Comparing Incremental Learning strategies for Image Classification

Alessandro Desole, Gabriele Tiboni, Roberto Franceschi

Politecnico di Torino

{alessandro.desole, gabriele.tiboni, roberto.franceschi} @studenti.polito.it

Abstract

Recent studies in machine learning aim at developing models that are able to incrementally learn new concepts over time with minimal effort in terms of resource usage. In this work, we recall and reproduce from scratch two of the most popular approaches to this problem ("Learning without forgetting" and "iCaRL") and we conduct an in-depth ablation study on the previous methods. We finally propose a variation that exploits an implicit hysteresis effect of the network when storing a dynamic number of samples from old classes throughout the incremental learning process, which allows to consistently increase the average performances without varying the overall resource overhead.

1. Introduction

The intelligence of living organisms lies in their ability to learn by adapting to new conditions. In human being context, all of us have to be able to take on new information continuously, adding them to the previous ones, with limited forgetfulness. For instance, people who want to start speaking a foreign language will learn about new words and grammar without forgetting notions coming from the native one.

This challenge is also present in the context of Deep Learning. Here, many computer vision applications require incremental learning capabilities to increase the number of items stored continually, without forgetting previously learned notions and with minimal effort in terms of memory and time. For example, a face recognition system should be able to handle new faces to identify new individuals reminding ones already learned. But while this is trivial to accomplish for most people, it is not the case for a machine learning system based on traditional models, because these would require all the samples always available to work fine. Indeed conventionally, deep neural networks are trained offline, relying on a large dataset prepared in advance. But, as the number of data grows incrementally, storing and retraining on such data becomes infeasible. Indeed, re-training a model from scratch whenever a new class is encountered can be prohibitively expensive not only due to training data storage requirements but also to the computational cost of full retrain.

So, recently has been introduced a new paradigm, that is receiving increasing attention from the scientific community, and is considered as a promising solution to the practical challenges mentioned above. It is just named as *Incremental Learning* and allows a model to be continually updated on new data, instead of being trained once on a whole dataset. This approach sees increasing demand in recent years when many modern applications are exposed to continuous streams of data during everyday operation.

It is worth defining the notion of incrementality since the literature review shows that this term is often used in many different contexts, such as the following dealt scenario called *class-incremental learning* in which a visual object classification system should be able to incrementally learn about new classes, every time training data becomes available for them. A truly incremental classification learning algorithm must be characterized by the following properties:

i. ability to being trained from a flow of data, with examples of different classes appearing in any order, and at any time,

ii. good performance on classifying new classes, also maintaining the classification performance on old ones, although original training data for them are no longer accessible,

iii. a reasonable number of parameters and bounded or very slowly growing computational and memory requirements with respect to the number of categories seen so far.

Therefore, an ideal approach would be able to train on an infinitely large number of classes incrementally, without losing accuracy, and having approximately the same number of parameters, as if it were trained from scratch. However, it has been observed that *class-incremental learning* is subject to a fundamental difficulty. Indeed, although some attempts have been made to address this topic, most of the models still suffer from a dramatic decrease in performance on the old classes when new information is added, in particular in that models that exploit stochastic gradient descent (SGD) for optimizing the loss function [4]. This deterioration of classification accuracy is an effect known in the literature as *catastrophic forgetting* or *catastrophic interference* [12].

Many techniques are accordingly found to manage this issue and in this paper we review some of these algorithms, evaluating them within the same experimental settings based on iCaRL implementation [13] in order to provide an as objective comparative study as possible. Moreover, we will apply some changes on starting configuration by trying different loss criteria and classifiers. Finally, we will present a new variation which is the result of our study about one of the problems related to class-incremental learning. Our approach is based on the intuition that if we could make use of more data to learn achieve higher accuracies in the first batches, then we could have better performances also in the following steps as we will have learned more robust features from the beginning. Therefore, to do this we will analyze how a dynamic number of exemplars stored can lead to a 1.5% increase in the average incremental accuracy.

2. Related works

As previously said, incremental learning is a longstanding problem in machine learning and the main issue to be managed consists of alleviating negative effects caused by *catastrophic forgetting*. In the late 1980s, *McCloskey et al.* [12] first identified the consequence of this issue in the connectionist models, where the memory about the old data is overwritten when retraining a neural network with new data.

Recently, thanks to the exciting progress in deep learning, there has been a lot of research toward this field, to actively develop methods to improve CNN performances. Some of these approaches require storing some data from old classes. Hence, the methods considered are presented below.

Finetuning [3]. This method learns a multi-class network for new incoming categories without taking any measures to prevent *catastrophic forgetting*. Indeed, fine-tuning degrades the performance of the network on previously learned tasks because parameters are optimized every time that a new batch of classes is considered. So, the shared parameters change without new guidance for the original task-specific prediction parameters and, as consequence, only the last categories analyzed are possible to be recognized, while the others are progressively overwritten.

Joint Training [2]. In Joint Training all parameters of the convolutional neural network are jointly optimized, i.e., the model is trained both on new coming data and on all the classes learned so far. This method's performance may be seen as an upper bound of what an approach can achieve. Joint training becomes increasingly cumbersome in training as more tasks are learned and, since old samples are not always available, this approach is not feasible in real-world applications.

Learning without Forgetting (LwF) [10]. Using only samples for the new task, this method retains knowledge of preceding tasks by means of knowledge distillation, i.e. without the need of old task's images and labels. Before training the new batch of classes, the network outputs for the new task are recorded, and are subsequently used during training to distill prior task knowledge. Thus, the success of this method depends heavily on the new task data and how strong it is related to prior tasks because the model is fine-tuned with knowledge distillation loss [7], to encourage the output probabilities of old classes for each image to be close to the original network outputs.

Incremental Classifier and Representation Learning (iCaRL) [13]. iCaRL is a practical strategy presented by Rebuffi et al. where the tasks of learning the classifier and the feature representation in the class-incremental setting are decoupled. iCaRL uses the *nearest-mean-of-exemplars* (NME) approach to classify test samples, i.e., assuming fixed allocated memory, it maintains an auxiliary set containing old and new data samples. The selection and storing of these exemplars are based on a technique called *herding*. The data representation model is so updated when new samples are available, using a combination of knowledge distillation and classification loss.

3. Background

Before moving on to the ablation study and our proposal to address *class-incremental learning*, we describe the mechanisms and implementation details of the most important state-of-the-art approaches just cited in point 2, and we report the results we obtained when reproducing them from scratch. In this section, we will also describe in detail the dataset and the network adopted in this work.

Dataset. For this task, the images and labels are taken from the CIFAR-100 dataset [9] which was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The dataset contains 60000 colour images downscaled to 32×32 pixels, split evenly between 100 distinct object classes grouped into 20 superclasses (5 classes for each superclasses). In particular, the dataset is originally divided

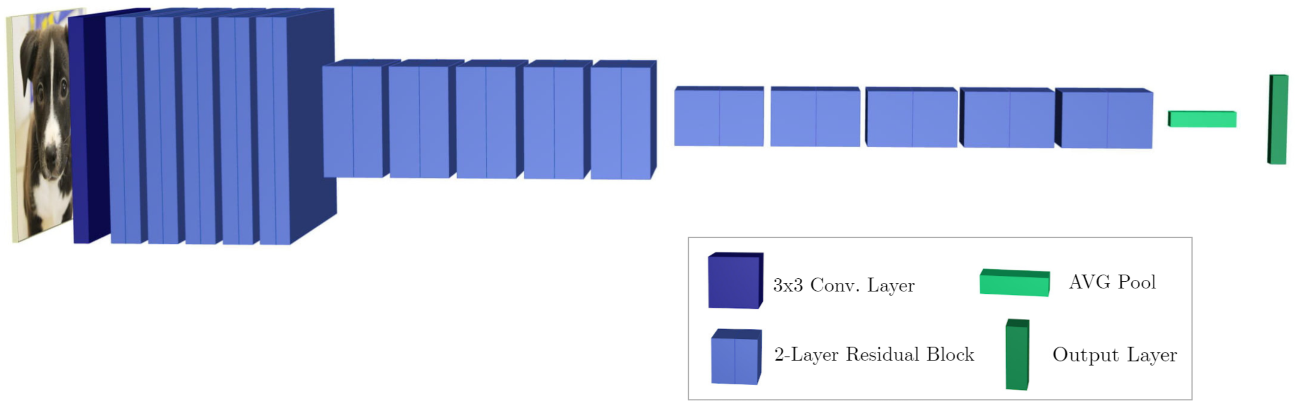


Figure 1. ResNet-32 architecture. Each block shows the size of the feature maps created during the forward pass. The average pooling at the end of the convolutional part extracts 64 features which are later forwarded to the 100-neuron output layer.

into 50k as training (500 per class) and 10k as testing (100 per class). Each image has 3 colour channels (i.e. RGB) and pixel dimensions 32×32 for an overall size per input of $3 \times 32 \times 32 = 3072$.

