DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion
Matthieu Cord, Arthur Douillard, Alexandre Ramé, Guillaume Couairon

To cite this version:
Matthieu Cord, Arthur Douillard, Alexandre Ramé, Guillaume Couairon. DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion. IEEE/CVF Conference on Computer Vision and Pattern Recognition, Jun 2022, New Orleans, United States. 10.1109/CVPR52688.2022.00907 . hal-03997873

HAL Id: hal-03997873
https://hal.science/hal-03997873
Submitted on 20 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion

Arthur Douillard1,2, Alexandre Ramé1, Guillaume Couairon1,3, Matthieu Cord1,4
1Sorbonne Université, 2Heuritech, 3Meta AI, 4 valeo.ai
arthur.douillard@heuritech.com, {alexandre.rame, matthieu.cord}@sorbonne-universite.fr, gcouairon@fb.com

Abstract

Deep network architectures struggle to continually learn new tasks without forgetting the previous tasks. A recent trend indicates that dynamic architectures based on an expansion of the parameters can reduce catastrophic forgetting efficiently in continual learning. However, existing approaches often require a task identifier at test-time, need complex tuning to balance the growing number of parameters, and barely share any information across tasks. As a result, they struggle to scale to a large number of tasks without significant overhead.

In this paper, we propose a transformer architecture based on a dedicated encoder/decoder framework. Critically, the encoder and decoder are shared among all tasks. Through a dynamic expansion of special tokens, we specialize each forward of our decoder network on a task distribution. Our strategy scales to a large number of tasks while having negligible memory and time overheads due to strict control of the expansion of the parameters. Moreover, this efficient strategy doesn’t need any hyperparameter tuning to control the network’s expansion. Our model reaches excellent results on CIFAR100 and state-of-the-art performances on the large-scale ImageNet100 and ImageNet1000 while having fewer parameters than concurrent dynamic frameworks.1

1. Introduction

Most of the deep learning literature focuses on learning a model on a fixed dataset. However, real-world data constantly evolve through time, leading to ever-changing distributions: i.e., new classes or domains appeared. When a model loses access to previous classes data (e.g., for privacy reasons) and is fine-tuned on new classes data, it catastrophically forgets the old distribution. Continual learning models aim at balancing a rigidity/plasticity trade-off where old data are not forgotten (rigidity to changes) while learning new incoming data (plasticity to adapt). Despite recent advances, it is still an open challenge.

A growing amount of efforts have emerged to tackle catastrophic forgetting [49, 34, 63, 29, 18, 64]. Recent works [65, 39, 30, 21, 24, 54] dynamically expand the network architectures [65, 39] or re-arrange their internal structures [21, 54, 30, 24]. Unfortunately at test-time, they require to know the task to which the test sample belongs — in order to know which parameters should be used. More recently, DER [64] and Simple-DER [41] discarded the need for this task identifier by learning a single classifier on the concatenation of all produced embeddings by different sub-
sets of parameters. Yet, these strategies induce dramatic memory overhead when tackling a large number of tasks, and thus need complex pruning as post-processing.

To improve the ease of use of continual learning frameworks for real-world applications, we aim to design a dynamically expandable representation (almost) `for free` by having the three following properties: #1 limited memory overhead as the number of tasks grows, #2 limited time overhead at test time and #3 no setting-specific hyperparameters for improved robustness when faced to an unknown (potentially large) number of tasks.

To this end, we leverage the computer vision transformer ViT [15]. Transformers [60] offer a very interesting framework to satisfy the previously mentioned constraints. Indeed, we build upon this architecture to design a encoder/decoder strategy: the encoder layers are shared among all members of our dynamic network; the unique decoder layer is also shared but its forward pass is specialized by a task-specific learned token to produce task-specific embeddings. Thus, the memory growth of the dynamic network is extremely limited: only a 384d vector per task, validating property #1. Moreover, this requires no hyperparameter tuning (property #3). Finally, the decoder is explicitly designed to be computationally lightweight (satisfying property #2). We nicknamed our framework, DyTox, for DYNAMIC TOken eXpansion. To the best of our knowledge, we are the first to apply the transformer architecture to continual computer vision.

