DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion: 
Supplementary Materials

Arthur Douillard\textsuperscript{1,2}, Alexandre Ramé\textsuperscript{1}, Guillaume Couairon\textsuperscript{1,3}, Matthieu Cord\textsuperscript{1,4}
\textsuperscript{1}Sorbonne Université, \textsuperscript{2}Heuritech, \textsuperscript{3}Meta AI, \textsuperscript{4}valeo.ai
arthur.douillard@heuritech.com, \{alexandre.rame, matthieu.cord\}@sorbonne-universite.fr, gcouairon@fb.com

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x^t_i, y^t_i))</td>
<td>Input sample &amp; its label from the (t^{th}) task</td>
</tr>
<tr>
<td>(C^t)</td>
<td>Label set of the (t^{th}) task</td>
</tr>
<tr>
<td>(C^{1..t})</td>
<td>All labels from all seen tasks</td>
</tr>
<tr>
<td>(\theta_t)</td>
<td>Task token of the (t^{th}) task</td>
</tr>
<tr>
<td>(\text{Clf}_t)</td>
<td>Independent classifier of the (t^{th}) task</td>
</tr>
<tr>
<td>(\text{SAB}_t)</td>
<td>(l^{th}) Self-Attention Block</td>
</tr>
<tr>
<td>(\text{TAB})</td>
<td>Task-Attention Block</td>
</tr>
</tbody>
</table>

Table 1: Notations used in the paper.

A. Appendix

Table 1 summarizes the notations used along this paper.

A.1. Experimental details

Datasets We use three datasets: CIFAR100 [15], ImageNet100, and ImageNet1000 [4]. CIFAR100 is made of 50,000 train RGB images and 10,000 test RGB images of size 32 × 32 for 100 classes. ImageNet1000 contains 1.2 million RGB train images and 50,000 validation RGB images of size 224 × 224 for 1000 classes. ImageNet100 is a subset of 100 classes from ImageNet1000. We follow PODNet [6] and DER [28] and use the same 100 classes they’ve used. Fine details about the datasets, like the class orders, can be found in the provided code in the options files (see readme).

Implementation For all datasets, we train the model for 500 epochs per task with Adam [13] with a learning rate of \(5e^{-4}\), including 5 epochs of warmup. Following UCIR [11], PODNet [6], and DER [28], at the end of each task (except the first) we finetuned our model for 20 epochs with a learning rate of \(5e^{-5}\) on a balanced dataset. In DyTox, we applied the standard data augmentation of DeiT [25] but we removed the pixel erasing [32], MixUp [30], and CutMix [29] augmentations for fair comparison. In contrast, in DyTox+ we used a MixUp [30] with beta distribution \(\beta(0.8, 0.8)\). During all incremental tasks (\(t > 1\)), the old classifiers \(\text{Clf}_i, i < t\) and the old task tokens \(\theta_i, i < t\) parameters are frozen. During the finetuning phase where classes are rebalanced [2, 11, 6, 28], these parameters are optimized, but the SABs are frozen.

Hyperparameter tuning In contrast with previous works [6, 28], we wanted stable hyperparameters, tuned for a single setting and then applied on all experiments. This avoids optimizing for the number of tasks, which defeats the purpose of continual learning [7]. We tuned hyperparameters for DyTox using a validation subset made of 10% of the training set, and this only on the CIFAR100 experiment with 10 steps. We provide in Table 2 the chosen hyperparameters. Results in the main paper shows that those hyperparameters reach state-of-the-art on all other settings and notably on ImageNet.

Baselines E2E [2] and Simple-DER [18] results come from their respective papers. All other baseline results are taken from the DER paper [28]. We now further describe their contributions. iCaRL [23] uses a knowledge distillation loss [10] and at test-time predicts using a k-NN from its features space. E2E [2] learns a model with knowledge distillation and applies a finetuning after each step. UCIR [11]...
uses cosine classifier and euclidean distance between the final flattened features as a distillation loss. BiC [27] uses a knowledge distillation loss and also re-calibrates [9] the logits of the new classes using a simple linear model trained on validation data. WA [31] uses a knowledge distillation loss and re-weights at each epoch the classifier weights associated to new classes so that they have the same average norm as the classifier weights of the old classes. POD-Net [6] uses a cosine classifier and a specific distillation loss (POD) applied at multiple intermediary features of the ResNet backbone. RPSNet [21] uses knowledge distillation and manipulates subnetworks in its architecture, following the lottery ticket hypothesis [8]. DER [28] creates a new ResNet per task. All ResNets’ embeddings are concatenated and fed to a unique classifier. ResNets are pruned using HAT [24] masking procedure. Note that DER pruning has multiple hyperparameters that are set differently according to the settings. Furthermore, the reported parameters count, after pruning, in [28] is an average of the count over all steps: the final parameters count (necessarily higher) wasn’t available. Finally, Simple-DER [18] is similar to DER, with a simpler pruning method which doesn’t require any hyperparameter tuning.

