet} + \lambda_2 \left(L_{actual}^{(t-1)\to t} + L_{actual}^{(t-2)\to t} \right)$	76.61	50.49	76.50	51.92

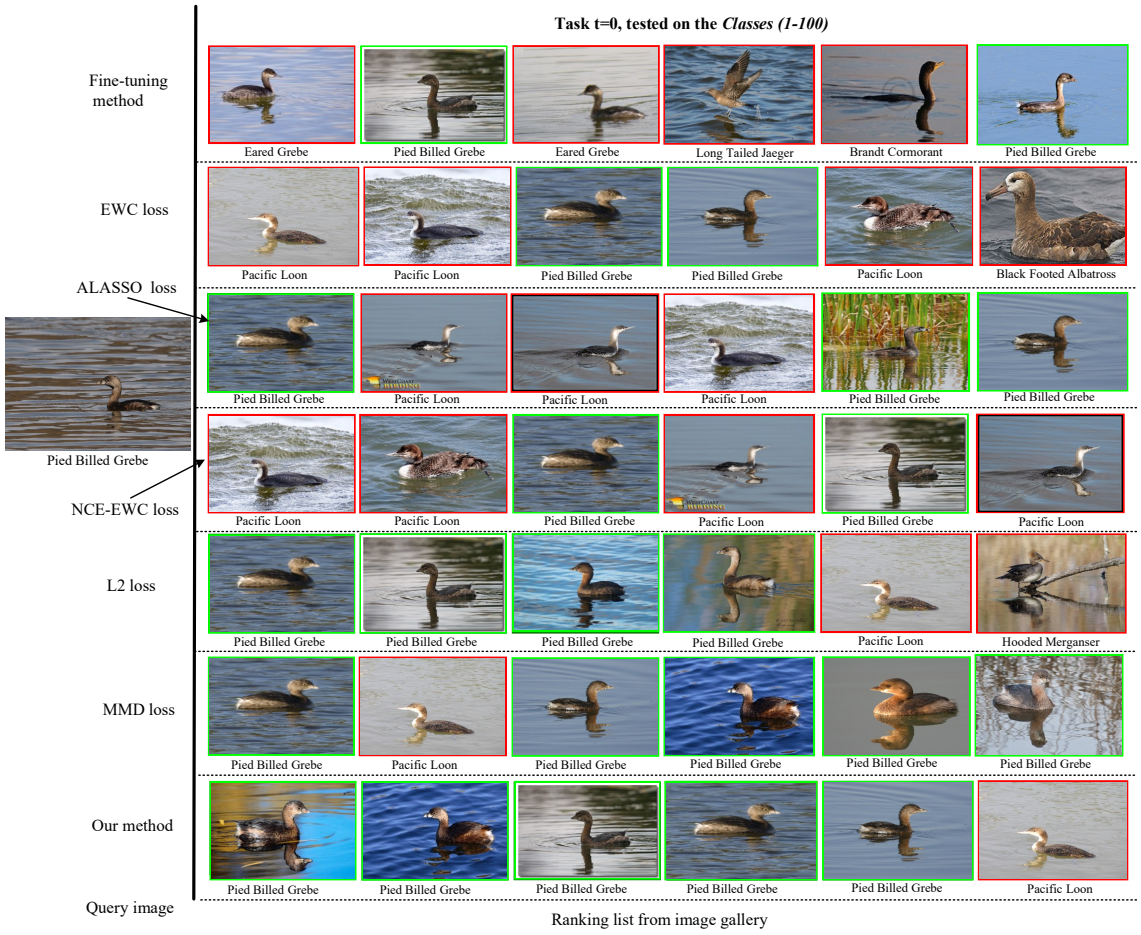

Figure 6.12: Visualization of retrieved images and their class names on the CUB-200 dataset. The Top 6 images tested on *classes (1-100)* are listed from left to right.

6.4.5 Retrieval visualization

We visualize the retrieval results for different methods on 4-tasks sequentially incremental learning on the CUB-200 dataset. *For all methods on different incremental stages, the query image is the same. The red box means an image is retrieved incorrectly, while the green box indicates the retrieved image has the same class label as the query image.* We use the model trained at the end of the 4-step sequentially

incremental training, *i.e.*, the model f_4*(176-200)*, and test this model on the old *classes (1-100)*. Considering the differences among images are subtle, we report the retrieved images and corresponding class names. We select an image from class "*Pied Billed Grebe*" as the query item. This image is difficult to retrieve and is prone to cause forgetting issue because the color of the object in this image is similar to the background, as well as its incomplete appearance. The top 6 retrieved results are shown in Figure 6.12. Overall, all methods can return images with similar scenes. Other incremental algorithms suffer catastrophic forgetting and return more incorrect images. By contrast, our method effectively reduces the forgetting ratio and still returns more correct images of the old tasks after a process of 4-step incremental learning. When the model f_4*(176-200)* are validated on on previous tasks: *classes (101-125)*, *classes (126-150)*, and *classes (151-175)*. Note that *classes (176-200)* are used as the current new classes. The visualizations are depicted in Figure 6.13, Figure 6.14, Figure 6.15, and Figure 6.16, respectively.