Our strategy is robust to different settings, and can easily scale to a large number of tasks. In particular, we validate the efficiency of our approach on CIFAR100, ImageNet100, and ImageNet1000 (displayed on Fig. 1) for multiple settings. We reach state-of-the-art results, with only a small overhead thanks to our efficient dynamic strategy.

2. Related work

Continual learning models tackle the catastrophic forgetting of the old classes [56, 22]. In computer vision, most of continual learning strategies applied on large-scale datasets use rehearsal learning: a limited amount of the training data of old classes is kept during training [50]. This data is usually kept in raw form (e.g., pixels) [49, 4, 9] but can also be compressed [26, 31], or trimmed [17] to reduce memory overhead; others store only a model to generate new samples of past classes [33, 55, 38]. In addition, most approaches aim at limiting the changes in the model when new classes are learned. These constraints can be directly applied on the weights [34, 66, 1, 7], intermediary features [29, 14, 69, 18, 16], prediction probabilities [40, 49, 4, 5], or on the gradients [43, 8, 20, 52]. All these constraint-based methods use the same static network architectures which doesn’t evolve through time, usually a ResNet [27], a LeNet [36], or a small MLP.

Continual dynamic networks In contrast, our paper and others focus on designing dynamic architectures that best handle a growing training distribution [65, 39], in particular by dynamically creating (sub-)members each specialized in one specific task [21, 24, 30, 51, 10, 61]. Unfortunately, previous approaches often require the sample’s task identifier at test-time to select the right subset of parameters. We argue this is an unrealistic assumption in a real-life situation where new samples could come from any task. Recently, DER [64] proposed a dynamic expansion of the representation by adding a new feature extractor per task. All extractors’ embeddings would then be concatenated and fed to a unified classifier, discarding the need for a task identifier at test-time. To limit an explosion in the number of parameters, they aggressively prune each model after each task using the HAT [54] procedure. Unfortunately, the pruning is hyperparameter sensitive. Therefore, hyperparameters are tuned differently on each experiment: for example, learning a dataset in 10 steps or in 50 steps use different hyperparameters. While being impracticable, it is also unrealistic because the number of classes is not known in advance in a true continual situation. Simple-DER [41] also uses multiple extractors, but its pruning method doesn’t need any hyperparameters; the negative counterpart is that Simple-DER controls less the parameter growth (2.5x higher than a base model). In contrast, we propose a framework dedicated to continual learning that seamlessly enables a task-dynamic strategy, efficient on all settings, without any setting-dependant modification and at almost no memory overhead. We share early class-agnostic [45] layers similarly to TreeNets [37] and base our strategy on the Transformer architecture.

Transformers were first introduced for machine translation [60], with the now famous self-attention. While the original transformer was made of encoder and decoder layers, later transformers starting from BERT [13] used a succession of identical encoder blocks. Then, ViT [15] proposed to apply transformers to computer vision by using patches of pixels as tokens. Multiple recent works, including DeiT [58], CaiT [59], ConViT [11], and Swin [42], improved ViT with architecture and training procedures modifications. PerceiverIO [32] proposed a general architecture whose output is adapted to different modalities using specific learned tokens, and whose computation is reduced using a small number of latent tokens. Despite being successful across various benchmarks, transformers have not yet been considered for continual computer vision to the best of our knowledge. Yet, we don’t use the transformer architecture for its own sake, but rather because of the intrinsic properties of transformers; in particular, the seminal encoder/decoder framework allows us to build an efficient architecture with strong capabilities against catastrophic forgetting.
3. DyTox transformer model

Our goal is to learn a unified model that will classify an increasingly growing number of classes, introduced in a fixed amount of steps $T$. At a given step $t \in \{1 \ldots T\}$, the model is exposed to new data belonging to new classes. Specifically, it learns from samples $\{(x^t_i, y^t_i)\}_i$, where $x^t_i$ is the $i$-th image of this task $t$ and $y^t_i$ is the associated label within the label set $C^t$. All task label sets are exclusive: $C^0 \cap C^1 \ldots C^T = \emptyset$. The main challenge is that the data are fully available only temporarily: following most previous works, only a few samples from previous tasks $\{1 \ldots t-1\}$ are available for training at step $t$ as rehearsing data. Yet, the model should remain able to classify test data coming from all seen classes $C^{1:t}$. A table of notations is provided in the supplementary materials.

