Incremental Learning in Image Classification

Ivan D'Onofrio*[1], Gabriele Degola*[1], Luca Dibattista*[1], Fabio Cermelli[2], and Barbara Caputo[2]

Department of Control and Computer Engineering
Politecnico di Torino, 10129 Turin, Italy

[1] [name.surname]@studenti.polito.it
[2] [name.surname]@polito.it

Abstract

One of the main issues concerning machine learning systems nowadays is the inability to incrementally learn features from the external world without incurring unwanted effects or the inability to preserve previous knowledge.

In our work, we started by reproducing the experiments of iCaRL to have a better understanding of the incremental learning concept and of the problems of current implementations. We tried to solve some of the issues by proposing our variations of the iCaRL architecture based on deep generative approaches.

Our code is publicly available: `https://github.com/gabridego/MLDL20_Incremental_Learning`.

1. Introduction

Incremental Learning consists in generating a learner that is able to learn continuously without incurring what is technically defined as *catastrophic forgetting* [11].

Formally, we demand the following three properties of an algorithm to qualify as class-incremental:

(i) it should be trainable from a stream of data in which examples of different classes occur at different times,

(ii) it should at any time provide a competitive multi-class classifier for the classes observed so far,

(iii) its computational requirements and memory footprint should remain bounded, or at least grow very slowly, with respect to the number of classes seen so far.

To solve issues related to this kind of setting different solutions have been proposed in recent years. This work aims to reproduce and improve results obtained by *iCaRL* (*incremental classifier and representation learning*) [15], overcoming some of its main issues using a personal approach that will be described in Section 5.

iCaRL focuses the attention on preserving not only previous knowledge over time, but also a proper representation of it thanks to a system of prototypes. There are three main ideas behind the iCaRL concept:

- classification by a *nearest-mean-of-exemplars* rule (Section 2.2),
- *prioritized exemplar selection* based on herding,
- representation learning using *knowledge distillation* and *prototype rehearsal* (Section 2).

While in Section 3 we will focus our attention on the methods used to implement and reproduce the results historically obtained by iCaRL, in Section 4 we will report experiments on *CIFAR100* dataset [8] and different classifiers showing that iCaRL can class incrementally and modularly learn over a long period.

Additionally, we tried to exploit different approaches to overcome some of iCaRL's weaknesses. Although the system can incrementally learn new information over time effectively, it suffers from some limitations:

(i) The number of exemplars per class, defined apriori, shrinks at each iteration, leading with the increase of classes over time to a decrease in model performances.

(ii) Imbalance between new images and stored exemplars biases the representation of previous knowledge in favor of incoming data.

(iii) The herding method exploited by iCaRL tends to become less effective with classes characterized by a multitude of different representations, causing a loss of relevant information over time and overfitting.

*indicates equal contribution

We experimented two distinct approaches to face both issues:

- implementing a new herding system based on clustering to store in an efficient way bunches of heterogeneous and representative exemplars;
- extending the concept of *prototype rehearsal* increasing the number of exemplars per iteration generating a new set of balancing synthetic features using a generative approach.

In Section 5 we will focus our attention on these solutions reporting also the results of each experiment.

2. Related work

Several works deal with catastrophic forgetting, a consequence of the "stability-plasticity dilemma". Ideally, a neural network should be plastic enough to gain knowledge from new data, but stable enough to preserve old important information. Standard neural networks are characterized by excessive plasticity, forgetting most of the old information after training on new ones. In a class-incremental scenario, a straightforward way to solve this problem consists of training a new network every time new classes arrive on all the classes seen so far, but this is not feasible as previous information might be no more available and this solution is not able to scale.

Different solutions have been proposed to add new classes to an already trained model and the most effective ones are based on "rehearsal", training the network with the newly received data together with some previous information. In this work, we mainly focus on iCaRL, which adopts several interesting strategies to address the problem. First of all, it stores a fixed number of examples of old classes, called exemplars, in a prioritized way and uses them to preserve old information through distillation, a concept originally introduced by Hinton *et al.* [6] to transfer knowledge from a large to a small network and then adopted by Li and Hoiem [9] as *Learning without Forgetting* for task-incremental learning. Then, the exemplars are used in classification exploiting a "nearest-mean-of-exemplars" (NME) strategy, computing, according to the current network, the average feature of all the exemplars belonging to each class (prototypes) and the feature vector for the image to be classified, assigning the class label with the most similar prototype. This is an adaptation of the *nearest-class-mean* classifier [12] to the class-incremental setting, where the true class mean is not available as only the exemplars are stored.