Neural Network. The whole work has been mainly conducted on top of the iCaRL setting, being one of the most popular attempts for addressing incremental learning. From here, not only the same training dataset has been used but also the same ResNet-32, optimized for the CIFAR dataset (see section 4.2 of [5]). Given our code implementation in PyTorch some discrepancies may still arise from the different framework used, but an almost identical version of the iCaRL ResNet, implemented in PyTorch, has been found on github[1]. After a careful examination of the network, we noticed a couple of small structural differences, and we modified the net according to iCaRL source code to exactly match their setting. In particular, the following details have been adjusted from the PyTorch ResNet32 found:

i. removed ReLu [1] nonlinearity from the very last convolutional layer, so that features extracted may also be negative;

ii. added *Kaiming initialization* [6] to the output layer parameters, instead of the default random initialization.

However, in point 3.2, we will further discuss how the previous modifications have little to no impact on the final results w.r.t. to the original network. Note that the ResNet-32 used has not being pretrained on ImageNet before the analysis. Such approach would break the whole point of incremental learning (the network should have no prior knowledge of future classes). Finally, a full representation of the ResNet-32 and its fundamental blocks is shown in fig. 1.

[1]https://github.com/hshustc/CVPR19_Incremental_Learning/blob/master/cifar100-class-incremental/resnet_cifar.py

Learning without forgetting. A popular method to avoid catastrophic interference is LwF [10] that applies the concept of knowledge distillation, i.e., the transferring of knowledge from a large, highly regularized model into a smaller one [7]. This regularization approach alleviates forgetting by imposing constraints on the update of the weights by adding a regularization term that encourages new classes to perform similarly on the old tasks. According to the LwF algorithm, we keep a copy of the model M_{t-1} before learning the new task and distills its predicted probabilities into the new model M_t (which may otherwise suffer interference from the current task t). Our implementation of LwF method uses a modified *binary cross-entropy loss* described by the following equation.

$$BCE(p_y(x_i), y_i) = -y_i log(p_y(x_i)) - (1-y_i) log(1-p_y(x_i)) \tag{1}$$

In particular, for old classes, we take as output the ones computed at time $t-1$, while for new classes (i.e. the current batch of 10 classes) we take the current network's outcomes. Note that we use the implementation as described in [13], which has a distinct criterion and a different neural net with respect to the original one of [10]. However, it is observed in our experiments (see results in Section 3.2) and [13], [8] that the LwF approach shows a bias towards recent classes.

iCaRL. The strategy adopted by iCaRL [13] to deal with the above problem is storing a subset of real data of old classes (called *exemplars*) that are used along with new data to dynamically adapt the weights of the feature extractor. This approach learns a classifier and a representation of data during training. Afterward, at the end of each group of classes, a classification strategy named *nearest-mean-of-exemplars* is applied. Specifically, it computes a prototype vector μ_y by averaging features of all samples for each class. During inference, it extracts the features for a test sample and assigns the class label of the most similar prototype as follows:

$$y^* = \underset{y=1,\dots,t}{argmin} \|\varphi(x) - \mu_y\| \tag{2}$$

Since we work with L2-normalized vectors the previous equation 2 can be rewritten as:

$$y^* = \underset{y=1,\dots,t}{argmax}\ \mu_y^T \varphi(x) \tag{3}$$

Regarding the last formula, further theoretical details are provided by appendix A.1. By doing so, the predicted label can now be seen as a classical classification task where the weights are substituted with the class means, as shown in the previous equation. The representation of the image is computed as the L2-norm output of the penultimate layer of a neural network and the class with the closest representation is assigned. By combining new and old data, this approach is able to prevent catastrophic forgetting but at the expense of a higher memory usage (given by a fixed threshold K). So, for a given memory budget, the number of exemplars stored should decrease when the number of classes increases, which inevitably leads to a decrease in performance.

3.1. Implementation details

All models are implemented with PyTorch and trained on Google Colab GPUs. As already mentioned, we adopt a 32-layer ResNet for CIFAR-100. For the training dataset, we perform data augmentation on the input images, applying a random crop with padding, and then a random horizontal flip (with 50% probability) is performed. For test images, instead, only a normalization to the interval [-1, 1] is performed. Note that for the sake of our work we do not use a validation set since we are reproducing the results of other papers, and so we don't have to perform hyperparameters tuning.

For each method experimented all the hyperparameters are the same as in [13]. So, each training step consists of 70 epochs. The learning rate starts at 2.0 and it is decreased with a step-down policy by a factor of 0.2 after 49 and 63 epochs. For the optimizer we use stochastic gradient descent with a weight decay of 0.0001 and a momentum of 0.9. Regarding iCaRL we store up to K = 2000 exemplars. For all the methods reported in the results (see Section 3.2) we used Binary Cross-Entropy loss. Besides, the learning rate specified above may seem high, but this is due to the average BCE which is calculated not only on the batch size but also on the number of classes.

For the experiments, the classes are arranged in a fixed random order (we will average over 3 different random seeds when comparing the results). Each method is then applied with a class-incremental approach. After each incremental phase, the output model is evaluated on all the classes observed so far. Thus, the evaluation result for each method is a curve of the classification accuracies after each phase. Furthermore, if a more general indicator is preferred, we report the *average incremental accuracy* as in [13].

Our source code and further data are available at https://github.com/gabrieletiboni/Incremental-learning-on-CIFAR100.

3.2. Our results on LwF and iCaRL

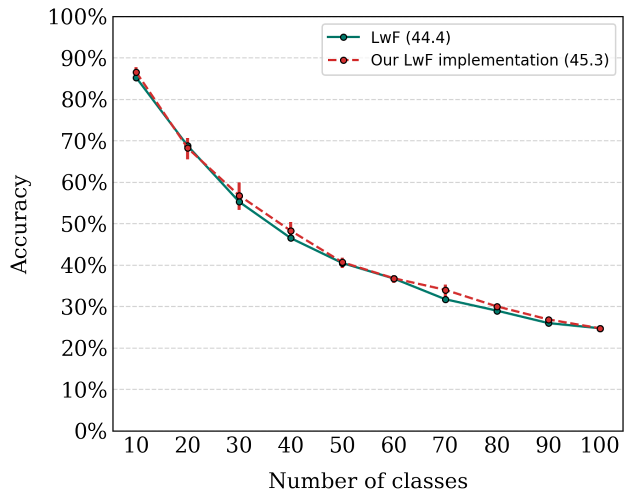
To start off the project all efforts have been put into reproducing the same results of Learning without forgetting (LwF) and iCaRL approaches. As previously stated, our work is mainly based on the iCaRL settings (losses, hyperparameters, dataset, network) and LwF results later described should not be compared to the original paper, which makes use of different classification and distillation losses, a different dataset, etc.

In addition to the two main approaches, *Finetuning* and *Joint Training* have also been tested and shown below. The former one does not take any particular measures to prevent catastrophic forgetting, and it represents the naive attempt in addressing class-incremental learning by finetuning the entire network on the new task only, without storing old-tasks samples. Joint Training, instead, refers to the results obtained if all classes were stored at all times (even samples from previously seen classes), which is to say if no incremental learning was present. This approach can be seen as an upper bound for our analysis.

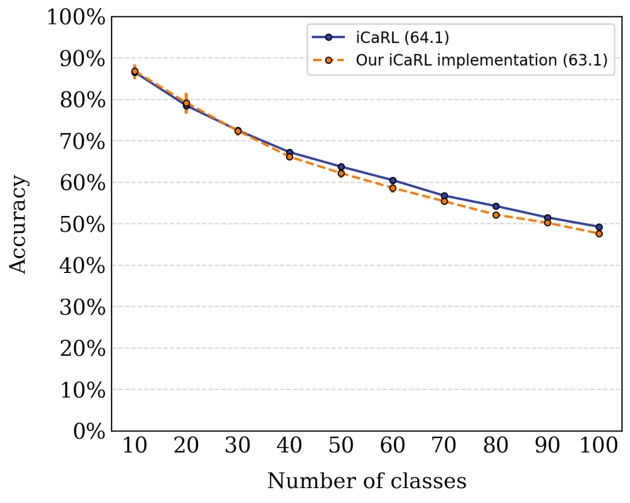
Note that all results have been obtained with 10 incremental steps of 10 classes each, and have been averaged over three different shuffles of classes in the CIFAR-100 dataset to get a better estimation. As previously mentioned, we also report the average incremental accuracy (average accuracy obtained over the 10 batches of classes) as a way to describe the goodness of tested models in one single number.