The Fig. 2 displays our DyTox framework, which is made of several components (SAB, TAB, and Task Tokens) that we describe in the following sections.

3.1. Background

The vision transformer [15] has three main components: the patch tokenizer, the encoder made of Self-Attention Blocks, and the classifier.

Patch tokenizer The fixed-size input RGB image is cropped into $N$ patches of equal dimensions and then projected with a linear layer to a dimension $D$. Both operations, the cropping and projection, are done with a single 2D convolution whose kernel size is equal to its stride size. The resulting tensor $x_0 \in \mathbb{R}^{N \times D}$ is extended with a learned class token $x_{cls} \in \mathbb{R}^{D}$ resulting in a tensor of shape $\mathbb{R}^{(N+1) \times D}$. Following [23], a learned positional embedding $p \in \mathbb{R}^{(N+1) \times D}$ is added (element-wise).

Self-Attention (SA) based encoder The tokens are fed to a stack of transformer blocks that we denote here as Self-Attention Blocks (SABs):

\[
\begin{align*}
x^l_0 &= x_l + \text{SA}_{l} (\text{Norm}_{l,1} (x_l)) , \\
x_{l+1} &= x^l_0 + \text{MLP}_{l} (\text{Norm}_{l,2} (x^l_0)) ,
\end{align*}
\]

with SA a Self-Attention layer [60], Norm a layer normalization [2], and MLP a Multi-Layer Perceptron with a single hidden layer. We repeat these operations for each SAB, from $l = 1$ to $l = L$. The resulting tensor (which keeps the same dimension after every block) is $x_L \in \mathbb{R}^{(N+1) \times D}$. We display a visual illustration of a SA Block in Fig. 3.
3.2. Task-Attention Block (TAB)

Contrary to previous transformer architectures, we don’t have a class token, but rather what we nickname “task tokens”; the learned token of the \(i^{th}\) task is denoted \(\theta_i\). This special token will only be added at the last block. To exploit this task token, we define a new attention layer, that we call Special Attention Block (SAB). The overall structure of the DyTox strategy is illustrated in Fig. 2. We also show in Algo. 1 the pseudo-code of a forward pass at test-time after having learned the task \(t\). Critically, the test image can belong to any of the previously seen tasks \(\{1 \ldots t\}\). Our dynamic task token expansion is more efficient than a naive parameter expansion that would create a new copy of the whole network for each new task.

The memory overhead is thus almost null. (2) The computationally intensive blocks (i.e., the SABs) are executed

As detailed, the TAB has two inputs: the patch tokens \(x_L\) extracted from the image and a learned task-token \(\theta_i\). We’ll now detail how our framework evolves in a continual situation at each new step.

During the first step, there is only one task token \(\theta_1\). At each new step, we propose to expand our parameter space by creating a new task token while keeping the previous ones. Thus, after \(t\) steps, we have \(t\) task tokens (\(\theta_i\) for \(i \in \{1 \ldots t\}\)). Given an image \(x\) — belonging to any of the seen tasks \(\{1 \ldots t\}\) — our model tokenizes it into \(x_0\), and processes it through the multiple SABs: this outputs the patch tokens \(x_L\). Finally, our framework does as many forward passes through the TAB as there are tasks: critically, each TAB forward pass is executed with a different task token \(\theta_i\), resulting in different task-specific forwards, each producing the task-specific embeddings \(e_i\) (see Fig. 2):

\[
e_1 = \text{TAB}(\theta_1, x_L) ,
\]

\[
e_2 = \text{TAB}(\theta_2, x_L) ,
\]

\[
\vdots
\]

\[
e_t = \text{TAB}(\theta_t, x_L) .
\]

Rather than concatenating all embeddings \(\{e_1, e_2, \ldots, e_t\}\) together and feeding them to one classifier, we leverage task-specific classifiers. Each classifier \(\text{clf}_i\) is made of a Norm\(_i\) and a linear projection parametrized by \(\{W_i, b_i\}\), with \(W_i \in \mathbb{R}^{c_i \times d}\) and \(b \in \mathbb{R}^{c_i}\). It takes as input its task-specific embedding \(e_i\) and returns:

\[
y_i = \text{clf}_i(e_i) = \sigma(W_i \text{Norm}_i e_i + b_i) ,
\]

the predictions for the classes \(y_i \in \mathcal{C}^i\), where \(\sigma(x) = 1/(1+e^{-x})\) is the sigmoid activation. In comparison with the softmax activation, the element-wise sigmoid activation reduces the overconfidence in recent classes. Consequently, the model is better calibrated, which is an important attribute of continual model [3, 63, 68]. The loss is the binary-cross entropy. The independent classifiers paradigm coupled with the sigmoid activation and binary cross-entropy loss exclude explicitly a late fusion [48] of the task embeddings resulting in more specialized classifiers.