Although iCaRL outperforms previously existing methods it presents some problems and later works try to improve it. One of the most critical points consists of the bounded memory requirement, which causes the number of stored exemplars per class to decrease each time new information arrives, with a high imbalance between old and new classes and a large bias towards the new ones. Hou *et al.* [7] tackle this problem using *cosine normalization* in the last layer of the network, as the features extracted from the new classes are higher in magnitude with respect to the old ones, and introduce the *Less-Forget Constraint* for distillation, encouraging the orientation of features extracted by the current network to be similar to those by the original model. Wu *et al.* [21] demonstrate that the last fully connected layer is biased and propose a *Bias Correction Layer*, which applies a linear function to the output of the new classes using some estimated bias parameters. This is particularly effective on large datasets.

Other works exploit additional generative models to generate synthetic examples from previously learned distributions and use them in place of the exemplars, an approach known as pseudo-rehearsal [16]. Shin *et al.* [18] are inspired by the human brain and combine a generator and a solver, both trained incrementally on the output data of the old generator. He *et al.* [5] instead train a conditional generative network for each class to better model the data distributions and train the classifier on a mix of real exemplars and generated data for the old classes. Interestingly, Liu *et al.* [10] generate features instead of images, which is an easier task for complicated datasets, and use them for feature distillation in the feature extractor and feature replay in the classifier.

2.1. Learning Without Forgetting

Li and Hoiem [9] tried to decrease the effect of catastrophic forgetting using the Knowledge Distillation loss [6]:

$$\mathcal{L}_{kd}^{\theta^t}(x) = - \sum_{c \in K^{t-1}} p_{\theta^{t-1}}^{c}(x) \log p_{\theta^t}^{c}(x) \tag{1}$$

where $p_{\theta^t}^{c}$ is the softmax probability that the network gives for class c at time t, and K^{t-1} is the set of classes known by the network at time $t-1$. The final loss is:

$$\mathcal{L}_{lwf}(x) = \mathcal{L}_{ce}^{\theta^t}(x) + \lambda_0 \mathcal{L}_{kd}^{\theta^t}(x) \tag{2}$$

where $\mathcal{L}_{ce}^{\theta^t}(x)$ is the cross entropy loss, and λ_0 is a loss balance weight.

2.2. iCaRL

Rebuffi *et al.* [15] improved the LwF approach using a classifier based on *nearest-mean-of-exemplars* and a selection method for exemplars based on herding.

The *nearest-mean-of-exemplars* assigns to the input image x the label:

$$y^* = \operatorname*{argmin}_{c \in K^t} \|\phi(x) - \mu_c\| \tag{3}$$

where $\phi(x)$ is the features vector of the image x, and μ_c is the mean of the features of the exemplars of class c.

The exemplars are selected by assigning to the k^{th} element (for $k = 1, \ldots, m$):

$$p_k \leftarrow \operatorname*{argmin}_{c \in K^t} \left\| \mu - \frac{1}{k}[\phi(x) + \sum_{j=1}^{k-1} \phi(p_j)] \right\| \tag{4}$$

where m is the number of exemplars per class to store.

2.3. Generative Feature Replay

Liu *et al.* [10] try to solve the imbalance between old and new classes through generative feature replay, exploiting a conditional Wasserstein Generative Adversarial Network to generate features for the previously seen classes. Many methods train a generator to generate images, but it is difficult to produce accurate samples for datasets with a large number of classes like *CIFAR100*, while feature generation is considerably easier. In this way exemplars are not necessary and a the relevant amount of memory previously occupied by the K exemplars is freed. In addition, the information is preserved over time using a replay alignment loss $\mathcal{L}^{RA}_{G_t}$, introduced in [20], as distillation for the generator, encouraging the current generator G_t to produce the same feature as the previous one G_{t-1} when conditioned on a old class c and on a given latent vector $\mathbf{z}$ sampled from a Gaussian distribution p_z. Formally, it is defined as:

$$\mathcal{L}^{RA}_{G_t} = \sum_{j=1}^{t-1} \sum_{c \in C_j} \mathbb{E}_{\mathbf{z} \sim p_z} \left[\| G_t \left(c, \mathbf{z} \right) - G_{t-1} \left(c, \mathbf{z} \right) \|_2^2 \right] \tag{5}$$

and is summed during training to the standard conditional WGAN loss for the generator:

$$\mathcal{L}^{WGAN}_{G_t} \left(\mathcal{X}_t \right) = -\mathbb{E}_{\mathbf{z} \sim p_z, c \in C_t} \left[D_t \left(c, G_t \left(c, \mathbf{z} \right) \right) \right] \tag{6}$$

where $\mathcal{D}_t = (\mathcal{X}_t, \mathcal{C}_t)$ contains the new images and labels at time t. The current generator G_t is trained against the discriminator D_t, whose objective is to mark as real the images belonging to $\mathcal{D}_t$ and as fake the images generated by G_t. Therefore, it is trained according to the conditional WGAN loss:

$$\begin{aligned} \mathcal{L}^{WGAN}_{D_t} \left(\mathcal{X}_t \right) = &+ \mathbb{E}_{\mathbf{z} \sim p_z, c \in C_t} \left[D_t \left(c, G_t \left(c, \mathbf{z} \right) \right) \right] \\ &- \mathbb{E}_{\mathbf{u} \sim \mathcal{D}_t} \left[D_t \left(c, \phi_t \left(\mathbf{x} \right) \right) \right] \end{aligned} \tag{7}$$

