A. Appendix

A.1. Experiment details

A.1.1 Implementation details

We train our model on all datasets for 500 epochs. The optimizer is Adamw which sets weight-decay as $1 \times 10^{-6}$, decay-rate as 0.1 and cosine learning rate as $1 \times 10^{-3}$, including 5 epochs of warmup like other vision transformer’s training. Also, the backbone is modified from RVT [6]. we applied the standard data augmentation of RVT [6]. We use RandAugment [1], random cropping, horizontal flip, normalization and cutout [2]. The cutout [2] is only used in CIFAR100 like FOSTER [8]. To alleviate the influence of unbalanced datasets, following LUCIR [4], DER [10], and Dytox [3], at the end of each task, we finetune our model with the balanced datasets selected in the memory buffer to alleviate the bias between the old categories and new categories. We choose to finetune our model with a learning rate $1 \times 10^{-4}$. In CIFAR100, For the 10 steps set, we finetune the model for one epoch before 3 steps and 10 epochs for other steps. For the 5 steps set, we finetune the model for one epoch before 2 steps and 10 epochs for other steps. For the 20 steps set, we finetune the model for one epoch before 4 steps and 10 epochs for other steps. In ImageNet100/1000, we finetune the model for 5 epochs before 3 steps and 10 epochs for other steps. During model training, the task-specific tokens and stability classifier in the duplex classifier are frozen to preserve the previous knowledge. Conversely, the plastic classifier in the duplex classifier and the task-general token are allowed to learn new knowledge and update the general knowledge in the tasks, respectively. During finetuning, we fix the feature extractor including SABs and GKAB. Corresponding to this, we open SKAB and duplex classifier to reduce the bias towards the new categories. We fix the task-specific tokens to maintain specific knowledge for previous tasks and set the task-general token trainable for GKAB to update the general knowledge. The order and sampler strategy are following Dytox [3]. Our batch size is 256 to suit our GPUs.

To balance the training time and accuracy, we set batch size 256 for two GPUs (RTX3090) on CIFAR100 and ImageNet100, and 256 for four GPUs on ImageNet1000 to accelerate training.

A.1.2 Memory Buffer Setting

Differing from CNN, Vision Transformers usually use adam [5] optimizer and DistributedDataParallel instead of SGD and DataParallel. So the previous work [3] has made a mistake. The model selects N times anchors than the memory buffer, in which N is the number of GPUS, causing the accuracy of the paper much higher than true performance (73.3% → 67.33% in 10 steps on CIFAR100). So in erratum distributed, It [3] reports the true performance and we compare our model with this performance. We solve this bug to maintain a suitable memory buffer. We suspend all but GPU0 and use GPU0 to select anchors using the way in iCaRL [7] like other continual learning methods [3, 7, 9–11]. We store the anchors as a new file and then push the anchors into other GPUs to guarantee that the memory buffer is suitable. Our experiment in Table 1 shows that our model can get similar performance in two or four GPUS just need to change the batch-size like 128 in four GPUs and 256 in two GPUs.

A.1.3 Computation FLOPs

Although FLOPs may not be a critical factor in incremental learning tasks, we posit that they can serve as a useful metric to gauge the superiority of a model. Thus, we have elected to include a comparison of FLOPs with respect to previous tasks in table 2. In Table 2, we have provided FLOPs of the proposed DKT and the competitive DyTox global and DER w/o P methods on CIFAR100 and ImageNet1000. We can see that, as DKT only needs to increase the computation of a task-specific token for each incremental task addition-

<table>
<thead>
<tr>
<th>GPUs</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>75.83</td>
</tr>
<tr>
<td>4</td>
<td>75.61</td>
</tr>
</tbody>
</table>

Table 1. Performance on several GPUs in 10 steps on CIFAR100
ally, instead of accumulatively computing the decoder like DyTox, it is almost computation cost-free in learning new tasks (less than 1% FLOPs increase after the final task). Instead, DyTox global and DER w/o P suffer about 30.8% and 899.7% FLOPs increase after the final learning task. This result demonstrates the efficiency of our novel design using cross-attention and task-specific tokens in learning new tasks.

<table>
<thead>
<tr>
<th>Methods</th>
<th>CIFAR100 FLOPs start</th>
<th>CIFAR100 FLOPs final</th>
<th>ImageNet1000 FLOPs start</th>
<th>ImageNet1000 FLOPs final</th>
</tr>
</thead>
<tbody>
<tr>
<td>DER w/o P</td>
<td>0.372</td>
<td>3.716</td>
<td>1.827</td>
<td>18.265</td>
</tr>
<tr>
<td>DyTox global</td>
<td>0.606</td>
<td>0.793</td>
<td>2.010</td>
<td>2.552</td>
</tr>
<tr>
<td>Our DKT</td>
<td>0.615</td>
<td>0.618</td>
<td>2.064</td>
<td>2.066</td>
</tr>
</tbody>
</table>

Table 2. Computation complexity (i.e., FLOPs (G)) on CIFAR100 and ImageNet1000. FLOPs start/final represent the FLOPs before/after 10 incremental learning tasks, respectively.

References


