Small is Better: An Analysis of Instance Quantity/Quality Trade-off in Rehearsal-based Continual Learning

Francesco Pelosin
Ca' Foscari University
Venice, Italy
francesco.pelosin@unive.it

Andrea Torsello
Ca' Foscari University
Venice, Italy
andrea.torsello@unive.it

Abstract—The design of machines and algorithms capable of learning in a dynamically changing environment has become an increasingly topical problem with the increase of the size and heterogeneity of data available to learning systems. As a consequence, the key issue of Continual Learning has become that of addressing the stability-plasticity dilemma of connectionist systems, as they need to adapt their model without forgetting previously acquired knowledge. Within this context, rehearsal-based methods i.e., solutions in where the learner exploits memory to revisit past data, has proven to be very effective, leading to performance at state-of-the-art. In our study, we propose an analysis of the memory quantity/quality trade-off adopting various data reduction approaches to increase the number of instances storable in memory. In particular, we investigate complex instance compression techniques such as deep encoders, but also trivial approaches such as image resizing and linear dimensionality reduction. Our findings suggest that the optimal trade-off is severely skewed toward instance quantity, where rehearsal approaches with several heavily compressed instances easily outperform state-of-the-art approaches with the same amount of memory at their disposal. Further, in high memory configurations, deep approaches extracting spatial structure combined with extreme resizing (of the order of $8 \times 8$ images) yield the best results, while in memory-constrained configurations where deep approaches cannot be used due to their memory requirement in training, a variation of Extreme Learning Machines (ELM) offer a clear advantage. Code and experiments available at https://github.com/francesco-p/smaller-is-better

Index Terms—continual learning, dimensionality reduction, rehearsal

I. INTRODUCTION

Continual Learning (CL) is increasingly at the center of attention of the research community due to its promise of adapting to the dynamically changing environment resulting from the huge increase in size and heterogeneity of data available to learning systems. It has found applications in several domains. Its prime application, and still most active field, is computer vision, and in particular object detection [1]–[3]; however it has since found applications in several other domains such as segmentation [4]–[6], where each segmented class has to be learned in an incremental fashion, as well as in other fields, among which we mention Reinforcement Learning (RL) [7], [8] and Natural Language Processing (NLP) [9]–[11].

Ideally, the behaviour of CL systems should resemble human intelligence in its ability to incrementally learn in a dynamically environment [12], with minimal waste of resources, spatial or computational. The main problem encountered by these systems resides in the famous stability-plasticity dilemma of neuroscience, resulting in the so called catastrophic forgetting [13], a phenomenon where new information dislodges or corrupts previously learned knowledge, resulting in the deterioration of the ability to solve previously learned tasks.

Solutions to this problem typically incur in a increase in resource requirements [14] both for CL’s very nature (the more tasks arrive the more data the agent need to process), and for the nature of the systems that try to solve it, both in the increased complexity of the typically deep learning models, and in the time and space requirements of continuously learning multiple models. This problem become particularly evident in rehearsal-based methods.

Rehearsal-based methods, i.e., approaches that leverage a memory buffer to cope with catastrophic forgetting, are emerging as the most effective methodology to tackle CL. Their performance, backed by extensive empirical evidence [14], finds also a theoretical justification in Knoblauch and co-workers’ finding that optimally solving CL would require perfect memory of the past [15]. In fact, if we were able to completely re-train a new system with all previous data every time a new task arrives, Continual Learning would not appear to be any different from any other learning problem. However, this approach is both spatially and computationally infeasible for most real-world problems and we can argue it is precisely these memory and computational limitations that characterize CL and distinguish it from other learning problems.

Our investigation aims to analyze the trade-offs on limited-memory CL systems. In particular, we focus on the quantity/quality trade-off for memory instances. We do so through the analysis of several dimensionality-reduction schemes applied to data instances that allows us to increase the number of examples storable in our fixed-capacity memory. In particular we adopted deep learning encoders such as a variation of ResNet18 [16] and Variational Autoencoders (VAE) [17], the...
simple yet surprisingly effective extreme resizing of image data, and, lastly, we explored Random Projections for dimensionality reduction. The latter scheme turns out to be very effective in low memory scenarios also reducing the model’s parameter complexity. Indeed, we will show that a variation of Extreme Learning Machines (ELM) offers a simple yet effective solution for resources-constrained CL systems.