Finally, feature distillation is performed forcing the new feature extractor ϕ_t to produce similar features to the ones of ϕ_{t-1}, minimizing their L2 distance:

$$\mathcal{L}_{\phi_t} \left(\mathcal{X}_t \right) = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}_t} \left[\| \phi_t \left(\mathbf{x} \right) - \phi_{t-1} \left(\mathbf{x} \right) \|_2 \right] \tag{8}$$

3. Method

To compare our results and variations with the experiments performed by Rebuffi *et al.* [15] we have used a ResNet32 on *CIFAR100* with the same hyperparameters of the original paper. When used, the number of exemplars is fixed to $K = 2000$. Each training phase consists of 70 epochs. The learning rate is initially set to 2 and is divided by 5 after 49 and 63 epochs when using binary cross-entropy loss for both classification and distillation. A batch is composed of 128 images and the weight decay is 10^{-5}. Before training, the images are augmented random cropping 32 pixels with 4 pixels of padding and flipping horizontally with probability 0.5.

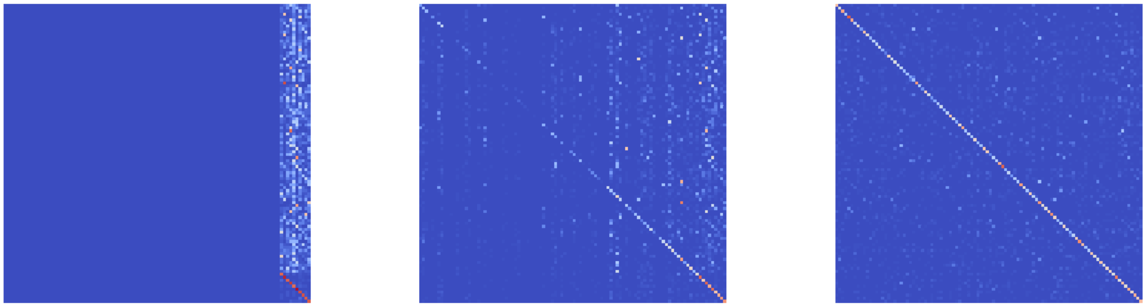
We incrementally trained the network with one batch per time, each containing 10 classes. We performed three experiments with different classes per batch, randomly dividing the training set into the 10 batches three times, and then we used the same setting for the subsequent experiments to guarantee consistence. In addition, we keep 10% of the images to validate the methods on the current batch of 10 classes, while they are tested on all the test images belonging to the previously seen classes.

To study iCaRL and how it deals with the incremental learning problem, we implemented and compared four models: fine-tuning baseline, *Learning without Forgetting* in a multi-class scenario, *hybrid1* and, finally, iCaRL. Each model extends and improves the previous one and their performances on the test set per-batch of 10 classes are compared in Figure 2a and are reported in the first rows of Table 1. In addition, confusion matrices are reported in Figure 1.

The first step is the fine-tuning baseline, in which the model learns new classes without the distillation loss but with only the classification one. The classification is performed by the last fully connected layer and the accuracy is good on the first 10 classes but immediately drops on the next batches, as the network completely forgets the previous classes when it receives new information, falling into *catastrophic forgetting*.

Then the distillation term is added to the classification loss, as explained in Section 2.1 and in [15]. In this way, the network is encouraged to predict the correct label for the new class, while reproducing the scores of the previous network for the old classes. As shown in the graph, the model can retain some previous knowledge but the performances are still far from being satisfying.

The third model is the first hybrid setup used in [15], adding the exemplars-based rehearsal during training but still using the fully connected layer for classification. Basically, this model is equivalent to iCaRL without the NME classifier. The total number of exemplars is bounded, therefore the number of exemplars per class continuously shrinks



(a) Finetuning

(b) LwF.MC

(c) iCaRL

Figure 1: Confusion matrices of different methods on *CIFAR100*. The classes are represented by order of arrival, from left to right. The horizontal axis represents the predicted classes, while the vertical axis represents the real ones.

from 200 after the first batch to 20 when the model is trained on all classes. The introduction of exemplars has a high impact on accuracy.

Finally, the full iCaRL model is implemented, adding the classification through *nearest-mean of exemplars* to the *hybrid1* solution. The obtained results are comparable with the ones of [15], which are obtained on ten runs instead of three, so we can say that the model has been successfully replicated. The confusion matrix in Figure 1c shows that the network performs well even in the first classes it has been trained on. It is important to underline that the exemplars are stored without the previously mentioned data augmentation but are transformed before the training phase. In this way, they are comparable with the test images and can be correctly used for classification, as no data augmentation is performed on the test set. In addition, we experimented with two different policies for exemplars selection, taken randomly or according to the herding policy described in Section 2.2. As reported in Table 1, we are able to demonstrate that the model with herding obtains negligibly better performances, despite its higher complexity and required time with respect to random selection. For this reason, in the next experiment, the exemplars are randomly selected unless otherwise indicated.