3.3. Dynamic task token expansion

We defined in the previous section our base network, made of a succession of SABs and ended by a single TAB.
only once despite learning multiple tasks. In contrast, the TAB has as many forwards as there are tasks. Though, this induces minimal overhead because the Task-Attention has a linear complexity w.r.t the number of patches while the Self-Attention is quadratic. Therefore, the time overhead is sub-linear. We quantitatively show this in Sec. 4.

### Context
The current transformer paradigm starting from BERT [13] and continuing with ViT [15] is based on an encoder+classifier structure. Differently, our dynamic framework strays is a resurgence of the encoder/decoder structure of the original transformer [60]: the encoder is shared (both in memory and execution) for all outputs. The decoder parameters are also shared, but its execution is task-specific with each task token, with each forward akin to a task-specific expert chosen from a mixture of experts [44]. Moreover, multi-tasks text-based transformers have natural language tokens as an indicator of a task [46] (e.g. "summarize the following text"), in our context of vision we used our defined task tokens as indicators.

### Losses
Our model is trained with three losses: (1) the classification loss $\mathcal{L}_{\text{clf}}$, a binary-cross entropy, (2) a knowledge distillation [28] $\mathcal{L}_{kd}$ applied on the probabilities, and (3) the divergence loss $\mathcal{L}_{\text{div}}$. The distillation loss helps to reduce forgetting. It is arguably quite naive, and more complex distillation losses [53, 29, 18] could further improve results. The divergence loss, inspired from the “auxiliary classifier” of DER [64], uses the current last task’s embedding $e_i$ to predict $(|C^t| + 1)$ probabilities: the current last task’s classes $C^t$ and an extra class representing all previous classes that can be encountered via rehearsal. This classifier is discarded at test-time and encourages a better diversity among task tokens. The total loss is:

$$\mathcal{L} = (1 - \alpha)\mathcal{L}_{\text{clf}} + \alpha\mathcal{L}_{kd} + \lambda\mathcal{L}_{\text{div}},$$ (9)

with $\lambda$ a hyperparameter set to 0.1 for all experiments. $\alpha$ correspond to the fraction of the number of old classes over the number of new classes $\frac{|C^t| + 1}{|C^{t-1}|}$ as done by [68]. Therefore, $\alpha$ is automatically set; this removes the need to finely tune this hyperparameter.

### Algorithm 1
DyTox’s forward pass at step $t$

| Input: $x_0$ (initial patch tokens), $y$ (ground-truth labels) |
| Output: $\hat{y}_{1:t}$ (predictions for all classes of $C^{1:t}$) |
| 1: $x_L \leftarrow \text{SAB}_{t=L} \circ \ldots \circ \text{SAB}_{t=1}(x_0)$ |
| 2: for $i \leftarrow 1$; $i \leq t$; $i++$ do |
| 3: $e_i \leftarrow \text{TAB}([\theta_i, x_L])$ |
| 4: $\hat{y}_i \leftarrow \text{clf}_{i}(e_i)$ |
| 5: end for |
| 6: $\hat{y}_{1:t} \leftarrow [\hat{y}_1, \ldots, \hat{y}_t]$ |

Table 1: DyTox’s architectures for CIFAR and ImageNet. The only difference between the two architectures is the patch size, as the image sizes vary between datasets.