Our analysis will focus on computer vision tasks and use GDumb [18] as a rehearsal-baseline. GDumb is a model that has been proposed to question the community’s progress in CL thanks to the fact that in lieu of its outstanding simplicity, it was still able to provide state-of-the-art performance. Further, its simplicity also results in high versatility, as it proposes a general CL formulation comprising all task formulations in the literature. GDumb is fully rehearsal-based, and it is composed by a greedy sampler and a dumb learner, that is, the system does not introduce any particular strategy in the selection of replay data. Therefore, it represents the ideal candidate method to carry out our analysis.

The experimental findings highlighted in our paper are multiple: first, we show that when the memory buffer is fixed and extreme values of resizing of instance data is applied, we can easily push the state-of-the-art of CL rehearsal systems by a minimum of +6% to a maximum of +67% in terms of final accuracy. This surprising result suggests that the optimal trade-off between data quantity and quality is severely skewed toward the former and that in general the informational content required to correctly classify images in standard datasets is relatively low. Then, we analyze the consumption of resources of rehearsal CL systems as we saturate the rehearsal buffer, and show that ELM offer a clear solution on CL systems constrained by very low resources environments.

II. RELATED WORKS

Following some recent surveys [3], [12], [19], we divide CL approaches into three main categories: regularization-based approaches, data rehearsal-based approaches and architectural-based approaches. Although a few novel theoretical frameworks based on meta-learning have been introduced recently [12], the majority still fall within these categories (or in a mixture of them).

Regularization-based approaches address catastrophic forgetting by controlling each parameter’s importance through the subsequent tasks, by means of the addition of a finely-tuned regularizing loss criterion. Elastic Weight Consolidation (EWC) [20] was the first well established approach of this class. It uses Fisher information to estimate each parameter’s importance while discouraging the update for parameters with greatest task specificity. Learn without Forgetting (LwF) [21] exploits the concept of “knowledge distillation” to preserve and regularize the output for old tasks. More recently, Learning without Memorizing (LwM) [22] adds in the loss an information preserving penalty exploiting attention maps, Continual Bayesian Neural Networks (UCB) [23] adapts the learning rate according to the uncertainty defined in the probability distribution of the weights in the network, while Pomponi et al. [24] propose a regularization of network’s latent embeddings.

a) Rehearsal-based: Rehearsal-based solutions allocate a memory buffer of a predefined size and devise some smart schemes to store previously used data to be replayed in the future, i.e., to be added to future training samples. One of the first methodologies developed is Experience Replay (ER) [25], which stores a small subset of previous samples and uses them to augment the incoming task-data. Aljundi et al. [26] propose an evolution of ER which takes in consideration Maximal Interfered Retrieval (ER-MIR). Their proposal lies between rehearsal and regularization methods, its strategy is to retrieve the samples that are most interfered, i.e. whose prediction will be most negatively impacted by the foreseen parameters update. Among other mixed approaches we have Rebuffi et al. [27] that proposes a method which simultaneously learns strong classifiers and data representation (iCaRL). Gradient Episodic Memory (GEM) [28] and its improved version Averaged-GEM (AGEM) [29] exploits the memory buffer to constrain the parameter updates and stores the previous samples as trained points in the parameter space, while Gradient based Sample Selection (GSS) [26] diversifies/prioritizes the gradient of the examples stored in the replay memory. Finally, a recent method proposed by Shim et al. [30] scores memory data samples according to their ability to preserve latent decision boundaries (ASER).

b) Architectural-based: Architectural methods alter their parameter space for each task. The most influential architectural-based approach is arguably Progressive Networks...
(PN) [31], where a dedicated network is instantiated for each task while Continual Learning with Adaptive Weights (CLAW) [32] grows a network that adaptively identifies which parts to share between tasks in a data-driven approach. Note that, in general, the approaches that use incremental modules suffer the lack of task labels at test time, since there is no easy way to decide which module to adopt.