3.1. Classifiers

The last layer of an artificial neural network is almost always a fully connected layer with a number of neurons equal to the number of classes. Despite being the most straightforward way to classify, it is possible to use the features extracted from the previous layers of the network and use them for other classifiers.

As in iCaRL, we applied the *nearest-mean-of-exemplars* (NME) classifier discussed in Section 2.2. It computes, for each class, the mean vector of the features of the elements that belong to that class. When the input x has to be classified, the NCM assigns to it the class with the closest mean to its features vector.

With a similar reasoning and inspired by [7] we tried classification trough *cosine similarity*. We computed the mean vectors of the features for each class and, for each input x, we extract and normalize the features vector and assign it to the class with the highest cosine similarity. In this way, we take into consideration the spatial orientation of the extracted features, which may contain important information.

Support Vector Machines (SVMs) are the state-of-the-art large margin classifiers. SVMs may be used in the training phase [19], but we decided to use a fully connected layer during the training, and replacing it with an SVM for classification. We used the radial basis function kernel with $C = 1.0$ and $\gamma = (n_{\text{features}} \cdot \sigma^2)^{-1}$, where n_{features} is the number of features (the number of neurons of the penultimate layer) and σ^2 is the variance. During the training phase we used the Soft Nearest Neighbor Loss [4] (SNN loss) described in Section 3.2.3 to better separate images of different classes.

We tried other classification methods like k-nearest neighbors (k-NN) [14] with $k = 5$ and random forest (RF) [2] with 100 trees.

3.2. Losses

iCaRL uses binary cross-entropy loss for classification and the LwF loss for distillation, but several choices are possible and can improve the results. In this section we describe few possible losses, while in Section 4 we analyze some combinations, their motivations, and their performances.

Table 1: Incremental learning results (accuracy, $\mu \pm \sigma$, $n_{\text{samples}} = 3$) on *CIFAR100* dataset with an increment of 10 classes. Finetuning and LwF [9] do not use any exemplars from the old classes. *hybrid1* [15], iCaRL [15], iCaRL with a random selection of the exemplars, and all the variations use the same amount of exemplars from the old classes. The first lines contain the results that we obtained by replicating the iCaRL paper. The middle lines contain the iCaRL implementation with different losses (binary cross-entropy, mean square errors, Less-Forget constraint, Soft Nearest Neighbors). The bottom lines contain the results combining the iCaRL implementation with different classifiers (Support Vector Machine, k-NN, Random Forest).

Model	Mean accuracy	1^{st} batch accuracy	10^{th} batch accuracy	Gap 1^{st} to 10^{th} batch
iCaRL	0.6076 ± 0.1210	0.8540 ± 0.0422	0.4681 ± 0.0004	45%
iCaRL (random policy)	0.6000 ± 0.1224	0.8430 ± 0.0548	0.4567 ± 0.0135	46%
hybrid1	0.5486 ± 0.1555	0.8560 ± 0.0394	0.3629 ± 0.0056	58%
LwF.MC	0.4382 ± 0.1875	0.8483 ± 0.0468	0.2424 ± 0.0084	71%
Finetuning	0.2527 ± 0.2214	0.8490 ± 0.0484	0.0911 ± 0.0030	89%
iCaRL + MSE loss	0.6153 ± 0.1190	0.8527 ± 0.0403	0.4752 ± 0.0078	44%
iCaRL + BCE + MSE loss	0.5745 ± 0.1499	0.8553 ± 0.0556	0.3913 ± 0.0068	54%
iCaRL + LFc	0.5563 ± 0.1627	0.8593 ± 0.0448	0.3517 ± 0.0101	59%
iCaRL + SNN loss	0.4756 ± 0.1486	0.7940 ± 0.0430	0.3060 ± 0.0010	61%
iCaRL + cosine similarity	0.6058 ± 0.1210	0.8527 ± 0.0452	0.4676 ± 0.0036	45%
iCaRL + SVM	0.5998 ± 0.1227	0.8447 ± 0.0527	0.4555 ± 0.0071	46%
iCaRL + SVM + SNN loss	0.4731 ± 0.1465	0.7840 ± 0.0360	0.3011 ± 0.0051	62%
iCaRL + 5-NN	0.5857 ± 0.1366	0.8527 ± 0.0444	0.4176 ± 0.0010	51%
iCaRL + RF	0.5818 ± 0.1377	0.8547 ± 0.0425	0.4164 ± 0.0096	51%

3.2.1 Binary Cross Entropy Loss

We started our experiments with the Binary Cross Entropy Loss (BCE loss) [13]:

$$\mathcal{L}_{bce}(x, y) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \tag{9}$$

where y_i is 1 if the i^{th} element belongs to a certain class, 0 otherwise and $p_i \in [0, 1]$ is the probability that item i belongs to that class.