### 4. Experiments

#### 4.1. Benchmarks & implementation

**Benchmarks & Metrics** We evaluate our model on CIFAR100 [35], ImageNet100 and ImageNet1000 [12] (descriptions in the supplementary materials) under different settings. The standard continual scenario in ImageNet has 10 steps: thus we add 10 new classes per step in ImageNet100, and 100 new classes per step in ImageNet1000. In CIFAR100, we compare performances on 10 steps (10 new classes per step), 20 steps (5 new classes per step), and 50 steps (2 new classes per step). In addition to the top-1 accuracy, we also compare the top-5 accuracy on ImageNet. We report the “Avg” accuracy which is the average of the accuracies after each step as defined by [49]. We also report the final accuracy after the last step (“Last”). Finally, in our tables, “#P” denotes the parameters count in million after the final step.

**Implementation details** As highlighted in Table 1, our network has the same structure across all tasks. Specifically, we use 5 Self-Attention Blocks (SABs), 1 Task-Attention Block (TAB). All 6 have an embedding dimension of 384 and 12 attention heads. We designed this shallow transformer to have a comparable parameters count to other baselines, but also made it wider than usual "tiny" models [15, 58, 59]. We tuned all hyperparameters for CIFAR100 with 10 steps on a validation set made of 10% of the training set, and then kept them fixed for all other settings, ImageNet included. The only difference between the two datasets is that ImageNet images are larger; thus the patch size is larger, and overall the base transformer has slightly more parameters on ImageNet than on CIFAR (11.00M vs 10.72M) because of a bigger positional embedding. We use the attention with spatial prior (introduced by ConViT [11]) for all SABs which allows training transformers on a small dataset (like CIFAR) without pretraining on large datasets or complex regularizations. Following previous works [49, 64], we use for all models (baselines included) 2,000 images of rehearsal memory for CIFAR100.
Table 2: Results on ImageNet-100 and ImageNet-1000 datasets, learned with 10 steps of respectively 10 and 100 new classes. E2E [4] and Simple-DER [41] results come from their respective papers, and used a different class ordering. Other results come from [64]. The † symbol means that [64] needed setting-sensitive hyperparameters. Moreover, its reported parameters count was an average over all steps ([64] reported 14.52M on ImageNet1000): the final parameters count (necessarily higher) was not available.

4.2. Quantitative results

ImageNet We report performances in Table 2 on the complex ImageNet dataset. The † marks the DER with setting-specific pruning, and DER w/o P is for the DER without pruning. In ImageNet100, DyTox reaches 69.10% and outperforms DER† by +3.04 percentage points (p.p) in “Last” top-1 accuracy. Though, DyTox and DER w/o P somehow perform similarly in “Avg” accuracy on this setup, as highlighted in the performance evolution displayed in Fig. 4. Most importantly, on the larger-scale ImageNet1000, DyTox systematically performs best on all metrics despite having lower parameters count. Specifically, DyTox reaches 71.29% in “Avg” top-1 accuracy, and 63.34% in “Last” top-1 accuracy. This outperforms the previous state-of-the-art DER w/o P (68.84% in “Avg”, 60.16% in “Last”) which has 10 ResNet18 in parallel and 116.89M parameters. Compared to the pruned DER†, DyTox has a +4.56 p.p in top-1 and a +1.51 p.p in top-5 for the “Avg” accuracy. All models evolutions on ImageNet1000 are illustrated in Fig. 1: DyTox constantly surpasses previous state-of-the-art models — despite having a comparable performance at the first step and fewer parameters.

DyTox is able to scale correctly while handling seamlessly the parameter growth by sharing most of the weights across tasks. In contrast, DER had to propose a complex pruning method; unfortunately, this pruning required different hyperparameter values for different settings. Despite this, the pruning in DER† is less efficient when classes diversity increase: DER† doubles in size between

and ImageNet100, and 20,000 images for ImageNet1000. The implementations of the continual scenarios are provided by Continuum [19]. Our network implementation is based on the DeiT [58] code base which itself uses extensively the timm library [62]. The code is released publicly².

The full implementation details are in the appendix.

²https://github.com/arthurdouillard/dytox

Figure 4: Performance evolution on ImageNet100. The top-5 accuracy (%) is reported after learning each task. Our model DyTox (in red) surpasses significantly most baselines, and is of equal performance as the complex DER that uses pruning with setting-specific hyperparameters.
### Table 3: Results on CIFAR100 averaged over three different class orders. Baselines results are come from [64]. The † symbol means that [64] needed setting-sensitive hyperparameters. Moreover, its reported parameters count was an average over all steps: the final parameters count (necessarily higher) was not available.