III. METHODOLOGY

Before introducing the dimensionality reduction approaches adopted in our quantity/quality analysis we have to introduce the CL scenario considered and its task composition. Unfortunately the community has not yet converged to a unique standard way to define a CL setting [33]. Here we adopt GDumb’s formulation which is the most general one and specifically resembles Lomonaco and Maltoni’s formulation [34]. In particular, we focus on the new class (NC)-type scenario [34] where each task $T_i$ introduces data instances of $C_T$, new, previously unseen, classes. More formally a dataset benchmark $D$, containing examples from $C_T$, is divided into $n$ tasks. Each task, $T_i$ with $i = 1 \ldots n$, carries a set of examples $T_i = \{X_{T_i}, Y_{T_i}\}$ whose class is previously unseen i.e. $Y_{T_j} \cap Y_{T_i} = \emptyset$ with $j \neq i$ and $Y_{T_i} = \{c_1 \ldots c_{T_i}\}$. In other words, the model experiences a shift in the distribution of data as we train on each new task. We also consider the more realistic class incremental scenario (CI), that is, we are not allowed to know task labels at test time.

As incremental approach we use the recently proposed GDumb, which is composed of a simple learner and a greedy balancer. That is, given a fixed amount of memory $M$, each instance of task data is randomly sampled in order to balance class instances in the memory, so that, at the end of the $T_i$ task experience, the memory contains an equal number of instances of all previously encountered classes i.e. each class has $\left[\frac{M}{C_T \times n}\right]$ instances in memory.

Besides providing state-of-the-art performances, GDumb has been proposed as standard baseline to question our progresses in continual learning research, since after experiencing a task, the simple learner (such as a ResNet18 [16] or a MLP) is trained only with memory data, making GDumb a fully rehearsal based approach with random filtering of incoming data, and thus the ideal candidate to carry our study. In the following paragraphs, we briefly describe all the strategies adopted for dimensionality reduction.

A. Random Projections (RP)

Extreme Learning Machines (ELM) [35] are a set of algorithms that exploit random projections as dimensionality reduction technique to preserve computational and spatial resources while learning. ELM have been introduced in 2006 and recently have found application in neuroscience [36], [37] and in other problems such as in molecular biology [38]. The idea can be roughly described as a composition of two modules where the first one performs a random projection of the data, while the second one is a learning model. The appealing property of RP lies in the Johnson-Lindenstrauss lemma [39] which states that given a set of points in a high dimensional plane, there is a linear map to a subspace that roughly preserves the distances between data points by some approximation factor.

The Johnson-Lindenstrauss lemma guarantees that we can obtain a low-distortion to the dimensionality reduction by multiplying each instance vector by a semi-orthogonal random matrix $Q_{m \times n}$ in the $(m, n)$ Stiefel manifold. More formally, let $x_i$ be an image of the current task of width, height and number of channels $w$, $h$, and $c$ respectively, then the size of $x_i$ is $n = hwc$. We can consider its vectorization as $v_i \in \mathbb{R}^n$ and its compressed representation

$$v_i' = Qv_i \quad \text{s.t.} \quad Q^TQ = I_m$$

with $v_i' \in \mathbb{R}^m$.

The usage of ELM unsuspectedly unlocks two main advantages: First it allows us to exploit the dimensionality reduction by increasing the number data instances storable in the memory buffer. Secondly and, more importantly, allows us to use models with significantly fewer parameters. On the other hand, the approach loses coordinate contiguity and, with that, shift co-variance, rendering convolutional approaches inapplicable.

After the random projection, data instances will be forwarded to the greedy sampler of GDumb to fill the memory $M$. Then, we perform a rehearsal train with any MLP-like architecture, resulting in an order-of-magnitude reduction in the amount of parameters needed to process visual data...
allowing the usage of CL rehearsal based solutions in very low resource scenarios.

B. Deep Encoders

Deep encoders are neural models \( \phi \) that take as input an image \( x_i \) and, depending from the structure of such model, can output either a latent vectorial representation \( v_i' \) or a squared feature map which we consider as a noise-free shrunked image \( x_i' \). Figure 2 (b) reports visually the two possible encoding scenarios. In this work, we adopt a Variational AutoEncoder (VAE) [17] for the first case and a pretrained ResNet18 [16] cut up to a predefined block (CutR) as a prototype for the second.