Since the BCE loss deals with binary classification, we used a one-hot encoding for the target and the sigmoid function for the output. iCaRL uses BCE for distillation too, concatenating the output of the old network and the actual labels to balance the contribution of old and new classes.

3.2.2 L2 loss

L2 loss function is the mean squared error between the true and the predicted value:

$$\mathcal{L}_{mse}(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2 \tag{10}$$

This loss can be used for both classification, minimizing the L2 distance between the probability that an element belongs to a class and the one-hot encoding of the actual class as presented above, and for distillation, for example, computing the distance between the features obtained from the old network and the new ones.

3.2.3 Soft Nearest Neighbor Loss

Frosst *et al.* [4] introduced, in 2019, the Soft Nearest Neighbor Loss (SNN loss) for a batch of n samples (x, y):

$$\mathcal{L}_{snn}(x, y, T) = -\frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{\sum_{\substack{j \in [n] \\ j \neq i \\ y_i = y_j}} e^{-\frac{\|x_i - x_j\|^2}{T}}}{\sum_{\substack{k \in [n] \\ k \neq i}} e^{-\frac{\|x_i - x_k\|^2}{T}}} \right) \tag{11}$$

where T is the temperature.

SNN loss is a measure of the entanglement of the labeled images. High values of the loss means that the classes are muddled up, while it is easier to separate classes with a lower value of the loss. For this reason, we used this loss with the SVM.

The SNN loss requires high values of batch size since it computes the distance between points on the batch only. If in a single batch there are few points per class (eventually outliers of their classes) the loss value is not reliable.

The temperature has an important role in the loss value. Fontanel *et al.* [3] suggest to set $T = \sigma^2$ to keep the loss value as stable as possible.

By increasing the batch size to 1024 and using $T = \sigma^2$ we increased the accuracy on the test set on the first batch from 0.472 to 0.694.

We tuned the learning rate, that we set to 1.0, and we increased the number of epochs to 150. The learning rate was multiplied by 0.2 after 105 and 135 epochs (respectively $\frac{7}{10}$ and $\frac{9}{10}$ of the number of epochs, as for iCaRL).

In Section 6 we discuss how to improve more the network using the SVM and the SNN loss.

3.2.4 Less-Forget constraint

Hou *et al*. [7] propose *Less-Forget constraint* (LFc) as distillation loss, alternatively to LwF. They freeze the network trained on the old classes and compute the distillation loss on the features as:

$$\mathcal{L}_{lfc}(x) = 1 - \langle \bar{\phi}_{t-1}(x), \bar{\phi}_t(x) \rangle \tag{12}$$

where $\bar{\phi}_t$ is the normalized features vector at time t. This encourages the orientation of the features extracted by the current network to be similar to the features extracted by the old ones, with the aim of preserving the previous knowledge.

4. Ablation study

Before proceeding with our personal approach to the class-incremental learning problem, we propose a simple ablation study with some modifications to the key features of iCaRL: the distillation loss and the NME classifier. Here we describe the results of some experiments performed with some combinations of the losses and classifiers presented in Sections 3.1 and 3.2 and provide motivations for our implementation choices. The results of the performed experiments are reported in the second and third sections of Table 1 too. In the next experiment, when using different losses for classification and distillation, the total loss function is computed as:

$$\mathcal{L} = \lambda \mathcal{L}_d + (1 - \lambda)\mathcal{L}_c \tag{13}$$

where $\mathcal{L}_d$ is the distillation loss, $\mathcal{L}_c$ is the classification loss and $\lambda = \frac{n}{n+m}$, where n and m are the numbers of old and new classes respectively, is used to balance between the two losses and consider each class equally.

Regarding the different choices for the losses, we started using L2 loss instead of binary cross-entropy for both classification and distillation, maintaining the same hyperparameters as before. In this way, we are forcing the network predictions to be near in L2 norm to the ground truth for the new classes and the outputs for the old classes to be similar to the outputs of the old network. This is conceptually similar to what is done by iCaRL and achieves slightly better performances.

Then we tried a different approach, using BCE for classification and MSE for distillation, encouraging the features extracted by the current feature extractor to be similar to the ones of the old model. This distillation has been formalized in equation 8 and is particularly useful when it is required to maintain similar feature representations over the training step. In addition, we tried to include it in our modification, as detailed in Section 5.2.2. With the fixed hyperparameters, the results are not satisfying.