6.5 Chapter Conclusions

In this chapter, we explored fine-grained image retrieval in the context of incremental learning, where one-task and multi-task scenarios are validated. To achieve a trade-off performance for old tasks and new tasks, we used new data only and regularized their features extracted from the teacher model and the student model. In terms of multi-task incremental learning, saving all previous models for correlations distillation may cause a great demand in memory storage. We made an attempt to address the issue via a feature estimation method. That is, instead of storing a stream of old models, we saved the accuracy of models to compute the accuracy change during training each task. The semantic correlations of the estimated features, as an additional regularization, further mitigated the catastrophic forgetting ratio on previous tasks. Compared to previous approaches, the advantages of the proposed method were verified by thorough quantitative and qualitative results on two fine-grained datasets. Now, incremental image retrieval methods still need supervisory information. In the future, it is potentially valuable to explore incremental image retrieval in an unsupervised learning manner. Further, the data used in old tasks and new tasks share similar semantic commonalities, it is also interesting to examine for heterogeneous data.

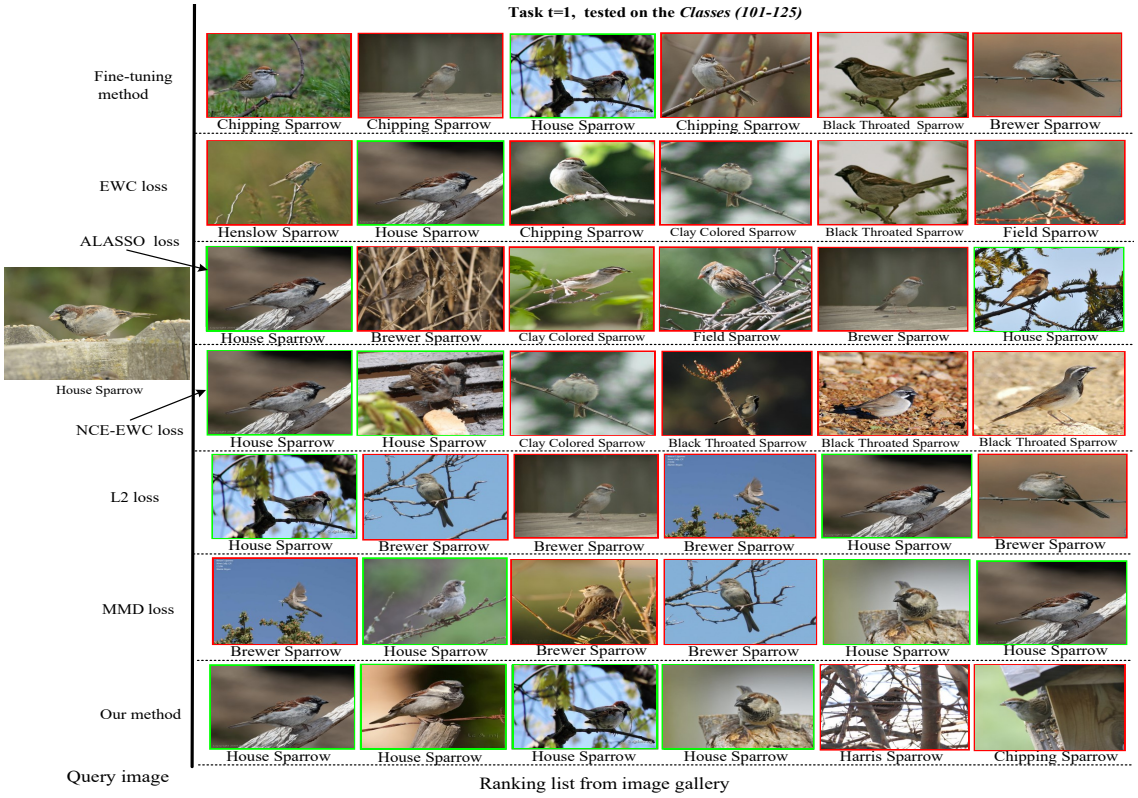

Figure 6.13: The top-6 retrieval results of the model f_4*(176-200)* tested on the previous old *classes (101-125)*.