a) VAE: Variational Autoencoders [17] have been introduced as an efficient approximation of the posterior for arbitrary probabilistic models. A VAE is essentially an autoencoder that is trained with a reconstruction error between the input and decoded data, with a surplus loss that constitutes a variational objective term attempting to impose a normal latent space distribution. The variational loss is typically computed through a Kullback-Leibler divergence between the latent space distribution and the standard Gaussian, the total loss can be summarized as follows:

\[
L = L_r(x_i, \hat{x}_i) + L_{KL}(q(z_i|x_i), p(z_i))
\]

given an input data image \( x_i \), the conditional distribution \( q(z_i|x_i) \) of the encoder, the standard Gaussian distribution \( p(z_i) \), and the reconstructed data \( \hat{x}_i \). We use the encoding part of a VAE pretrained on a dataset by feeding each incoming image and retrieving the vectorial output representation \( v_i' \), then the data point is forwarded to GDumb’s greedy sampler to feed \( \mathcal{M} \).

b) CutR: As our second encoding approach, we use a pretrained ResNet18 [16] cut up to a predefined block. ResNets models are Convolutional Neural Networks (CNNs) introducing skip connections between convolutional blocks to alleviate the so called vanishing gradient [40] problem afflicting deep architectures. The idea behind it is to use the cut ResNet18 as a filtering module that outputs a smaller feature map, giving us \( x_i' \). In fact, we cut the network towards later blocks, since neurons in the last layers, encode more structured semantics with respect to the early ones [41]. Therefore, we are able to extract semantic knowledge from unseen images leveraging transfer learning [42]. This is to exploit the ability of a model to generalize over unseen data. We refer to this method with the name CutR(ResNet18). We use CutR instance encoding by feeding each image belonging to the current task and retrieving the shrunked output \( x_i' \) which is then forwarded to the greedy sampler module of GDumb to fill the memory \( \mathcal{M} \).

In our analysis, we adopted the less resource-hungry VAE scheme for datasets where shift co-variance is not as important, such as the MNIST, in which the digits are centered in the image and thus most approaches at the state-of-the-art use a MLP as classifier. In all other instances, we used the CutR scheme.

C. Resizing

We used also the simplest instance reduction approach one can think of i.e., resizing the images to very low resolution through standard bilinear interpolation. The resized images are then fed to the sampler of GDumb to balance the classes in \( \mathcal{M} \) and all training and prediction is performed on the lowered resolution images.

Independently of the approach adopted, all data instances are reduced before storing them in memory \( \mathcal{M} \), then we use GDumb’s greedy sampler to select and balance class instances, and finally, we use a suitable learner to fit memory data and assess the performance. In general, following GDumb, we adopt ResNet18 for large-scale image classification tasks for all approaches that maintain shift co-variance, reverting to a simple MLP for approaches without shift co-variance like RP.

IV. EXPERIMENTS

We performed our analysis on the following standard benchmarks:

- MNIST: the dataset is composed by 70000 28 × 28 grayscale images of handwritten digits divided into 60000 training and 10000 test images belonging to 10 classes.
- CIFAR10 [44]: consists of 60000 RGB images of objects and animals. The size of each image is 32 × 32 divided in 10 classes, with 6000 images per class. The dataset is split into 50000 training images and 10000 test images.
- CIFAR100 [45]: is composed by 60000, 32 × 32 RGB images subdivided in 100 classes with 600 images each. The dataset is split into 60000 training images and 10000 test images.
- ImageNet100 [45]: the dataset is composed of 64 × 64 RGB images divided in 100 classes; it is composed of 60000 images split into 50000 training and 10000 test images.
- Core50 [34]: the dataset is composed of 128 × 128 RGB images of domestic objects divided in 50 classes. The set consists of 164866 images split into 115366 training and 49500 test.

Following [18], we use final accuracy as the evaluation metric throughout the paper. The metric is computed at the end of all tasks against a test set of never seen before images composed of an equal number of instances per class. This allows us to directly compare against the largest number of competitors in the literature.

All the experiments have been conducted with an Intel i7-4790K CPU with 32GB RAM and a 4GB GeForce GTX 980 machine running PyTorch 1.8.1+cu102.

A. Parameter Sensitivity

In the first experiment, we compared different dimensionality reduction strategies as we altered the parameters. The analysis was conducted on three different datasets: MNIST, CIFAR10 and ImageNet100. In this evaluation we fixed the amount of memory buffer used for GDumb during rehearsal training, and we measured the final accuracy as the parameters varied for each dimensionality reduction method. In particular we subdivided both MNIST and CIFAR10 datasets into 5 tasks.
of 2 classes each, with 600 KiB dedicated memory buffer, while ImageNet100 was divided into 10 tasks of 10 classes each, with 12 MiB memory buffer.