As a third combination, we investigated the previously mentioned methods to reduce the natural bias of an incremental setting towards the new classes. We implemented the Less-Forget constraint introduced in [7] as an alternative to Learning without Forgetting and trained the network with the BCE as classification loss. However, we were not able to replicate the results of the paper and the performances were very poor with respect to the standard iCaRL. The work by Hou *et al.* includes the contribution of a third complex loss, called margin ranking loss, to enforce the inter-class separation in the feature space and rely on a cosine normalization layer to preserve the same order of magnitude for features in different learning stages. In addition, they dynamically tune the parameter λ and their method has proven to be particularly effective if the first batch contains more classes than the others, as they start with 50 classes in the first group. The behavior of our model with these losses is reported in Figure 2b.

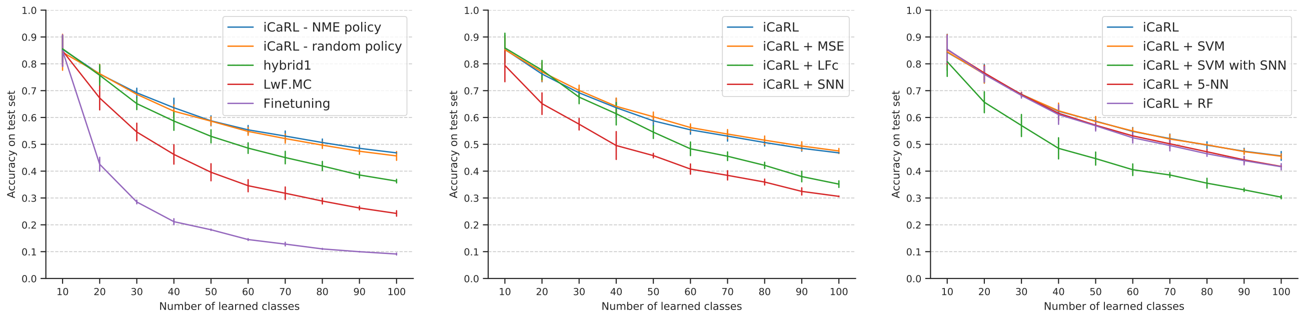
Regarding the classifiers, we implemented different traditional machine learning classifiers as described in Section 3.1, such as Support Vector Machines, K-Nearest Neighbors and Random Forest, plugged in place of the last FC layer of the network trained with the default iCaRL setting and used for classification in place of NME. They all obtained similar performances and are comparable with iCaRL, as represented in Figure 2c. Even classification through cosine similarity performs similarly to iCaRL, so the mean and the spatial orientation of the stored exemplars in the feature space probably carry similar information.

Finally, we experimented with SVM applied on a model trained with SNN loss for both classification and distillation and the performances are worse than iCaRL, as analyzed in Section 3.2.3. However, this deserves further discussion and is addressed in Section 6.

5. Personal implementation

In this section, we discuss some personal experiments carried out trying to overcome some of iCaRL limitations. Specifically, we focused our attention on two distinct steps of system pipeline:

(i) Exemplars herding, affected by overfitting and information loss as the classes to remember increases over time.



(a) Replica of iCaRL experiments.

(b) iCaRL with different losses.

(c) iCaRL with different classifiers.

Figure 2: Incremental learning results (accuracy) per batch with settings of Table 1.

(ii) Limited exemplars storage, which in the long run leads to a strong bias between the classes to remember in favor of the new one.

To overcome these issues, we propose below some alternatives designed to artificially increase the number of exemplars involved in *prototype rehearsal* while maintaining the system scalable, and others to improve the herding system in the exemplars selection phase preserving representation distribution.

5.1. Herding with Affinity Propagation

One of the main weaknesses of the original iCaRL herding system is the choice of exemplars based on minimizing the euclidean distance between candidates and the current exemplars representation mean. In the long run, this setting could lead to a loss of information.

(i) Images and associated feature maps are generally characterized by heterogeneous representations. Selecting the ones which are closer to the exemplars mean means carry out *prototype rehearsal* using only images whose representation is comparable to the most common and homogeneous distribution within a class, without considering remaining heterogeneous information;

(ii) The previous statement implies that the model is also subject to overfitting since the information that is propagated through time by the system is just the representation of the most common class distribution.

We were able to demonstrate so far that iCaRL achieves similar performances both using *nearest-exemplar mean* and *random* herding policies. In this section, we will explain an alternative solution based on clustering that achieved slightly better performances compared with the previous ones. We implemented a new policy based on *Affinity Propagation (AP)* (Algorithm 1) clustering algorithm based on the concept of “message passing” between data points.