<table>
<thead>
<tr>
<th>Methods</th>
<th>#P</th>
<th>Avg</th>
<th>Last</th>
<th>#P</th>
<th>Avg</th>
<th>Last</th>
<th>#P</th>
<th>Avg</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet18 Joint</td>
<td>11.22</td>
<td>80.41</td>
<td></td>
<td>11.22</td>
<td>81.49</td>
<td></td>
<td>11.22</td>
<td>81.74</td>
<td></td>
</tr>
<tr>
<td>Transf. Joint</td>
<td>10.72</td>
<td>76.12</td>
<td></td>
<td>10.72</td>
<td>76.12</td>
<td></td>
<td>10.72</td>
<td>76.12</td>
<td></td>
</tr>
<tr>
<td>iCaRL [49]</td>
<td>11.22</td>
<td>65.27±1.02</td>
<td>50.74</td>
<td>11.22</td>
<td>61.20±0.83</td>
<td>43.75</td>
<td>11.22</td>
<td>56.08±1.02</td>
<td>36.62</td>
</tr>
<tr>
<td>PODNet [18]</td>
<td>11.22</td>
<td>58.03±1.27</td>
<td>41.05</td>
<td>11.22</td>
<td>53.97±1.27</td>
<td>35.02</td>
<td>11.22</td>
<td>51.19±1.02</td>
<td>32.99</td>
</tr>
<tr>
<td>UCIR [29]</td>
<td>11.22</td>
<td>68.80±0.71</td>
<td>53.54</td>
<td>11.22</td>
<td>66.48±0.32</td>
<td>47.02</td>
<td>11.22</td>
<td>62.09±0.85</td>
<td>41.04</td>
</tr>
<tr>
<td>WA [68]</td>
<td>11.22</td>
<td>69.46±0.29</td>
<td>53.78</td>
<td>11.22</td>
<td>67.33±0.15</td>
<td>47.31</td>
<td>11.22</td>
<td>64.32±0.28</td>
<td>42.14</td>
</tr>
<tr>
<td>BiC [63]</td>
<td>11.22</td>
<td>68.60±0.60</td>
<td>57.05</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>RPSNet [47]</td>
<td>11.22</td>
<td>68.60±0.60</td>
<td>57.05</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DER w/o P [64]</td>
<td>11.22</td>
<td>75.36±0.36</td>
<td>65.22</td>
<td>224.55</td>
<td>74.09±0.33</td>
<td>62.48</td>
<td>561.39</td>
<td>72.41±0.36</td>
<td>59.08</td>
</tr>
<tr>
<td>DER † [64]</td>
<td>-</td>
<td>74.64±0.23</td>
<td>64.35</td>
<td>-</td>
<td>73.98±0.36</td>
<td>62.55</td>
<td>-</td>
<td>72.05±0.55</td>
<td>59.76</td>
</tr>
<tr>
<td>DyTox</td>
<td>10.73</td>
<td>73.66±0.02</td>
<td>60.67±0.34</td>
<td>10.74</td>
<td>72.27±0.18</td>
<td>56.32±0.61</td>
<td>10.77</td>
<td>70.20±0.16</td>
<td>52.34±0.26</td>
</tr>
<tr>
<td>DyTox+</td>
<td>10.73</td>
<td>75.54±0.10</td>
<td>62.06±0.25</td>
<td>10.74</td>
<td>75.04±0.11</td>
<td>60.03±0.45</td>
<td>10.77</td>
<td>74.35±0.05</td>
<td>57.09±0.13</td>
</tr>
</tbody>
</table>

Figure 5: Performance evolution on CIFAR100. The top-1 accuracy (%) is reported after learning each task. Left is evaluated with 10 steps, middle with 20 steps, and right with 50 steps.

CIFAR100 Table 3 shows results for all approaches on CIFAR100. The more steps there are, the larger the forgetting is and thus the lower the performances are. Those settings are also displayed in Fig. 5 after each task. In every setting, DyTox is close to DER w/o P for much fewer parameters (up to 52x less). Critically, DyTox is significantly above other baselines: e.g. DyTox is up to +25% in “Last” accuracy in the 50 steps setup.

