Report of Choco_Leibniz Team
SSLAD Competition Track 3A (ICCV 2021)

Gabriele Graffieti, Guido Borghi, Davide Maltoni, Matteo Ferrara
Department of Computer Science and Engineering (DISI)
University of Bologna
Via dell'Università 50, Cesena (FC), Italy
{name.surname}@unibo.it

1. Introduction

In this report, we describe the approach used to address the "ICCV 2021 Workshop SSLAD Track 3A - Continual Object Classification" competition. We divided our analysis into the main aspects on which we focused in the development of our solution.

2. Competition Requirements

The Track 3A of the competition is focused on continual object recognition. The dataset (see Sect. 2.1) is composed of 6 classes. The samples arrive at the model in a streaming fashion, maintaining the temporal coherence of the original data. The samples have to be processed at a maximum of 10 samples at a time, and the model needs to be available for evaluation at any time, without any additional computation between training and testing. Moreover, the data must be seen only once, so no additional epochs on the same experience are allowed, so once a group of 10 samples has been processed by the model, the same samples cannot be further exploited (except with a limited replay memory). Additional requirements are the following:

- The maximum number of parameters of the model is 25M, which is 105% of a ResNet-50 [3].
- The capacity of any sort of memory is limited to 1000 objects.
- The maximum size of a single batch is 10.
- Pretraining is allowed only on ImageNet.

2.1. Soda10M Dataset

The dataset used for the competition is the Soda10M dataset [2], a large-scale dataset collected for self-supervising learning and domain adaptation in autonomous driving. The original frames are collected every 10 seconds within 32 different cities under different weather conditions, from these frames several bounding boxes are cropped and used as input for the competition. In our opinion, among the specific challenges of the dataset, the main issues related to the competition, *i.e.* elements that can greatly influence the performance of the continual learning-based classifier, are the following:

- Occlusions: we note that a great number of objects depicted inside the patches are occluded, even by an object that belongs to the classes present in the Soda10M dataset (for instance, several buses are occluded by cars); therefore, these occlusions can easily fool the classifier's prediction.
- Image resolution: objects that are placed in the background far from the acquisition device may visually appear with a very low resolution once rescaled to the crop size of the competition (64×64).
- Bad quality images: images captured at night and under bad weather conditions are heavily affected by bad lighting and low contrast.

3. Model and Hyperparameters

We exploit the ResNet-50 [3] neural network as classification model. The last fully connected layer is substituted in order to fit the competition dataset, passing from 1000 (the number of classes on Imagenet) to 7 (the number of classes in the competition plus a null one) in final neurons. We add bias to maintain the architecture as similar as possible to the original network. We exploit a pretrained model on the ImageNet-1000 dataset [1]. The weights have been downloaded from the official torchvision website[1]. The reported model top-1 accuracy on ImageNet is 76.13%. We use the *Stochastic Gradient Descent* (SGD) optimizer with a learning rate of 10^{-2}, with momentum and weight decay

[1]https://pytorch.org/vision/stable/models.html

set to 0. As loss function, we use the *Cross Entropy* (CE) loss in the standard form, which can be expressed as:

$$\mathcal{L}(y, l) = -\log\left(\frac{\exp(y_l)}{\sum_j \exp(y_j)}\right), \tag{1}$$

where y is the output of the network (logits) while l is the index of the correct class. In addition to the CE loss of Equation 1, we add a weight α to each class, thus, the loss becomes ($\mathcal{L}(y, l) = \alpha_l \cdot \mathcal{L}(y, l)$). We set $\alpha_0 = 0$ and $\alpha_{1,\ldots,6} = 1$. The batch size is set to 10, the maximum value allowed by the competition.

4. Training procedure

The input batch is first transformed in a tensor of size (b, c, w, h), where $b = 10$ (batch size), $c = 3$ (RGB image channels), and $w = h = 64$ (the image dimensions, then the crop is squared). Then, the width and the height of the tensor are resized through the *Nearest Neighbour* interpolation, from 64×64 to 224×224 to resemble the original image size of the ImageNet dataset on which the model has been pretrained. Even though the appearance of the patches may be degraded by this operation (the original size of the input patches is about $^1/_{12}$ of the final size), we note that the convolutional kernels of the model respond better in resized images. This can be explained by the fact that ImageNet contains classes (such as trucks, cars, bicycles, etc.) similar to the ones present in the Soda10M dataset, thus the convolution filters have been tuned to extract strong features from those images. Images of the same size contain objects of about the same size (in pixels) so it is normal to think that the pretrained filters respond better if the original size is preserved.