Figure 2 shows our reproductions from scratch of LwF and iCaRL and how they performed compared to the original approaches (shown in Section 4.1 of [13]). We consistently reached the baseline of the former paper but struggled to obtain the exact same results reported by iCaRL on the final incremental steps. However, fairly large differences in accuracy have been observed when running the algorithm with a different order of classes. In addition, the different framework used by iCaRL (i.e. theano-lasagne[2]) may have

[2]https://lasagne.readthedocs.io/en/latest/



(a) LwF



(b) iCaRL

Figure 2. Accuracies on 10-class incremental steps, with our implementations of LwF (a) and iCaRL (b) averaged over 3 different shuffles of the CIFAR-100 classes. The average incremental accuracy is reported in parentheses.

Classes seen	10	30	50	100
Accuracy Standard Deviation	2.34%	2.81%	1.95%	0.78%

Table 1. Standard deviation on accuracies at the first, third, fifth and last incremental step, computed on 10 different shuffles of classes.

also played a role.

To further inspect the discrepancies between runs, Table 1 shows the standard deviations observed during the incremental steps with 10 different random seeds for shuffling the order of classes seen.

Overall, iCaRL reports an average incremental accuracy of 64.1, whereas our implementation only falls down of 1%, with an average of 63.1 on our selected seeds. Note that iCaRL does have a preferred random seed when showing the results.

As mentioned in point 3, we performed two modifications on the original PyTorch ResNet-32 found, and we further inspected how these affected the performances when compared to iCaRL on the same seeds. Fig. 3 shows how little the new network impacts on the accuracy of the model. When taking into account 3 different runs per seed, modifying the network to exactly match the iCaRL ResNet does not change the average incremental accuracy (with a 95% confidence), assuming random distribution.

Later, we discuss the efficiency of iCaRL's prioritized exemplar selection based on herding [13] with respect to a random sampling. Fig. 3 shows the results obtained with and without prioritized exemplar selection, averaged over 3 seeds. Random sampling images from previous classes slightly underperformed, on average, with respect to the herding selection strategy. Note that this effect has particu-

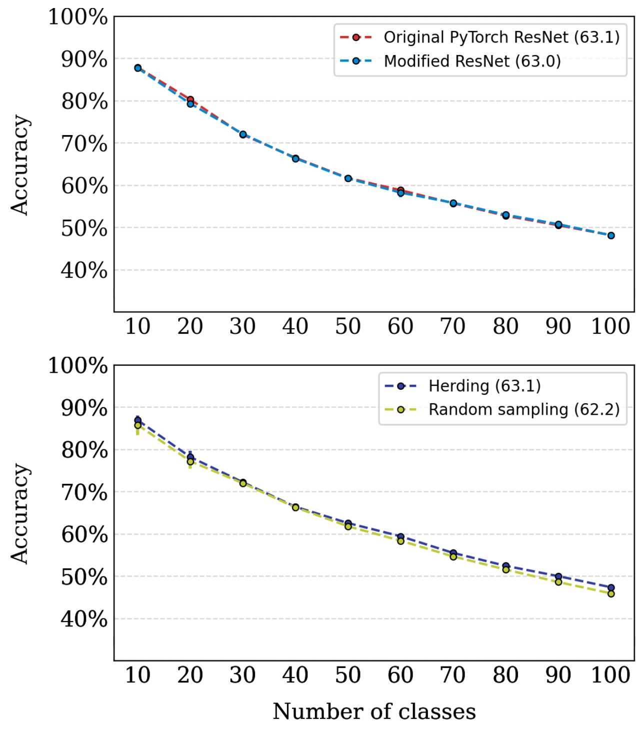


Figure 3. Comparison of the accuracies using the original ResNet-32 and our modified version (top), and using herding or random sampling to update the exemplars set (bottom).

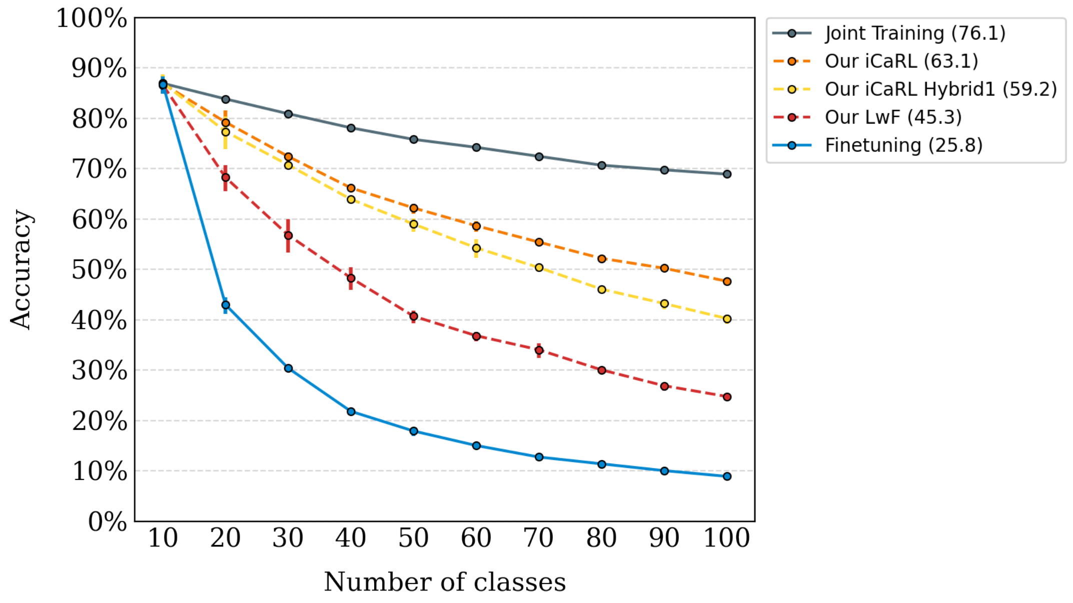


Figure 4. Top-1 accuracy on CIFAR-100 on all implemented methods

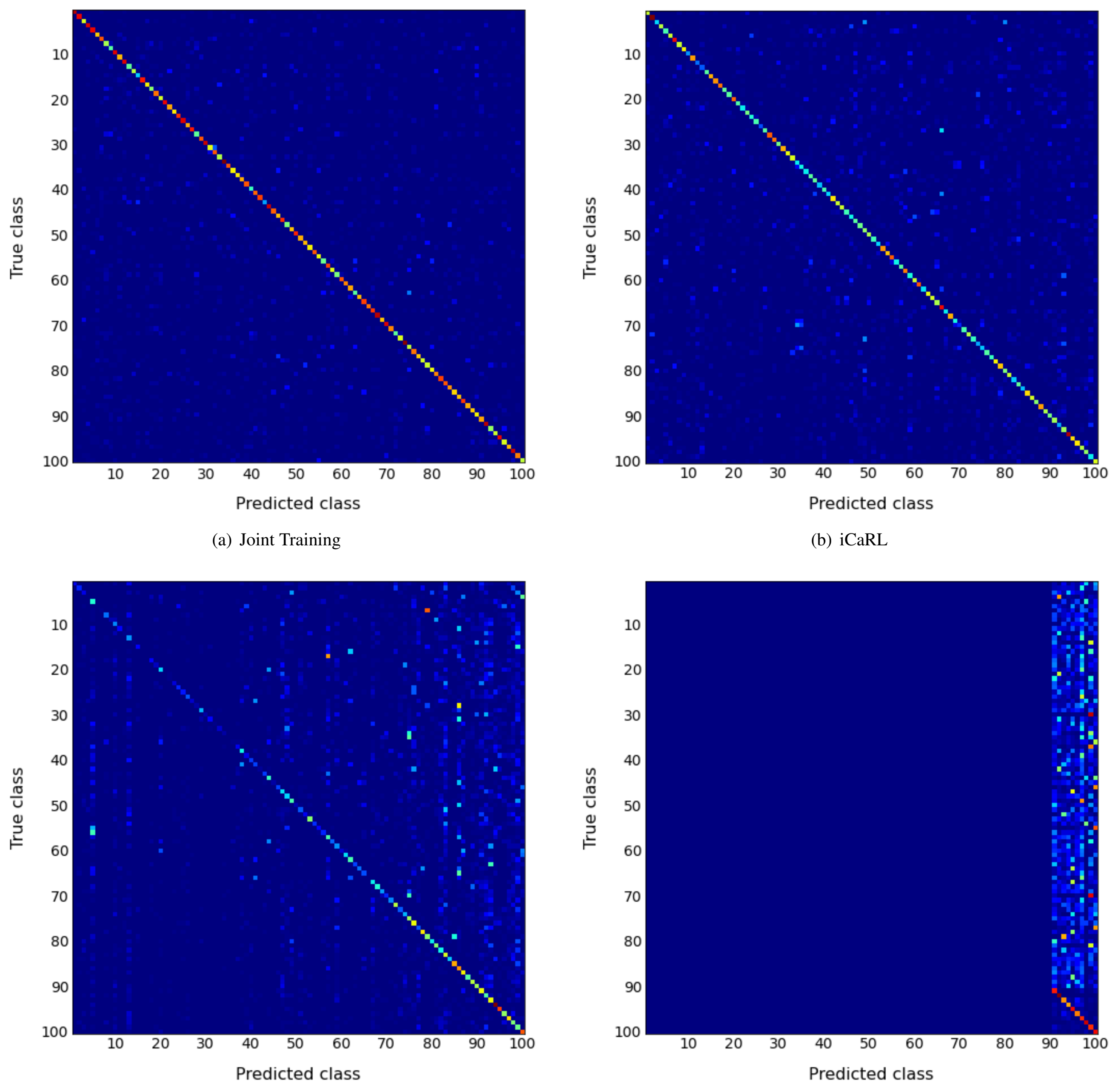
larly been observed on the second half of the training steps, demonstrating that exemplar selection plays an increasingly greater role in affecting the results as more classes are discovered. More precisely, an average increase of 1.43% in accuracy is recorded in the very last incremental step. On the other hand, selection based on herding increased the global computation time.