Algorithm 1: Affinity propagation

```
Input: Similarity matrix S between the individuals
        to be classified.
Output: Clusters grouping individuals who are
         alike.
Construct the graph from S;
Initialize availabilities;
Update availabilities and responsibilities:

    $$R(i,k) = s_{i,k} - \max_{k' \neq k} \left\{ A(i,k') + s_{i,k'} \right\}$$

    $$A(i,k) = \begin{cases} \min \left\{ 0, R(k,k) + \sum_{i' \in \{i,k\}} \max \left\{ 0, R(i',k) \right\} \right\} & i \neq k \\ \sum_{i' \notin (i,k)} \max \left\{ 0, R(i',k) \right\} & i = k \end{cases}$$

 while convergence not reached do
     Update availabilities and responsibilities
       according to a damping factor $\lambda \in [0,1]$ ant $t$
       number of current iteration:

     $$R^{(t+1)}(i,k) = (1-\lambda) R^{(t+1)}(i,k) + \lambda R^{(t)}(i,k)$$

     $$A^{(t+1)}(i,k) = (1-\lambda) A^{(t+1)}(i,k) + \lambda A^{(t)}(i,k)$$

end
foreach individual $i$ do
     Determine of the representative $k$:

          $$\arg\max_k (A(i,k) + R(i,k))$$

end
```

The feature that makes this method suitable for our purposes is that *AP* does not require the number of clusters to be determined or estimated before running the algorithm and it can find “exemplars”, members of the input set that are representative of clusters.

That approach allowed us to overcome and improve is-

Algorithm 2: Clustering policy based on AP

```
Input: Representation $\psi$ of each input image.
Output: Set of images $E^t$ as exemplars for class $t$.
Apply AP algorithm on $\psi(X^t)$ where $X^t$ is the set
  of all images belonging to a new class $t$ at current
  training step;
Sort $x \in X^t$ by asc. euclidean distance from its
  centroid and desc. number of elements of its
  cluster;
Take first $n$ images from sorted $X^t$ where $n$ is the
  number of exemplars that have to be stored in
  current training step:
      $n = \left\lfloor \frac{K}{\text{known classes}} \right\rfloor$
      $E^t \leftarrow X_i^t \quad i = 1, ..., n$
```

sues related to *nearest-exemplars mean*, storing as exemplars not only images coming from most common representation but also taking into account more heterogeneous representation identified by the clustering algorithm.

Specifically using this method we were able to:

- Preserve overtime at least the representation of the centroids obtained applying *AP* sorted by cluster cardinality and, additionally, some of the representations that best describe their centroid by euclidean distance.

- Since exemplars are stored sorted by importance, it is possible to scale easily the number of exemplars stored per class through time getting rid of the last elements of these sets.

Nevertheless, this method is subject to some throwbacks:

- When the number of exemplars per class shrinks, it is not possible to preserve information about large clusters, totally described by their centroids only.

- When the number of clusters is large, it is hard to preserve at least one exemplar for each of them.

5.2. Pseudo-rehearsal

In *pseudo-rehearsal* setting, instead of using a chosen subset of the dataset observed so far to preserve classes already learned, exemplars used to refresh model memory are generated by an auxiliary agent.

We exploited this method trying to improve the standard exemplars selection preserving its scalability since model memory requirement needs to remain constant - or at least bounded - until the end of the process. Specifically, we implemented two distinct methods able to generate synthetic data from knowledge learned up to this point:

(i) A stochastic generator based on known class features mean representation.

(ii) A deep incremental WGAN-based generator able to generate synthetic features trained on data observed so far.

The two solutions are characterized by different complexities, but both were designed to perform the same function: propagate over time previous knowledge in order not to make the system forget its knowledge.

These are straightforward since we request them not to generate images to train the entire network, but just to produce a synthetic features-set containing elements similar to the output of the last layer of the ResNet feature extractor to train and make its classifier able to remember previous classes.

5.2.1 Random pseudo-rehearsal

To increment the number of exemplars without using more memory or, eventually, using $K = 0$ exemplars, we tried the pseudo-rehearsal method by Robins [16], that we will call random pseudo-rehearsal. It consists of generating random images by assigning 0 or 1 with the same (50%) probability to each bit of the image. After generating the image, it will be forwarded through the network to compute the label and use it as an exemplar.

We observed that using a random image generation approach, after the first batch more than 70% of the images belonged to the same class. After the fourth batch, all the images belonged to the same class.

Since adding exemplars to just one class is not balanced and may worsen the performances, we used the approach described in Algorithm 3. We decided to generate a vector of features instead of an image. For each learned class, we used the mean and the standard deviation of the exemplars' features that we had already computed when we got the representation of the exemplars to choose which ones to store.