Figure 3 plots the performance of the various schemes as we reduce the dimensionality of the instances and thus increase their number in the allocated memory. The orange line represents the performance of the resize scheme. For the MNIST dataset, we considered eight different target sizes\(^1\) \(x_i' \in \{27 \times 27, 24 \times 24, 20 \times 20, 16 \times 16, 12 \times 12, 8 \times 8, 4 \times 4, 2 \times 2, 1 \times 1\}\). We performed the same resizing for CIFAR10 data. We did not report CIFAR100 analysis since the data format is the same as CIFAR10 and the result would be analogous. For ImageNet100, we resized each instance to \(x_i' \in \{32 \times 32, 24 \times 24, 16 \times 16, 6 \times 6, 4 \times 4, 2 \times 2\}\).

The green line of Figure 3 represents the deep encoders. In particular, for MNIST we used a VAE [17] pretrained on KMNIST [46] and analyzed the performance of GDumb with compressed instances as we altered the size of the latent embedding vector to \(v_i' \in \{128, 64, 32, 16\}\). On the other hand, for the CIFAR10 and ImageNet100 dataset we considered different parameters for CutR. In particular, we cut the ResNet18 up to the sixth layer to get a \(4 \times 4\) output, to the fifth to have a \(8 \times 8\) encoding, and lastly up to the third block to get a \(16 \times 16\) feature map.

The CutR ResNet18 has been pretrained on the complete ImageNet, thus the results in the ImageNet100 benchmark can be biased. We denote these biased results with CutR*.

Lastly, the blue line of Figure 3 reports the accuracy of Random Projection followed by an MLP classifier. We recall that this kind of architecture is a variation of an Extreme Learning Machine (ELM), therefore we will refer to it with the term ELM. We analyzed the final accuracy as the size of the random projection changes, in particular the embedding sizes considered are \(v_i' \in \{512, 256, 128, 64, 32, 16\}\) for all the datasets.

For all the experiments in MNIST data, we used a 2-layer MLP with 400 hidden nodes as learning module, while we used a Resnet18 [16] for all the other analysis with exception of ELM scheme that maintains the 2-layer MLP model throughout. We did not perform any hyperparameter tuning on the learning module in accordance with the GDumb [18] experimental protocol. For completeness we report the learning parameters: the system uses an SGD optimizer, a fixed learning rate \([0.05, 0.0005]\), an SGD with a SGDR schedule with \(T_0 = 1\) and warm start of 1 epoch. Early stopping with patience of 1 cycle of SGDR, along with standard data augmentation is used (normalization of data). GDumb uses cutmix [48] with \(p = 0.5\) and \(\alpha = 1.0\) for regularization on all datasets except MNIST.

As we can also see from Figure 3 all the strategies considered unlock performance greatly above GDumb, thus suggesting that the quantity/quality trade-off is severely skewed toward quality since each dimensionality reduction technique greatly improves the amount of data instances that can be stored in the memory buffer. It is also evident that the simple resizing strategy gives the best performance improving GDumb by \(+6\%\) on MNIST and roughly by \(+20\%\) on both CIFAR10 and ImageNet100 datasets.

Moreover, we chose to consider extreme levels of encoding. We did so to find the level of compression that irreversibly

\(^1\)throughout the paper we omit to write the channel component for brevity

### Table I

**ImageNet100 and MNIST Final Accuracy (5 runs) analysis as we vary the memory. Each method has a fill color which is consistent among plots and tables.**

| Method      | ImageNet100 | CIFAR10 |
|-------------|-------------|---------|
|             | Acc @ 12MiB | Acc @ 6MiB |
| AGEM [29]   | 7.0 ± 0.4    | 7.1 ± 0.5    |
| ER [25]     | 8.7 ± 0.4    | 11.8 ± 0.9   |
| EWC [20]    | 3.2 ± 0.3    | 3.1 ± 0.3    |
| GSS [43]    | 7.5 ± 0.5    | 10.7 ± 0.8   |
| ER-MIR [26] | 8.1 ± 0.3    | 11.2 ± 0.7   |
| ASER [30]   | 11.7 ± 0.7   | 14.4 ± 0.4   |
| ASER* [30]  | 12.2 ± 0.8   | 14.8 ± 1.1   |
| GDumb [18]  | 13.0 ± 0.3   | 21.6 ± 0.3   |