Finally, all methods have been compared side-by-side in fig. 4 to make global observations across different strategies in a class-incremental learning scenario. iCaRL *Hybrid1* is also reported as in [13], representing predictions made with network outputs instead of *nearest-mean-of-exemplars* classification. Interestingly enough, Joint Training serves as reference upper bound for the analysis, while finetuning the network on new tasks leads to a clear catastrophic forgetting as in [12], [4]. Note that all methods start from the same initial value, as, when learning the first 10 classes, all approaches are exactly equal to each other (except for iCaRL's NME classification). Like in [13], the confusion matrix for each method has also been computed and is shown in fig. 5. The given matrices give very interesting insights on the advantages and disadvantages of each approach, highlighting what they are actually doing in the inside and how classification ends up looking like. In particular, finetuning the network is so drastically changing the feature representation and old output weights that previous classes don't get recalled a single time.

Finally, input normalization has been further inspected. Differently from a pretrained approach, no input normalization has been forced by the network used. In these cases, one could try to infer the means and standard deviations (or stds) from the training dataset for standardization of input features, in order to speed up convergence and obtain better results. Unlike standard learning scenarios though, only a small set of classes are seen at each incremental step and a global mean cannot be computed unless all samples are stored. Then, to prevent cheating from inferring future knowledge into the model, no ad-hoc input normalization has been performed (pixel values are only mapped to the interval $[-1, 1]$). However, differences in performances concerning a CIFAR-100 normalization with the corresponding dataset means and stds have been inspected but did not show noticeable advantages. Note that also in this case the test dataset has not been taken into account for computing the metrics, since it must remain unseen by the model by definition. A different approach computing a running mean and standard deviation on the batches of classes seen so far has also been tried, but eventually led to no significant improvements.
From this point on, all following graphs referring to iCaRL will do so according to the values of our implementation, in order to compare the ablation studies and our proposal on the same shuffles of classes.

4. Ablation Study

To further analyze class-incremental learning on top of previous attempts, we conducted an ablation study on the iCaRL setting. In particular, we worked on changing the classification and distillation losses, and implementing different classifiers on top of the feature extractor. Note that other original features remained the same as iCaRL, such as the herding based exemplars selection, the number of exemplars selected, the number of classes seen at each batch, etc.



(a) Joint Training

(b) iCaRL

(c) LwF

(d) Finetuning

Figure 5. Confusion matrices computed on the four different methods, on the same fixed order of classes. They highlight advantages and disadvantages of each approach. Finetuning (d) the network on new tasks clearly leads to a complete forgetting of old classes. The LwF approach (c), instead, does prevent some catastrophic forgetting by implementing a distillation loss, but it's still very biased towards recently seen classes. ICaRL (b) proves to be closest one to Joint Training (a), efficiently making use of the exemplars to balance all classes seen.

Classification and distillation losses. Regarding the change in losses during training, we tried three different combinations: *CE+MSE*, *BCE+MSE*, *MSE+MSE* (whereas each term denotes firstly the classification loss and later the distillation loss). In particular, CE stands for *Cross Entropy*, BCE for *Binary Cross Entropy* and MSE for *Mean Squared Error*, also denoted as *L2 loss*. These well-known cost functions have been implemented and swapped with the original *BCE+BCE* setting in iCaRL which led to the results in 3.2. The different combination of losses have been selected based on some rough initial intuitive reasoning. The *Cross Entropy* loss is usually well suited for classification problems but it might struggle when used as a distillation loss, both because it is usually applied on one only output neuron and because it does not have an analytical minimum around the target value (see fig. 6). On the other hand, a *Mean Squared Error* loss might be a reasonable choice for a distillation loss in order to encourage the network to keep output values close to the old ones. It is important to note, though, that the previous losses act differently in terms of final values obtained, and different weights should be given to each term in order to make them comparable during training. This effect is easily seen in fig. 6, which shows a simple comparison between the three plain losses when used as distillation terms. Furthermore, the default BCE loss implemented in PyTorch automatically averages on the number of classes (not only on the mini-batch size) making even more important weighting both losses when evaluating the results.

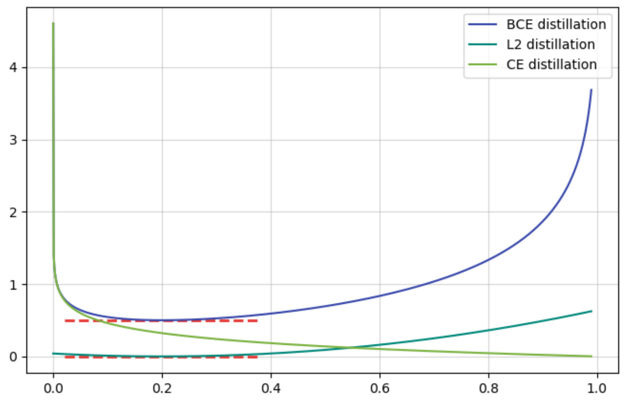


Figure 6. Behavior of the three different losses when used as distillation on a target response of 0.2. The target represents the old network output (sigmoid) that wants to be preserved in new classes. Note that BCE and L2 have a minimum in 0.2 in this case, encouraging the network to fluctuate around that value.

A new hyperparameter *alpha* is introduced for the MSE loss, indicating the amplitude of the parabola. Note that *alpha* has two intuitive interpretations: it can either be thought of as the curvature of the parabola (e.g. encouraging a higher loss when moving away from the target), or also as a general weight on the distillation term when using *CE+MSE* or *BCE+MSE*. The three formulas are later reported for clarity.

$$CE(p_y(x_i), y_i) = -y_i log(p_y(x_i)) \tag{4}$$

$$BCE(p_y(x_i), y_i) = -y_i log(p_y(x_i)) - (1-y_i) log(1-p_y(x_i)) \tag{5}$$

$$MSE(p_y(x_i), y_i) = \alpha(p_y(x_i) - y_i)^2 \tag{6}$$

where x_i is the input image i, $p_y(x_i)$ the binary probability of predicting class y for input x_i and y_i the target class for sample i. We remind that when the above losses are used as distillation terms, the targets y_i are instead equal to the old network outputs (sigmoid) that want to be preserved on new classes. Note that different approaches have also been explored by other papers, such as the *less forget constraint* proposed in [8] which sets as targets for distillation the old network features instead.

Changing the classifier. For our ablation study we also compare different classifiers on top of the feature extractor, and compare the results obtained with the standard *nearest-mean-of-exemplars* classifier used in the iCaRL implementation. We tried out three different classification methods: using *cosine normalization* as in [11], *k-nearest neighbors* (KNN), and *multi-layer perceptron* (MLP) classifier.

We implement *cosine normalization* to address the problem of class imbalance, i.e., that magnitudes of the parameters (weights and bias) for the new classes are significantly higher than those for the old classes as demonstrated in [8]. This approach involves the modification of the Resnet, which simply uses cosine similarity instead of dot product in the output layer. To implement this classifier the Resnet class has been modified adopting cosine normalization in the last layer and removing the ReLU in the penultimate layer to allow the features to take both positive and negative values. In order to do so, weights and features are normalized before the dot product, as described in the following equation:

$$p_i(x) = \frac{exp\left(\eta \left\langle \overline{\theta_i}, \overline{f(x)} \right\rangle\right)}{\sum_j exp\left(\eta \left\langle \overline{\theta_j}, \overline{f(x)} \right\rangle\right)} \tag{7}$$

where f and θ represent respectively the feature extractor and the weights both normalized. To control the peakiness of softmax distribution a learnable scalar η is introduced (not tuned in our experiments).