### A.2. Parameter sharing of the TAB

Previous dynamic methods as DER [28] and Simple-DER [18] shared no parameters between tasks until the final classifier. We proposed instead with DyTox to share the encoder (SABs) and the decoder (TAB) parameters across tasks, leading to a minimal memory overhead while also maintaining different tokens per task. In the first row, a different TAB is created per task, while in the second row the same TAB is used — which is the DyTox strategy. A different TAB per task leads to better results (56% v.s. 52% in “Last” accuracy) because the network can be more diverse with each TAB specialized to its associated task. This increased diversity has a drawback: the memory overhead is too important (97M v.s. 10M parameters). We find in practice that DyTox strikes a good balance between memory overhead and continual performance.

<table>
<thead>
<tr>
<th>TAB parameter sharing?</th>
<th>#P</th>
<th>Avg</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>97.59</td>
<td>72.20</td>
<td>56.00</td>
</tr>
<tr>
<td>✓</td>
<td>10.77</td>
<td>70.20</td>
<td>52.34</td>
</tr>
</tbody>
</table>

Table 3: Investigation of the parameter sharing of TAB. We report the “Avg” accuracy and the “Last” accuracy for the 50 steps setting on CIFAR100. The second row corresponds to DyTox.

### A.3. Novel continual training procedure

**DyTox++** We nicknamed DyTox+ our model when combined with a novel continual procedure based on MixUp [30]. We now refine DyTox+ into DyTox++ by adding a new component during the training: the Sharpness-Aware Minimizer (SAM) [16]. Indeed, aiming for wider minima is particularly important in continual learning [14, 26]. This is because sharp task-specific minima lead to over-specialization to a particular task and consequently to a forgetting of all other tasks. Weights constraints as EWC [14] or second-order optimization [17] have similar motivations. SAM estimates the worst closest parameters during a first forward/backward pass, and then optimizes the loss w.r.t. to them during a second forward/pass. In consequence, DyTox++ optimizes the loss not w.r.t. the current parameters but w.r.t. a region of possible parameters leading to wide local minima that span across multiple tasks. In practice, we used the Adaptive SAM (ASAM) [16], an extension of SAM that is more robust to hyperparameters.

**DyTox+ and DyTox++ experiments** The computational overhead of ASAM is lower than more complex second-order methods, but it still doubles the number of forward and backward passes. For this reason, we didn’t include it in our main experiment but propose in Table 4 and Table 5 experiments on CIFAR100 and ImageNet100. The gain provided by MixUp then ASAM on our model (DyTox++) leads to a consistent improvement of +4.7% in “Avg” compared to the previous State-of-the-Art DER [28] on CIFAR100 50 steps (Table 4 and +4.6% on ImageNet100 10 steps (Table 5). Future works could consider the promising Look-SAM [19] to reduce the time overhead.

**Training procedure introspection** In Table 6, we compare DyTox+ and DyTox++ on CIFAR100 in a joint setting (no continual) and in a continual setting with 50 steps. In the joint setting, our model slightly benefits from both MixUp and ASAM: the gain is limited (+1.79 p.p.). On the other hand, those two methods greatly improve the extreme continual setting of 50 steps (+6.42 p.p.). This shows that the gain is not due to absolute improvements of the model performance. Moreover, using the Chaudhry et al.’s forgetting [3] measure, we compare how much a model has forgotten relatively to its previous tasks. This metric is therefore agnostic to absolute performance improvements. DyTox had a forgetting of 33.15%, DyTox+ of 31.50%, and DyTox++ of 30.47%: a total reduction of 2.68 p.p. This validates our novel training procedures that are particularly efficient for continual learning.
Our model is the first application of transformers for continual computer vision. A key component of the transformer architecture is the patch tokenizer. The number of patch tokens in an image is determined by the patch size: a larger patch size means less tokens, and vice-versa. We wondered about the effect of the patch size on forgetting and tested three different kinds of patch sizes in Table 7. Echoing results in vision transformers [5, 25], a smaller patch size (4 vs. 8 and 16) performs best in a joint training. However, the forgetting defined by Chaudhry et al. [3] is relatively similar, with 33.15% for a patch size of 4, and 33.20% for a patch size of 16. Therefore, we argue that the transformer architecture is hardly sensitive to the patch resolution w.r.t. its forgetting in continual learning.