**Resize (8 × 8)**

| Method      | ImageNet100 | CIFAR10 |
|-------------|-------------|---------|
|             | Acc @ 12MiB | Acc @ 6MiB |
| ELM (128)   | 13.3 ± 0.2  | 15.4 ± 0.4 |
| CutR (8 × 8)| 36.25 ± 0.4*| 36.27 ± 0.5*|

**Table II

**CIFAR10 Final Accuracy (5 runs) analysis as we vary the memory for all schemes considered.**

| Method     | Acc @ 600KiB | Acc @ 1.5MiB | Acc @ 3MiB |
|------------|--------------|--------------|------------|
| EWC [20]   | 17.9 ± 0.3   | 17.9 ± 0.3   | 17.9 ± 0.3 |
| GEM [28]   | 16.8 ± 1.1   | 17.1 ± 1.0   | 17.5 ± 1.6 |
| AGEM [29]  | 22.7 ± 1.8   | 22.7 ± 1.9   | 22.6 ± 0.7 |
| iCARL [27] | 28.6 ± 1.2   | 33.7 ± 1.6   | 32.4 ± 2.1 |
| ER [25]    | 27.5 ± 1.2   | 33.1 ± 1.7   | 41.3 ± 1.9 |
| ER-MIR [26]| 29.8 ± 1.1   | 40.0 ± 1.1   | 47.6 ± 1.1 |
| ER5 [26]   | -            | -            | -          |
| ER-MIR5 [26]| -          | -            | -          |
| GSS [43]   | 26.9 ± 1.2   | 30.7 ± 1.2   | 40.1 ± 1.4 |
| ASER [30]  | 27.8 ± 1.0   | 36.2 ± 1.1   | 43.1 ± 1.2 |
| ASER* [30] | 26.4 ± 1.5   | 36.3 ± 1.2   | 43.5 ± 1.4 |
| GDumb [18] | 35.0 ± 0.6   | 45.8 ± 0.9   | 61.3 ± 1.7 |

**Resize (8 × 8)**

| Method     | Acc @ 1.5MiB | Acc @ 3MiB |
|------------|--------------|------------|
| ELM (128)  | 43.0 ± 0.3   | 47.1 ± 0.2  |
| CutR (8 × 8)| 54.4 ± 0.2   | 60.9 ± 0.2  |

**Table III

**MNIST Final Accuracy (5 runs) analysis as we vary the memory for all schemes considered.**

| Method     | Acc @ 382KiB |
|------------|--------------|
| GEN [49]   | 75.5 ± 1.3   |
| GEN-MIR [26]| 81.6 ± 0.9   |
| ER [25]    | 82.1 ± 1.5   |
| GEM [28]   | 86.3 ± 1.4   |
| ER-MIR [26]| 87.6 ± 0.7   |
| GDumb [18] | 91.9 ± 0.5   |
| Resize (8 × 8)| 97.2 ± 0.1 |
| ELM (128)  | 95.0 ± 0.2   |
| VAE (32)   | 94.6 ± 0.1   |
corrupts spatial information and thus makes learning impossible. Surprisingly, it turns out that a $2 \times 2$ resizing still works on CIFAR10 data with performances above GDumb while a $1 \times 1$ resize is still better than a random classifier whose performance would be 20% of final accuracy. This is a strong evidence that the amount of data storable in the memory buffer plays a central role, but also that CIFAR10 dataset constitutes an unrealistic benchmark and should not been considered to assess novel methodologies in the future.

After choosing and fixing the optimal parameters for each compression scheme, we study the performance of the rehearsal system as we alter the quantity of the memory allocated. In Tables II, I we compute the final accuracy for all the datasets previously considered, with the addition of CIFAR100 with an increase of 20% in performance. The amount of dedicated memory for the rehearsal buffer, has been chosen in order to be consistent with several other methods at GDumb, allowing us to compare GDumb’s performance on optimized memory schemes against other methods. As we can see, all memory optimizations still provide huge advantages as the memory buffer varies, suggesting again, that instance quantity plays a fundamental role in rehearsal systems even with extreme encoding settings.

Finally, we note that the deep models used for classification have a large number of degrees of freedom and require a large amount of instances to be properly trained to capture the complexity of the task at hand. Simpler, lower dimensionality instances allow both for more instances and simpler classifiers with fewer parameters without losing lot of informational content.