Later, image classification is performed by the *K-Nearest Neighbors* (KNN) approach [14]. Test pictures are labeled evaluating the K most similar training items by means of the Euclidean distance metric. Classification is so computed from a simple majority vote, assigning images to the data class which has the most representatives within its nearest neighbors. Furthermore, the parameter K is finetuned so that KNN can be less sensitive to noisy or irrelevant observations.

Finally, we adopt a feed-forward artificial neural network (ANN), that is the *multilayer perceptron* (MLP). In particular, model performed is composed by fully-connected layers: an input layer to receive the features about exemplars considered from the ResNet convolutionals, two hidden layer that distills some of the important patterns from the inputs and a final output layer to classify entry data into one of increasing CIFAR-100 categories. Each layer applies a linear transformation to the incoming data. The outputs of input layer and hidden layers are evaluated through an elementwise activation function, that is ReLU. The MLP is trained with a standard back-propagation algorithm each time new classes are coming and each training consists of 50 epochs. The learning rate starts at 0.01 and is divided by a factor 10 after 20 and 40 epochs. The exemplars considered are divided into minibatches, each of them having size equal to 100. Finally, the weight decay is equal to 0.00005. Before every training, the learnable weights and bias values of each module are initialized from the uniform distribution.

4.1. Results on ablation studies

In this section we provide the results of the different losses and classifiers previously stated to analyse the effect of each of them. Note that also in the ablation studies we do not use a validation set for simplicity, therefore grid searches are carried out based on the results obtained on the test set.

Results on different losses. With the introduction of *Mean Square Error* as distillation loss, we obtain considerably lower results comparing with our iCaRL implementation, in particular, 61.4% as average incremental accuracy. Further investigation has been done related to the hyperparameter of *distillation weight* (alpha) to understand the most convenient parabola's slope. Experimentally it has been seen that both increasing and decreasing it there have been no improvements compared to the default value (*alpha* = 1). This is likely since changing the distillation multiplier allowed to better remember the old classes at the expense of significantly less learning on the new batch of classes (in particular with *alpha* = 50 the average accuracy on novel classes decreased about 8.9% with respect to the iCaRL implementation). Table 2 shows the average incremental accuracy for the different *alpha* tuned values.

When implementing the *Cross Entropy* loss for classification particular attention must be put on the learning rate used. The high learning rate of 2.0 has indeed been tuned on the iCaRL settings based on a BCE loss with 100 outputs, it would likely not fit well in other scenarios with different losses. For all results concerning the use of the cross entropy a learning rate of 0.1 has been set, showing the best learning performances. Table 2 shows how the distillation weight *alpha* impacted the average incremental accuracy on different runs, when the combination *CE+MSE* is considered. Overall, a best value of 1 has been set.

On the *CE+MSE* combination, the results obtained were

Alpha	0.1	1	10	100
BCE + MSE	40.21	**61.42**	60.15	47.17
CE + MSE	60.88	**60.93**	60.85	59.34

Table 2. Global performances w.r.t. the distillation weight (alpha) when training the model with the combinations BCE+MSE and CE+MSE (classification and distillation losses).

lower with respect to our iCaRL implementation on all runs performed. It might be due to the fact that Cross Entropy is not computed on top of a standard softmax function on all outputs, but part of the network is controlled by the distillation loss.

Later, the *MSE+MSE* couple has been inspected and described below. In this particular case no distillation weight has been used since the classification and distillation losses are the same. Thus, *alpha* has been set to 1. If needed, more flexibility may still be obtained introducing a weight on the distillation term. Note that, even though the parameter alpha affects all neurons in the output layer during the loss computation, it is in common with all of them, thus only adjusting the learning rate is sufficient (see appendix A.2). With a best learning rate found of 5.0, fig. 7 reports the results obtained on the *MSE+MSE* combination with respect to iCaRL.

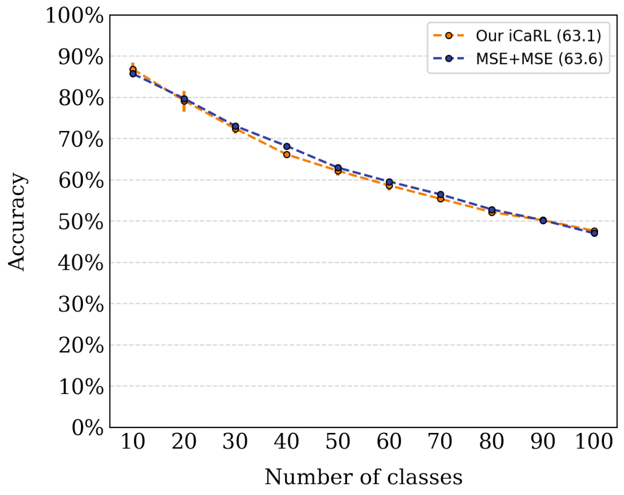


Figure 7. When using an L2 loss for both classification and distillation, the model slightly outperforms iCaRL in the short term but seems to consistently fall down on the final batches.

Note that the *L2+L2* loss slightly outperforms our iCaRL implementation with an average incremental accuracy of 63.6. From our experiments, it did show an interesting trend though: the accuracy would be on average higher on the first batches of classes, while eventually getting lower towards the very end. No more than 100 classes are given, but the L2 loss might eventually perform worse than iCaRL as we go on.

Finally, a slight different scenario has been investigated, which does not change neither losses or the classifier used. In the iCaRL setting, exemplars are added to the training dataset of the current batch of classes and the classification/distillation losses are applied to each training sample regardless of it being an exemplar or not. As a result, exemplars are being used during training with a distillation loss that aims at preserving their previously obtained sigmoid outputs. We tried changing this behavior, by forcing exemplars to be used as in standard classification, while maintaining the distillation loss on the novel samples only. In particular, for the loss computation, the exemplars targets have been set to the one-hot encoding of all 100 output labels (e.g. all zeros except for the correct class). This approach led to interesting results: predictions according to the network outputs reached iCaRL-like performances (the network keeps learning previous classes instead of just not forgetting them), but NME performances slightly dropped.

Results on different classifiers. Here we provide an overview of the results obtained with different classifiers to analyze the effect of each of them. With the introduction of *cosine normalization* on the last layer, we obtain significantly lower results compared to iCaRL. Specifically, the average incremental accuracy achieved with cosine similarity is 57.1 with respect to 63.1 obtained in Section 3.2 with iCaRL. Such low performance is likely since the application of cosine similarity to the network is changing not only the learning process of the network during training but also the classifier on the last layer. Indeed, the same network was subsequently trained based on the classification proposed in iCaRL (i.e. NME) showing results comparable to our implementation of such a method.

Regarding the KNN, different values of K have been tried in order to find the best-performing classifier. Though, our implementation of the K-Nearest Neighbors algorithm provides slightly worse results with respect to ones obtained by using original iCaRL-NME. To further try improving the performances of the algorithm, features have also been standardized. The best outcomes found for each K value during the experiments are summarized in table 3 where it is possible to see that the best accuracy has been obtained considering K equal to 9. Moreover, the same KNN grid-search shown above is performed applying *Neighborhood Component Analysis* (NCA) to select the principal features and so trying to relieve the curse of dimensionality problem, but contrary to what is expected, it has not improved the average accuracy score.

K	5	9	15	21
Average incremental accuracy	60.70	**62.66**	62.29	62.09

Table 3. Global performances w.r.t. K value.

Neurons in the hidden layers	128	256	512	1024
Average incremental accuracy	61.73	62.31	**62.43**	62.18

Table 4. Results of the grid-search on the number of neurons in the two hidden layers of the MLP, when the latter is placed for classification instead of NME.

Later, results obtained with the 3-layer MLP previously introduced are reported. Differently from the previous two classifiers, the *multi-layer perceptron* clearly needs far more attention in hyperparameter tuning. Fixing to two the number of hidden layers, the main component to be tuned is the amount of input and output neurons belonging to the hidden layers. The number of perceptrons per layer just tells us the width of the network. As it is not possible to explore the entire hyperparameters space, we study a small range of values that we consider representative of common architectures found in the related works and aim to explore the close range of the hyperparameters around successful settings from the literature. With this in mind, some combinations of values are performed, and has been noticed that when the number of neurons for each hidden layer has been kept the same, best results are obtained. Then, only the outcomes provided by this choice are considered. Clearly, the greater is the number of perceptrons, the greater will be the time spent to train the network. Relative results are shown in table 4. They are very similar, but the best average accuracy obtained during the incremental learning classification process is provided by the MLP having the two hidden layers composed of 512 neurons each.