After that, an "on-the-fly" data augmentation technique is applied, in order to increment the variety of the input tensors. Due to the characteristic of the benchmark, the input variety is limited since many similar images of the same objects are provided together to the network. The motivation is the streaming nature (*i.e.* the temporal order is preserved and original frames are acquired with a low frame rate) of the frames from which input patches are cropped. Therefore, before passing the tensor to the model, an horizontal flipping is applied and both the original and the flipped version are fed to the model.

Finally, only during the first experience, the resulting batch is put in temporary memory (of size 10) and passed twice through the model. After each forward step, the loss and the gradients are computed, and then the optimization procedure is applied. This operation is motivated by the observation that, in the first experience, the learning of the model (especially considering classes with few samples) can be boosted by seeing a larger number of (even identical) samples. This tries to imitate a traditional machine learning training procedure, in which the same images are fed to the network many times across the epochs.

5. Classification Head Protection

One of the main causes of catastrophic forgetting in neural networks is the so-called "learning in isolation" problem. This issue mainly arises when only a limited number of classes is present in each experience, or when some classes are underrepresented, *i.e.* they are present with a very low number of samples with respect to the other classes in the training data, leading to the forgetting of these classes. The forgetting especially occurs in the classification head, *i.e.* the last layer (usually a fully connected layer) of the adopted neural network.

In our experiments, we observed the same effect with the Soda10M dataset, probably due to the very limited presence (*e.g.* tricycle) or even the total absence (*e.g.* the second experience presents only cars, trucks, and trams) of certain classes. We address this issue using the CWR algorithm proposed in [4], where was presented as a baseline technique for continual learning from sequential batches. CWR maintains two sets of weights for the classification layer, namely *Consolidated Weights* (cw) and *Temporary Weights* (tw):

- cw: contains the weights from the previous experience that are used in the consolidation phase. In this phase, the cw weights are merged with the weights of the current experience tw;
- tw: contains the weights used to train the model in the current experience. The weights are initialized to 0 at the starting of the experience, and only the weights of the current experience's classes are loaded from cw.

Differently from the original paper, we do not freeze the weights of the feature extractor, as proposed in [5].

6. Replay Memory

Replay is known to be one of the most effective ways to contrast the catastrophic forgetting problem, especially in complex continual learning scenarios. Therefore, we decide to adopt this paradigm in our implementation, alongside CWR.

We created six different buffers of memory, one for each class. Each of these buffers is limited in size: in our final implementation, we set this limit to 100 for all classes, leading to a final memory size of 600 samples. We observe that including a larger number of samples per class or, alternatively, reducing the number of classes memorized (put only some classes in memory), do not produce any particular benefit. This is probably due to the following reasons: i) the total number of tricycles available in the training set is 82, lower than the fixed buffer size; ii) it is important to limit the number of replay samples for certain classes, such as car, that represent the large majority of input samples. Augmenting the class buffer size tends to produce a negative effect, since the number of tricycles is bounded by the training data (no more than 82) while the other classes fully occupy their buffer at the end of the first experience, making the replay memory more unbalanced. The combination of the two points above leads to a well-balanced memory class since, already starting from the second experience, the memory buffer is full (except for the tricycle class), and the imbalance is not too much marked. We also noticed that setting a different size for each class does not produce appreciable improvements.

We divide the minibatch (of size 10) into 5 samples from the current experience and 5 samples loaded from the replay memory. Using a minibatch composed in that way, the total number of minibatches doubles for each experience w.r.t. not using replay. The replay data is sampled randomly without replacement from the replay memory. When all the samples on the memory have been sampled, the sampling procedure is started again.

6.1. Memory Management

The memory is managed in the following way: as mentioned before, a buffer of dimension 100 is maintained for each class, for a total of 600 samples. During the first experience the memory is not used for the training, but the samples from the current experience are accumulated in the buffers. If a buffer for a specific class is not full and in the current minibatch are present patterns of the corresponding class, all the patterns are memorized in the buffer. If the buffer is full, we use *reservoir sampling* [6], which guarantees that every sample has the same probability to be in the replay memory at the end of the experience. We maintain a counter for each class that counts the samples seen so far, and we update the memory as follows:

```
1  $\mathcal{C}_c = 0 \ \ \forall c \in \{0, .., 6\}$
2  $\mathcal{M}_c \leftarrow \emptyset \ \ \forall c \in \{0, .., 6\}$
3  for $d, c$ in the training dataset do
4      $\mathcal{C}_c = \mathcal{C}_c + 1$
5      if $\mathcal{C}_c \leq 100$ then
6          $\mathcal{M}_c[\mathcal{C}_c] \leftarrow d$
7      else
8          $j = random\_int(1, \mathcal{C}_c)$
9          if $j \leq 100$ then
10             $\mathcal{M}_c[j] \leftarrow d$
11         end
12     end
13 end
```

Algorithm 1: Reservoir sampling

where d and c are the data and the class of a single sample in the minibatch, $\mathcal{C}$ is a counter for the number of samples viewed for each class, $\mathcal{M}$ is the replay memory, composed of a buffer of 100 elements for each class.