B. Resource Consumption

With the second experiment, we wanted to analyze the performance versus the total memory requirement for each approach. Here, we increased the number of instances in the memory buffer and added to the total consumption the working memory used by the classifier to store (and train) the parameters. We considered three different scenarios: first we used the plain GDumb CL system without dimensionality reduction (representing GDumb), then we used ELM (with fixed embedding size of $v'_i = 128$), and lastly the resizing scheme (images resized to $x'_i = 8 \times 8$). We selected the best parameters resulting from the previous experiment.

We then assessed the performance and resource usage using a new dataset, namely the Core50 [34]. The reason behind the use of Core50 to validate our findings is twofold: first, we test again whether the quantity of extremely encoded data plays a central role on our rehearsal scheme. Secondly, we measure the performance and the resource usage of a CL system on a more complex set of tasks. We divided the dataset into 10 tasks of 5 classes each.

In Figure 4, we report the results of this experiment. We can see that extreme levels of resizing still provide optimal results in all the datasets considered. One striking finding is
that in Core50 with extreme resizing, even if the size was not optimized for the dataset, the final accuracy is increased by +67% with respect to GDumb. Second, we note that ELM constitute a viable solution in low resources scenarios. Indeed, we can surpass the performance of GDumb for low memory scenarios where even just the classifier used in other approaches could not fit in the allocated memory, much less the rehearsal buffer. This is clearly observed from the Core50 results. We can appreciate that by randomly projecting image data and learning in a low resource scenario provides a boost of +34% in the final accuracy.

Finally, it is worth noting there is a striking dissonance in the literature of rehearsal-based method when the narrative around buffer-memory sizes revolves around decisions among sizes of the order of 300KiB to 600KiB when then the same systems adopt complex classifiers using several megabytes of memory just for the learned parameters and in the order of gigabytes of working memory for learning. In a real constrained-memory scenario, a simpler classifier with more instances offers a clear advantage.

In fact, in a real low-resources scenario deep convolutional systems using several megabytes of memory for the model parameters and gigabytes of working memory for learning are not a viable solution. In this case, a variation of Extreme Learning Machines offer a simple and effective solution.

V. CONCLUSIONS

In this study, we analyzed the quantity/quality trade-off in rehearsal-based Continual Learning systems adopting several dimensionality reduction schemes to increase the number of instances in memory at the cost of possible loss in information. In particular, we used deep encoders, random projections, and a simple resizing scheme. What we found is that even simple, but extremely compressed encodings of instance data provide a notable boost in performance with respect to the state of the art, suggesting that in order to cope with catastrophic forgetting, the optimization of the memory buffer can play a central role. Notably, the performance boost of extreme instance compression suggests that the quality/quantity trade-off is severely biased toward data quantity over data quality.

We suspect that some fault might be in the overly simplistic datasets adopted by the community, but mostly the deep models used for classification are well known to be data-hungry and the instances stored are not sufficient to properly train them, but can suffice for simpler classifiers with fewer parameters working on simplified instances.

It is worth noting there is a striking dissonance in the literature of rehearsal-based method. The narrative on buffer-memory sizes revolves around decisions among sizes of the order of 300KiB to 600KiB when then the same systems adopt complex classifiers using several megabytes of memory just for the learned parameters and in the order of gigabytes of working memory for training. In a real constrained-memory scenario, a simpler classifier with more instances offers a clear advantage.

REFERENCES

[1] S. Gidaris and N. Komodakis, “Dynamic few-shot visual learning without forgetting,” in Conference on Computer Vision and Pattern Recognition, (CVPR). IEEE, 2018.
[2] S. Thrun, “Is learning the n-th thing any easier than learning the first?” in Advances in Neural Information Processing Systems, (NIPS), D. S. Touretzky, M. Mozer, and M. E. Hasselmo, Eds. MIT Press, 1995.
[3] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, “Continual lifelong learning with neural networks: A review,” Neural Networks, 2019.
[4] F. Cermelli, M. Mancini, S. R. Bulo, E. Ricci, and B. Caputo, “Modeling the background for incremental learning in semantic segmentation,” in Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
[5] U. Michieli and P. Zanuttigh, “Incremental learning techniques for semantic segmentation,” in Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCV-W), 2019.