Finally note that, in fig. 8, iCaRL still outperforms all classifiers on the very last batch, proving the assumptions that are carried on in [13] about the simplicity and effectiveness of NCM classification when samples are limited and noisy.

5. Our proposal

In this section, we describe and report in detail a variation of the standard iCaRL setting that we propose to alleviate one of the main problematics of building an incremental deep neural network, which is finding an effective feature

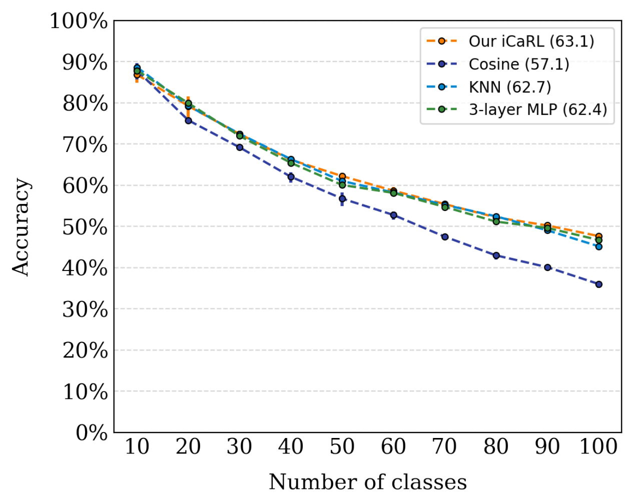


Figure 8. Final comparison among all classifiers cited, in their corresponding best configuration found.

space representation that makes it easy to distinguish among current and future classes. Our hypothesis states that if only the network could initially gain more knowledge on what are the relevant features to be extracted from images, then it could later find it easier to learn new labels without changing its structure so drastically. A simple analogy to this may be introduced by thinking of a 2-D world where all we knew was just circles the can be either red, green, or blue. At this point, we would think that the only thing that distinguishes objects from each other was their visual color, and it would be easy for us to learn about new colors if other circles were given to us. However, if one day a square appears, we would have to rethink our entire knowledge and tell ourselves that also the shape of the object was an important feature to take into account when classifying what we see.

In the real world, and specifically in deep learning scenarios, this often translates into needing more data at training time, or more frequently into the need of a transfer learning task from a pretrained network, prior to our analysis. Though, as well mentioned in the literature, class-incremental learning aims at finding more general models that can easily be updated on upcoming data with minimal effort in terms of resource overhead (memory and time), and transfer learning would make the whole point unfair in the specific case of computer vision for trivial classes (also note that doing transfer learning from pretrained networks on ImageNet on other datasets or domains does not always lead to better results). Thus, in order to keep a fair setting and encourage the network to build a more solid feature space representation from the start, we investigated how a different use of the number of exemplars throughout the incremental training process would make a difference. According to our hypothesis, for example, a larger number of exemplars may be used at the beginning to facilitate the network in finding a more correct feature space, and it might allow to use fewer exemplars in the later stages. We then propose to build a dynamic schedule for the number of exemplars used in each batch of classes, whereas iCaRL only keeps them to a fixed number (2000) throughout the whole process. Strongly note that while doing so we still do not want to generate an excess in the overall resource usage: the number of exemplars stored will only be arranged differently, but the average memory occupancy will globally remain the same. Finally, at the end of section 5.2 we also investigate how a temporary increase of the global memory available in the early stages impacts the quality of the model, to allow better decision making in resource allocation.

We first tested our hypothesis on the CIFAR-100 dataset, by storing all the incoming data until the fifth batch (50 classes) and suddenly reducing exemplars to 2000 at the end of the fifth batch, building the incremental model only from there to 100 classes. Fig. 9 shows how the accuracies on the second half of the training would change if such approach was chosen, with respect to iCaRL. Note that for results to be comparable, classification at the sixth step is still performed on 60 classes, meaning that exemplars are also taken from all previous tasks.

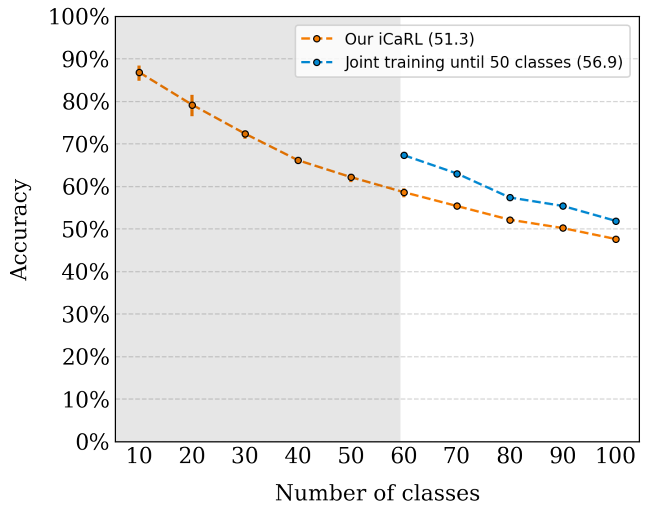


Figure 9. Accuracy at each incremental step when training the model with all incoming data until the fifth batch. The network seems to be able to learn a stronger feature representation for images, that even when resetting exemplars back at 2000 performances are significantly higher. The average incremental accuracy on the last 5 batches is reported in parenthesis

As shown, even when resetting exemplars back at 2000, performances are significantly higher on the second half of the training. We then further investigated our proposal.

5.1. Experiments

As previously mentioned, our set of experiments focuses on finding different arrangements of the number of exemplars stored such that the overall memory and computation time is still preserved and fixed to a predefined number. This means experimenting over a set of analytical functions (indicating the number of exemplars to be stored at each batch) whose mean value is 2000 (like in [13]) on the range from 0 to 9, included. More formally:

$$f \ : \ [0,9] \subset \mathbb{R} \ \to \ \mathbb{R}$$
$$f(x) \ \mapsto \ \text{"}\# \ of \ exemplars \ to \ be \ stored \ at \ batch \ x\text{"}$$

$$s.t. :$$

$$\frac{\int_0^9 f(x)dx}{9} = 2000 =: K \tag{8}$$

$$\wedge$$

$$g(x) := \frac{f(x)}{10(x+1)} \ : \ g(x_1) \geqslant g(x_2) \ \forall \ x_1 < x_2 \quad with \ x_1, x_2 \ \in \ [0,9] \tag{9}$$

For simplicity, the function f is drawn from real-valued functions of a real variable (even though we work on discrete batches), reminding that for linear functions in range 0 to 9 this still leads to an empirical average of 2000 exemplars stored. For non-linear functions (e.g. exponential), we redistributed any excess or lack of exemplars to reach the predefined value K of 2000 up to a 1% error. Note that one more constraint needs to be considered (see 9), as the number of exemplars stored per class cannot be increased in the process, but it can only be reduced (i.e. we cannot generate more samples from previous classes than we currently have).

Experiments have been conducted with all the following prototypes of functions in fig. 10. Each one of them has been carefully examined and tuned to find the best performing function for each type. Note that aside the different schedule in the number of exemplars stored, all other features and hyperparameters from iCaRL have remained the same, making this variation extremely easy to implement.

On top of the dynamic schedule, we also tried changing the standard behavior of exemplars during training, by forcing them to be used as in standard classification instead of as distillation samples only (as described in our ablation studies as well). We denote this as the exemplar classification trick. This small change encouraged the network to learn at times instead of just not forgetting, and proved to be best suited when a higher number of exemplars are used in the early stages. We were finally able to achieve the best performances (as shown in 5.2) by dynamically changing the exemplars behavior during training, using them for classification when exemplars are greater than 2000, and later reset their standard role to prevent forgetting.

5.2. Results and discussion

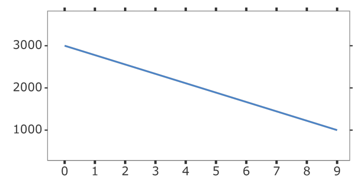
In this section, we report all results obtained in our experiments, together with a full discussion of advantages and disadvantages of our proposal.

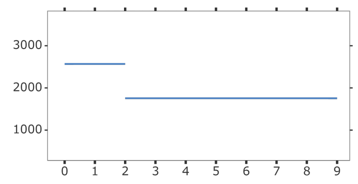
Dynamic number of exemplars. Before trying out different functions and performing a grid-search, we tested how a linear schedule in the number of exemplars (starting from 3000 and ending at 1000) would affect performances. Fig. 11 shows the results we obtained with our approach out-of-the-box.