A.4. Patch size effect on forgetting

Table 4: Results on CIFAR100 averaged over three different class orders. WA and DER w/o P results are reported from [28]. DyTox++ uses MixUp in addition of the DyTox strategy, DyTox further adds a sharpness-aware minimization [16].

Table 5: Results on ImageNet-100 with 10 steps of 10 new classes each. WA and DER w/o P results are reported from [28]. DyTox+ uses MixUp in addition of the DyTox strategy, DyTox++ further adds a sharpness-aware minimization.

Table 6: “Last” accuracy and forgetting [3] on CIFAR100 for the joint (1 step, no continual) and 50 steps settings.

A.5. ResNet backbone

DyTox is made of two main components: the SABs and the unique TAB. The TAB structure, taking as input both patch tokens and a task token, is crucial to our strategy. Yet, the SAB could be of any kind of features extractor, based on convolutions or attentions. Following the hybrid network proposed in ablations by Dosovitskiy et al. [5], we tried to replace the collection of SABs by a ResNet18. The final features of the ResNet, before global pooling, of shape \((W \times H \times D)\) can be seen as \(W \times H\) tokens of \(D\) dimension. We made a few modifications to this ResNet to boost its performance, namely removed the fourth and ultimate layer, and added a pointwise convolution with 504 output channels (so it can be divided by the 12 attention heads of the TAB), a batch normalization [12], and a ReLU activation. These simple modifications are sufficient for our proof of concept, and thus we also didn’t tune deeply this model. We display in Table 8 the comparison of the two backbones on CIFAR100 50 steps: (1) with ResNet, and (2) with SABs (DyTox). Performances are slightly lower than DyTox with SABs, however, they are still significantly higher than previous state-of-the-art like WA [31], especially in “Last” accuracy. Moreover, the parameters count is comparable to DyTox. This experiment shows that our DyTox framework, while designed with a transformer backbone in mind, is also efficient on non-token-based architectures such as a ResNet.


<table>
<thead>
<tr>
<th>Encoder</th>
<th>#P</th>
<th>Avg</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>10.68</td>
<td>68.53</td>
<td>50.05</td>
</tr>
<tr>
<td>SABs</td>
<td>10.77</td>
<td>70.20</td>
<td>52.34</td>
</tr>
</tbody>
</table>

Table 8: Hybrid network on CIFAR100 50 steps. While the features extractor is made of SABs in DyTox, here we instead use a modified ResNet18. Our framework still works well with a convolution-based approach.

<table>
<thead>
<tr>
<th>Task decoder</th>
<th>CIFAR100 Top-1</th>
<th>ImageNet100 Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg</td>
<td>Last</td>
</tr>
<tr>
<td>Residual Adapters</td>
<td>70.00</td>
<td>52.38</td>
</tr>
<tr>
<td>FiLM [20]</td>
<td>69.42</td>
<td>54.05</td>
</tr>
<tr>
<td>TAB (ours)</td>
<td>70.20</td>
<td>52.34</td>
</tr>
</tbody>
</table>

Table 9: Alternative task conditioner on CIFAR100 50 steps and ImageNet100 10 steps. While the simpler Residual Adapters and FiLM perform similarly to our TAB on CIFAR100, they forget significantly more on the complex ImageNet100.

A.6. Alternative task decoders

We investigate here other approaches for conditioning features to the different tasks. Residual Adapters [22] adds a different residual branch made of a pointwise convolution for each domain the model is learned (e.g., CIFAR then ImageNet then SVHN). This model needs the task/dataset/domain identifier at test-time to determine which residual to use. For VQA task [1], FiLM [20] proposes to modify the visual features using the textual query.

We adapt these two feature conditioning strategies for our transformer backbone architecture. We perform a global token pooling after the ultimate SAB, and apply for each learned task, a residual adapter or a FiLM. Residual adapter in our case is a MLP, and FiLM parameters are directly learned. As for DyTox, we forward each task-specific embedding to the respective task-specific classifier. We showcase the continual performance in Table 9 on CIFAR100 50 steps and ImageNet100 10 steps. On the former dataset, smaller and easier, the residual adapters and FiLM have similar performance as our TAB approach. On the other hand, as soon as the task complexity increases with the more detailed ImageNet100 dataset, FiLM and Residual adapter based conditioning strategies forget significantly more than our complete DyTox framework: TAB outperform the Residual Adapters by +2.98 p.p in “Last” top-5 accuracy and FiLM by +6.58 p.p.

References


Shienergy Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. (pages 1, 2, 3).