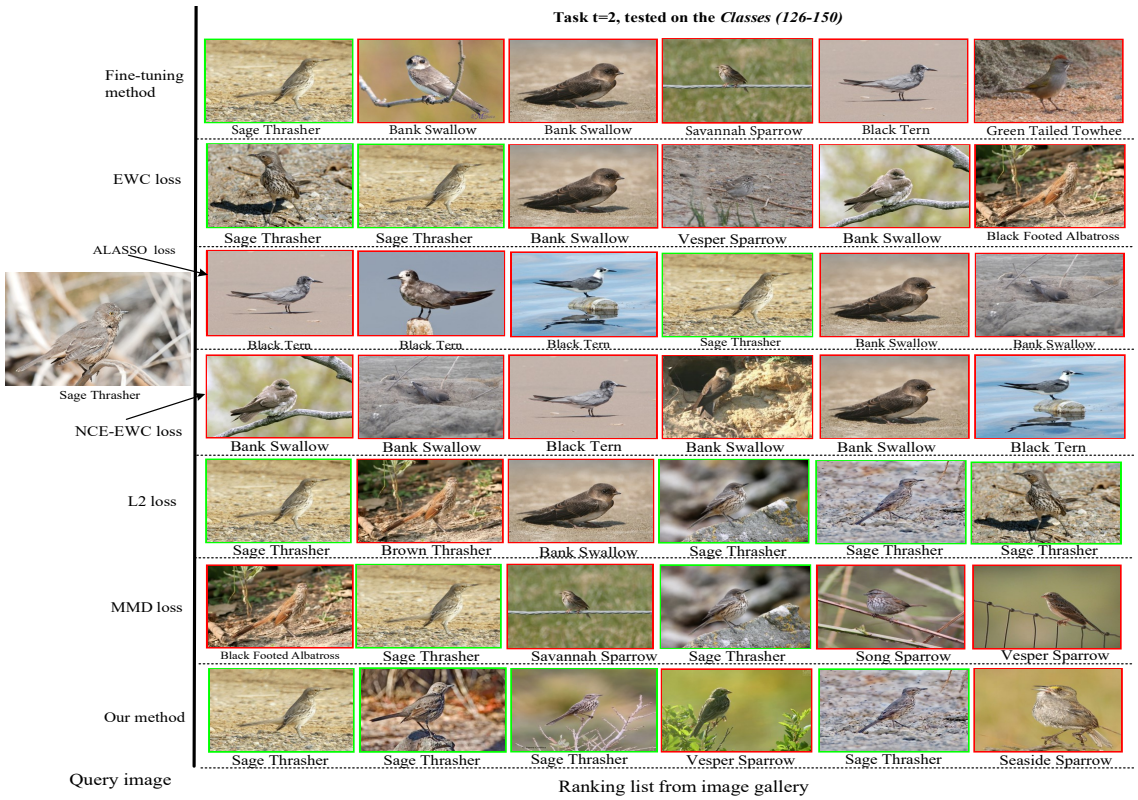

Figure 6.14: The top-6 retrieval results of the model f_4*(176-200)* tested on the previous old *classes (126-150)*.