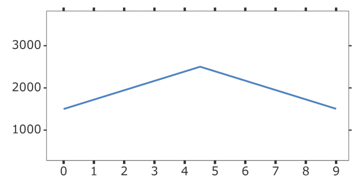
A linear change in the number of exemplars throughout the training process was indeed able to increase performances on all batches except the very last one, achieving overall an average incremental accuracy of 63.9. We then believe that by augmenting the number of exemplars used at the beginning of the training, the network was able to learn a richer and more solid feature representation for the task, which made it easier to learn new classes even with less than 2000 exemplars stored. Note, in fact, that from the 6th to the 10th batch the total number of exemplars used is fewer than iCaRL, but performances are still significantly affected by how the network learned in the first half. We may think of this as a *Hysteresis effect*[3] on the performances of the network. There is indeed a delay in the response of the model with respect to the number of exemplars currently stored, and only after further reducing the exemplars below 2000 we eventually go back to the original baseline performances. We exploit this effect to maintain the same average resource overhead while still achieving global better results. Note that this approach does require an architecture able to dynamically allocate storage as needed, and needs at times a higher maximum capacity (50% more).

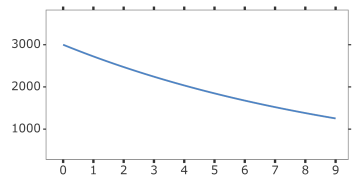
Different functions and optimizations. Later, all other functions previously cited have been tried, and all results obtained are reported in fig.12. Note that for each prototype a different grid-search (see table5) has been performed to find the best performing curve in terms of average incremental accuracy. On top of finetuning the hyperparameters, the classification trick on exemplars (as described at the end of section 5.1) has also been applied,

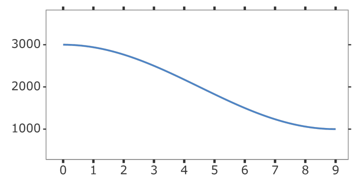
[3]https://en.wikipedia.org/wiki/Hysteresis


(a) Linear


(b) Step


(c) Triangular


(d) Exponential


(e) Cosine

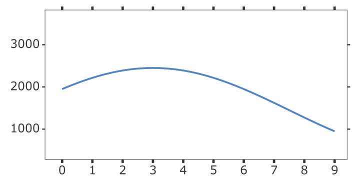

(f) Translated cosine

Figure 10. Prototypes of functions used in our experiments. For each, a new hyperparameter is introduced, being the number of starting exemplars. Note that all functions reported have a mean value of 2000.

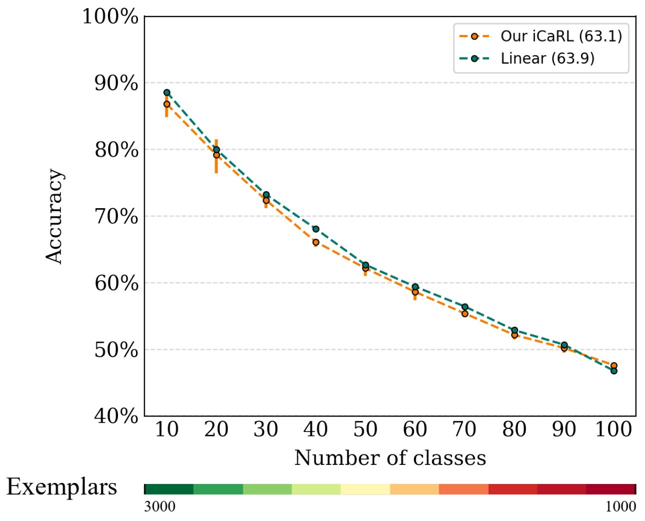


Figure 11. Accuracy at each incremental batch when the number of exemplars is linearly changed over the course of training, starting from 3000 and ending at 1000 exemplars to globally preserve the same average memory usage as iCaRL. The color bar indicates the number of exemplars stored at each batch during the process. Note that the new curve achieves better performances even when the number of exemplars is less than 2000, except for the last batch.

Starting # of exemplars	1500	2500	3000	3500
Linear	-	63.72	64.59	64.40
Exponential	-	64.30	64.45	64.06
Step	-	64.04 *	64.14	63.42
Triangular	64.07 *	-	-	-
Cosine	-	64.30	**64.78**	64.40

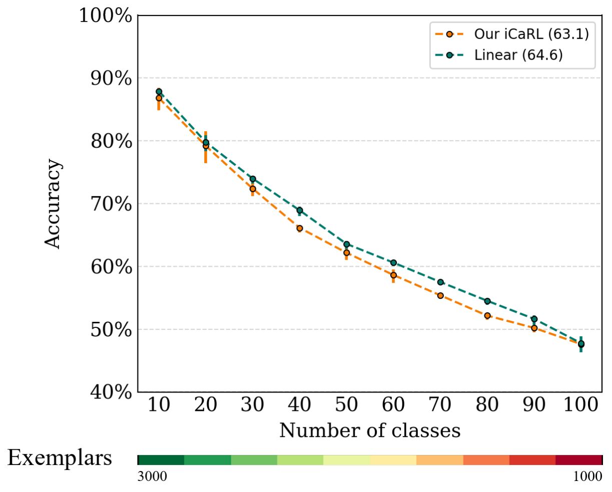
Translated Cosine	64.15 *

Table 5. Comparison between functions in terms of average incremental accuracy obtained. The "*" indicates that the given configuration achieved a higher score even on the last batch, with respect to our iCaRL implementation.

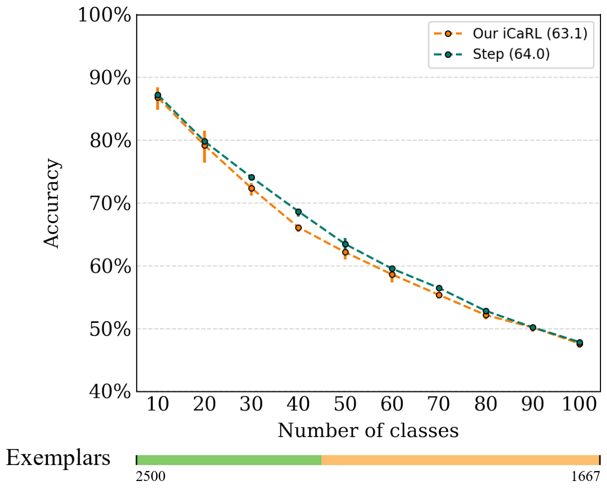
and consistently boosted the performances obtained. Indeed, some models achieve a higher accuracy even on the final batch (e.g. triangular functions), which showed to be the most critical one.

We were able to obtain higher results in terms of average incremental accuracy with all of the prototypes involved. This proved that a temporary increase in number of exemplars somewhere throughout the training would be always globally beneficial, even if later reduced. However, note that in some cases the performances on the very last batch are slightly lower, and it might not be wanted in some real case scenarios. If the model is meant to be used mainly on the run, while it's being incremented, than the highest boost in performances may be achieved with a cosine schedule, according to our experiments. Otherwise, if a best model in terms of final accuracy is needed, a cosine prototype, translated of 4 batches, slightly helps. Note that in both cases performances on the last batch are still very much comparable to our iCaRL implementation.

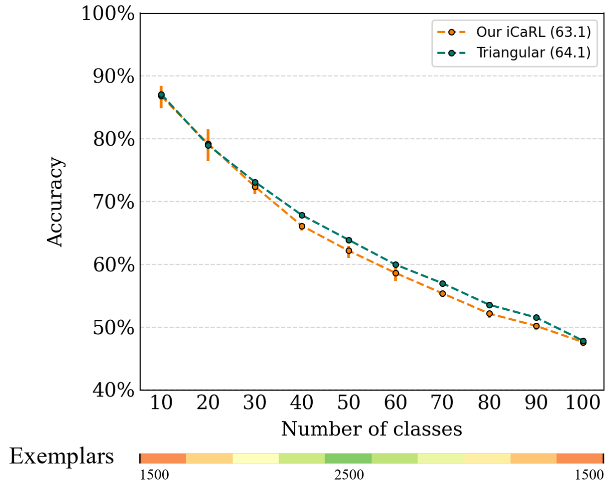
Overall, our proposal leads to a consistent gain in overall accuracy with minimum implementation effort, making it extremely likely to work on top of any other approach that is based on iCaRL and uses exemplars and a distillation loss. Though, it does require a scenario in which we suppose to have a somewhat accurate upper bound of classes, and up