**Improved training procedure** To bridge the gap between DyTox and DER w/o P on CIFAR100, we introduce a new efficient training procedure for continual learning. Using MixUp [67], we linear interpolate new samples with existing samples. The interpolation factor $\lambda \sim \text{Beta}(\alpha, \alpha)$ is sampled with $\alpha = 0.8$: the pixels of two images are mixed ($x = \lambda x_1 + (1-\lambda)x_2$) as their labels ($y = \lambda y_1 + (1-\lambda)y_2$). MixUp was shown to have two main effects: (1) it diversifies the training images and thus enlarges the training distribution on the vicinity of each training sample [6] and (2) it improves the network calibration [25, 57], reducing the overconfidence in recent classes. Thus MixUp has shared motivation with the sigmoid activation. When Dy-
Model ablations

We ablate the importance of the different classifiers. All three greatly improve over the baseline:

1. a divergence classifier, and
2. independent classifiers reduces the Chaudhry et al.'s forgetting [7] by more than 24%.

4.3. Model introspection on CIFAR100

Memory overhead

We only add a vector of size $d = 384$ per task; thus, the overhead in memory (not considering the growing classifier which is common for all continual models) is only of $+0.004\%$ per step. Even in the challenging setting of CIFAR100 with 50 tasks, our memory overhead is almost null ($+0.2\%$).

Computational overhead

The vast majority of the computation is done in the SABs, thus shared among all tasks. The dynamical component of our model is located at the ultimate TAB. Moreover, the Task-Attention, contrary to the Self-Attention, has a time complexity linear in terms of tokens and not quadratic reducing the time overhead to an acceptable sub-linear amount. Overall, for each new task, one forward pass takes 2.24% more time than for the base transformer.

Training procedure introspection

Our DyTox+ strategy with MixUp really reduces catastrophic forgetting and does not just improve raw performances. This is shown in Table 4, where we compare DyTox vs. DyTox+ strategies on CIFAR100. While MixUp only slightly improves by 1.39 $p.p$ the accuracy in joint learning (no continual, 1 step), MixUp greatly improves the performance by 4.75 $p.p$ in the 50 steps continual scenario. To further illustrate this, we also report the Chaudhry et al.'s forgetting [7] measure which compares how performances dropped compared to previous steps. MixUp reduces this forgetting by 1.65 $p.p$.

Model ablations

We ablate the importance of the different components of DyTox in Table 5. We add on the base transformer a naive knowledge distillation [28] and a fine-tuning [4, 29, 18, 64] applied after each task on a balanced set of new data and rehearsal data. Finally, our DyTox strategy exploits directly the very nature of transformers (separated task information from the pixels information) to tackle catastrophic forgetting with three components: (1) a task token expansion, (2) a divergence classifier, and (3) independent classifiers. All three greatly improve over the baseline transformer ($42.21\% \rightarrow 52.34\%$ in “Last”) while having almost no memory overhead ($+0.2\%$). The divergence classifier improves the diversity between task tokens: we observed that the minimal Euclidean distance between them increases by 8%. Moreover, we also remarked that having independent classifiers reduces the Chaudhry et al.’s forgetting [7] by more than 24%.

5. Conclusion

In this paper, we propose DyTox, a new dynamic strategy for continual learning based on transformer architecture. In our model, self-attention layers are shared across all tasks, and we add task-specific tokens to achieve task-specialized embeddings through a new task-attention layer. This architecture allows us to dynamically process new tasks with very little memory overhead and does not require complex hyperparameter tuning. Our experiments show that our framework scales to large datasets like ImageNet1k with state-of-the-art performances. Moreover, when a large number of tasks is considered (i.e. CIFAR100 50 steps) our number of parameters increases reasonably contrary to previous dynamic strategies.

Limitations: True continual learning aims at learning an almost unlimited number of tasks with low forgetting. No current approaches are yet able to do so. Thus, forgetting is not yet solved for continual learning but our model is a step forward in that direction.

Broader impact: Machine learning models often are biased, with some classes suffering from lower performances. Studying forgetting in continual learning provides insights about the difference in performances between classes. Our task-specialized model could help reduce these biases.

Acknowledgments: This work was partly supported by ANR grant VISA DEEP (ANR-20-CHIA-0022), and the HPC resources of IDRIS AD011011706.
References


[64] Shipeng 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, 5, 6, 7, 8).