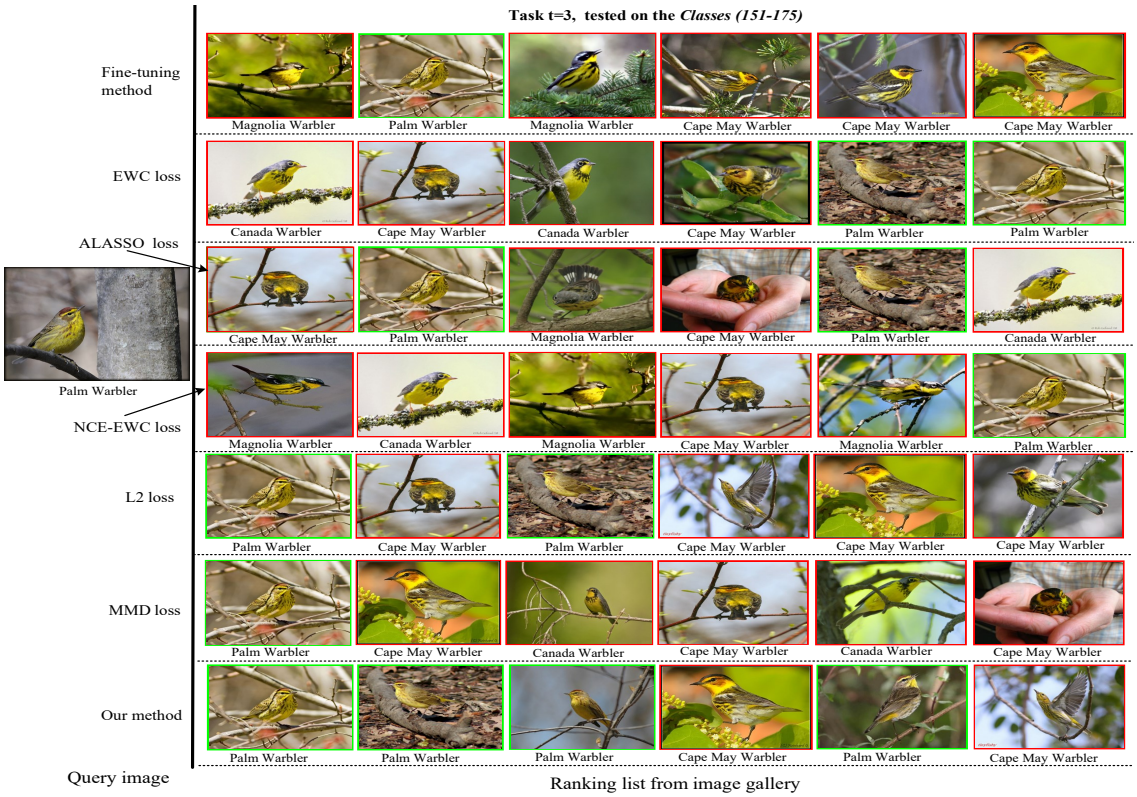

Figure 6.15: The top-6 retrieval results of the model f_4*(176-200)* tested on the previous old *classes (151-175)*.

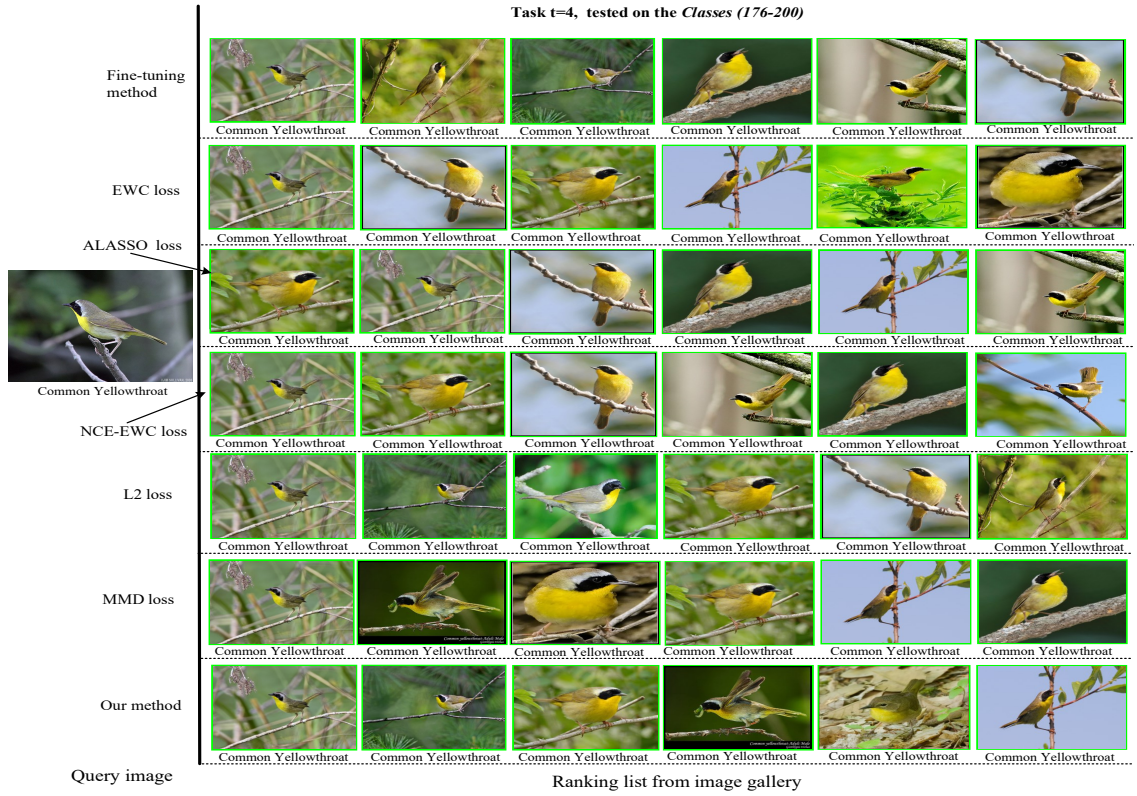

Figure 6.16: The top-6 retrieval results of the model f_4*(176-200)* tested on the current new *classes (176-200)*.