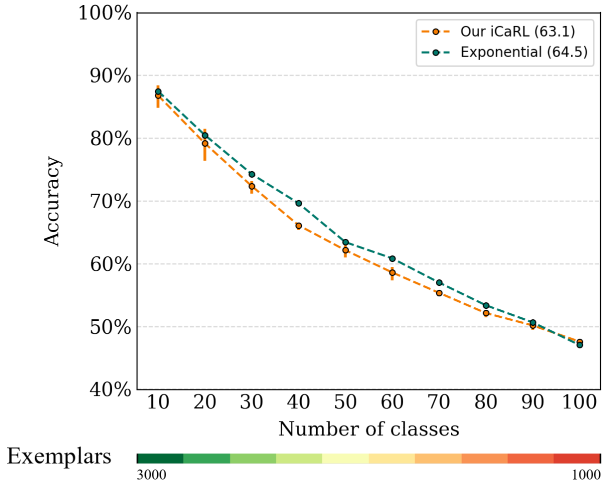
(a) Linear



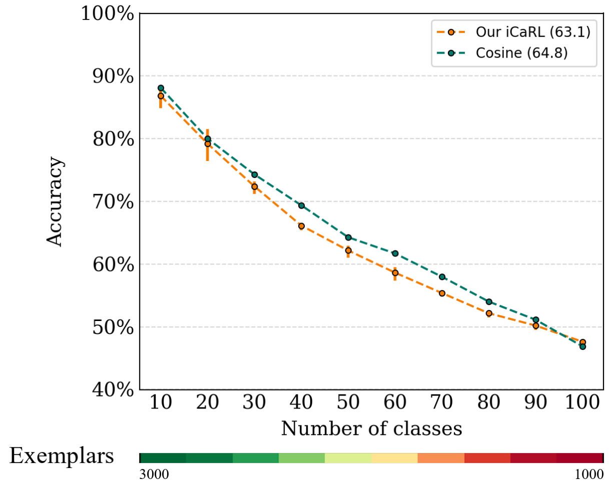
(b) Step



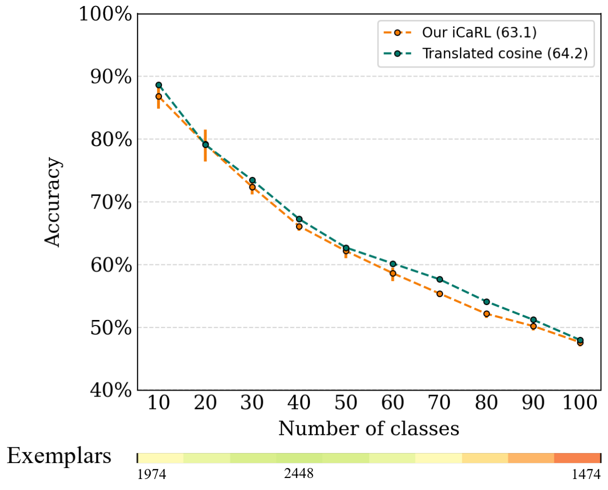
(c) Triangular



(d) Exponential



(e) Cosine



(f) Translated cosine

Figure 12. Results obtained with different prototype of functions, after a careful hypertuning of each of them. The color bars indicate what the current number of exemplars stored is for each batch on the x-axis. Interesting to note how the hysteresis effect on (f) leads to a boost in performances with a delay with respect to the maximum number of exemplars stored at the 4th batch. A cosine schedule gave us the best performing model in terms of average incremental accuracy.

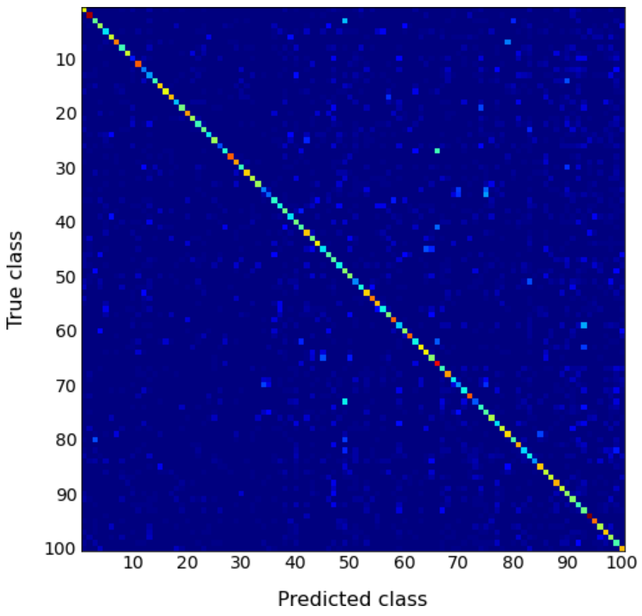


Figure 13. Confusion matrix corresponding to the cosine prototype, our best configuration found. Note that the global performance increase is not due to a better recognition of the first/last 50 classes (where the number of exemplars is above/below 2000), since predictions still show to be unbiased across both halves.

	Avg increm. accuracy	Resource cost	Ratio
S: 3500, Batch step: 4	+1.78%	+18.75%	0.095
S: 3500, Batch step: 3	+1.70%	+15%	0.113
S: 3000, Batch step: 4	+1.06%	+12.5%	0.085
S: 3000, Batch step: 3	+0.92%	+10%	0.092
S: 2500, Batch step: 4	+0.85%	+6.25%	**0.136**
S: 2500, Batch step: 3	+0.68%	+5%	0.136

Table 6. "S" indicates the number of exemplars initially stored, linearly decaying to 2000 after "batch step" batches. The ratio column computes the ratio among the first two values, to extract a possible quality measure.

to a +50% memory storage available. Finally, in table 6, we report the percentage increase in global performances with respect to an initial boost of memory available. Thus, in this case, the number of exemplars is higher at the beginning and it's later restored at 2000, with an increase of the total cost. We investigated this situation to allow better decision making in resource allocation, and study to what extent a larger expense in memory leads to better results.

6. Conclusion

In this work, we review and reproduce some of the most-known *class-incremental learning* approaches and propose a new variant that can be easily applied on top of each method that makes use of exemplars and a distillation loss. We perform experiments on the CIFAR-100 [9] dataset, on which we were able to successfully reproduce the results of LwF [10] and iCaRL [13] papers up to a margin of possible variation. Further ablation studies confirm that iCaRL strategies of *binary cross-entropy* and *nearest-mean-of-exemplars* outperform all changes in losses and classifiers we examined, except for a *MSE+MSE* configuration. Then, our variant is built on top of the iCaRL approach and proposes a dynamic scheduling of the number of exemplars used during the training process, such that the overall resource cost is still unchanged. This approach allows us to boost performances across 10 batches up to 1.5% (on average), likely by encouraging the network to learn a more solid feature representation to distinguish among the images given. A temporary resource increase is however needed, up to 50% more. Future directions worth exploring include studying how to manage an unknown number of incoming classes when it is hard to estimate a priori.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375, 2018.

[2] Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997.

[3] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[4] Ian J. Goodfellow, Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2014.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[7] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[8] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. June 2019.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[10] Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016.

[11] Chunjie Luo, Jianfeng Zhan, Lei Wang, and Qiang Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. *CoRR*, abs/1702.05870, 2017.

[12] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

[13] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016.

[14] Tülin undefinednkaya. A density and connectivity based decision rule for pattern classification. *Expert Syst. Appl.*, 42(2):906–912, Feb. 2015.

A. Appendices

A.1. Theoretical Proof

$$x : \ input \ image$$
$$\varphi(x) : \ features \ of \ x$$
$$\mu_y : \ mean \ vector \ representative \ of \ class \ y$$

$$y^* = \underset{y=1,\ldots,t}{argmin} \ \|\varphi(x) - \mu_y\| \ =$$

$$= \underset{y=1,\ldots,t}{argmin} \ \sqrt{2 - 2\cos\theta_{\varphi(x),\mu_y}}$$

$$\Updownarrow \ equivalent \ opt. \ problem$$

$$\underset{y=1,\ldots,t}{argmax} \ \cos\theta_{\varphi(x),\mu_y} \ =$$

$$= \underset{y=1,\ldots,t}{argmax} \ \mu_y^T \ \varphi(x)$$

A.2. Theoretical Proof

$$Loss_{weighted}(x_i) := \sum_{y=1}^{t} \alpha(o_y - y_i)^2$$
$$Loss(x_i) \ := \ \sum_{y=1}^{t} (o_y - y_i)^2$$
$$\frac{\partial}{\partial w_j} Loss_{weighted}(x_i) \ = \ \alpha \ \frac{\partial}{\partial w_j} Loss(x_i)$$

Note that adjusting the learning rate only still does not lead to an equivalent hyperparameter tuning, since, on the update of each weight of the network, the learning rate still also affects the weight decay term and it is itself affected by the scheduler.