Since we do not want that the replay memory used in the current experience is altered by the insertion of samples from the current experience, we used, as a temporary buffer for the insertion, the remaining 400 free slots of the memory. For each new experience we inserted $^{100}/_{i}$ ($i =$ index of the current experience) samples from the current experience (so 100 from the first experience, 50 from the second experience, and so on). In the first experience, we use the memory $\mathcal{M}$ and the operations described in algorithm 1. For the subsequent experiences, we use an additional buffer $\mathcal{B}$ of dimension $^{100}/_{i}$ for each class. The pattern are inserted following algorithm 1, but using $\mathcal{B}$ instead of $\mathcal{M}$ and buffer dimension $^{100}/_{i}$ instead of 100. At the start of a new experience, we remove $^{100}/_{i}$ elements from each class from $\mathcal{M}$ and insert the new replay samples from $\mathcal{B}$. After that $\mathcal{B}$ is emptied. Note that the maximum allocation of memory is 900 samples, since the dimension of $\mathcal{M}$ is always 600, while the maximum dimension of $\mathcal{B}$ is 300 (50 samples per class during the second experience).

7. Contribution of the Components

In this section we we provide some comments about the contribution of each component of the proposed solution, evaluating them in a fairly high-level manner. We use three levels of importance, characterized by one *, two **, or three *** stars. Components that are the most important for our solution (in terms of final accuracy on the validation set) are evaluated using 3 stars, components that contributed in a minor manner to the final accuracy are evaluated with 2 stars, components that do not contribute much to the final accuracy (changing them only change the accuracy of less than 1 percentual point) are evaluated 1 star.

Continuous components (such as learning rate) are evaluated changing them in a reasonable manner around the found optimal value. Discrete components (such as the model) are evaluated against similar possibilities (*e.g.* models with a similar number of parameters and similar accuracy on ImageNet-1000). On/off components (such as image resizing) are evaluated based on the difference between using or not using the analyzed component. The contributions are summarized in Table 1.

The evaluations of the contribution are not totally objective, and the reported importance should not be intended as a guide for designing continual learning strategies. Nonetheless, we believe that this can be useful for the readers, and future works and research directions.

Component	Contribution
Model	**
Learning rate	*
Optimizer	***
Loss weights	*
Image resizing	***
Data augmentation	**
Double batch (1st exp.)	*
CWR	**
Not freezing the feature extractor	***
Replay memory	***
Balanced memory	***
Reservoir sampling	**

Table 1. The importance of the contribution of each component of the proposed solution based on the final accuracy on the validation set. 3 stars *** indicates maximum importance, 1 star * indicates limited importance.

8. Hardware and Video Memory Usage

We run our experiments on two machines. In both cases, the computation load is influenced by the use of the replay memory: this is due to the caching (in RAM) of the replay memory that does not need to be loaded from disk at every batch.

The first server is equipped with an *Intel Xeon E5-2650@2.00GHz* processor, 64 GB of RAM, and an *Nvidia 1080 Ti* (11 GB). In this setting, with our implementation, the training and validation phases run at about 3.6 it/s in the first experience and about 5.7 it/s from the second experience. The second machine is equipped with an *AMD EPYC 7282*, 256GB of RAM, and an *Nvidia Quadro RTX 5000* (16GB). In this setting, with our implementation, the training and validation phases run at about 3.9 it/s in the first experience and about 7.5 it/s from the second experience.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 1

[2] Jianhua Han, Xiwen Liang, Hang Xu, Kai Chen, Lanqing Hong, Chaoqiang Ye, Wei Zhang, Zhenguo Li, Chunjing Xu, and Xiaodan Liang. Soda10m: Towards large-scale object detection benchmark for autonomous driving. *arXiv preprint arXiv:2106.11118*, 2021. 1

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[4] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019. 2

[5] Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10203–10209. IEEE, 2020. 2

[6] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, Mar. 1985. 3