We generated m random vectors, each one from a normal distribution with mean and variance of that class. Since having the same mean and standard deviation of a certain class does not mean that the generated representation belongs to that class, we forwarded the image through the network to compute its label. We added the generated features and the computed label to the training dataset. With n_{rpr} features vectors generated, and n_{batches} per epoch, we put $n_{\text{rpr}}/n_{\text{batches}}$ random features vectors per batch, randomly

Algorithm 3: Not-so-random pseudo-rehearsal

```
Input: $\mathcal{C}$ (set of learned classes)
Input: $m$ (images per class to generate)
Output: Set of not-so-random features generated
foreach $c$ in $\mathcal{C}$ do
    $\mu_c = \underset{x \in c}{\text{mean}}(\phi(x))$
    $\sigma_c = \underset{x \in c}{\text{std}}(\phi(x))$
    Initialize empty array $\phi_c^{\text{nsr}}$
    for $i$ = 1, ..., $m$ do
        $\phi_c^{\text{nsr}}[i] = \mathcal{N}(\mu_c, \sigma_c^2)$
    end
end
```

sampled from the generated ones. We forwarded the images through the entire network, and the generated features through the last fully connected layer.

5.2.2 GAN-driven pseudo-rehearsal

The most intuitive approach to pseudo-rehearsal is to train a generative model on real images and use them to sample images for the old classes from the learned distribution, which will then be used in place of the exemplars. This approach is followed for example in [18] and [5]. However, they obtain good results on *MNIST*, a much simpler dataset than *CIFAR100*, or use a separate model for each learned class, which goes against our bounded-memory requirement and may become infeasible when the number of classes becomes large. At first, we tried to adopt a similar strategy to [18] to keep constant the number of available examples per class k (for example 100) while the occupied memory remains bounded. When a new batch of 10 classes arrives, the network is trained on the new data together with the stored exemplars and, if they are not enough, different conditional Generative Adversarial Networks have been tried to generate synthetic exemplars, requiring images of the old classes. After training iCaRL's ResNet, the GAN is trained on the new data together with the exemplars and the previously generated images for distillation.

However, a similar approach presents some problems, as at each batch the new GAN is trained on the images generated by the old one, possibly propagating error and inaccurate representations. For the same reason, generative networks must perform almost perfectly and have to be trained for a very large number of epochs, which was not possible with our available resources and can lead to overfitting if the hyperparameters are not set correctly.

For these reason, we adopted an approach similar to [10], explained in Section 2.3. Instead of generating images, we exploited a conditional Wasserstein GAN [1] to generate synthetic features. As mentioned before, feature generation is easier than image generation on large datasets like *CIFAR100*, while WGANs can improve stability with respect to other generative networks thanks to a different loss function. That also allows reducing the overhead and complexity of the feature extractor.

Generate features for FC layer: First, we used the features generated by the WGAN to distill the knowledge in the fully connected layer. When the network receives the first batch of classes, it is trained according to the standard binary cross-entropy loss and the generator and discriminator are trained on the same images, according to the WGAN loss together with a conditional loss, which encourages the discriminator to correctly classify the features and the generator to sample from the required class distribution. At the next iterations, some features are generated, in order to have a minimum amount k of information for each class, and are passed through the last fully connected of both the new and the old network. The obtained values are concatenated with the networks' outputs for stored exemplars and used for distillation as in LwF. The GAN is trained as before, with the additional replay alignment loss for the generator described in Section 2.3, to avoid forgetting in the feature generator.

Generate features for classification: The previous solution introduces pseudo-rehearsal through generated features and helps the network to better remember previous knowledge. However, it does not adapt to a situation in which exemplars are not available, as it still relies on them for classification through NME. It is intuitive at this point to pass from *nearest-mean of exemplars* to *nearest-mean of generated features*, computing the mean of the representation of a given class considering not only the features extracted by the stored exemplars but also the features sampled by the generator, or considering only the latter in the most extreme cases. To guarantee consistent representations for the same classes at different training stages, we add a similar loss to the one of equation 8, to minimize the mean squared error between the features extracted at subsequent training steps.

In our case, this did not provide good results, with a heavy performance drop with respect to iCaRL. We can explain this behavior as we trained the WGAN for 200 epochs, with Adam optimizer and learning rate 10^{-4} for both generator and discriminator, and that may not be enough to correctly learn the class distribution and so provide good classification when the number of real exemplars is small, and it is not feasible to train for a much longer number of epochs or to perfectly tune the parameters with our available resources. In addition, the ResNet32 we used for *CIFAR100* has a feature space of 64, so the variability of the possible representations is quite limited. The works that obtained good results with similar methods trained the whole network for a very long time having at their disposal a lot

Algorithm 4: WGAN-driven pseudo-rehearsal

```
Input: $\mathcal{C}$: set of learned classes; $G$: generator; $m$:
    features per class to generate
Output: Set of synthetic features $F$
foreach $c$ in $\mathcal{C}$ do
    Generate random features as:
        $R_i \leftarrow \mathcal{N}(0,1) \quad i = 1, ..., m$
    Feed $G$ with $R$ in order to obtain synthetic
      features:
        $F_i \leftarrow G(R_i) \quad i = 1, ..., m$
end
Input: $\mathcal{P}$: exemplar set for $\mathcal{C}$; $\mathcal{D}_t$: set of new
    training data; $F$: set of synthetic features
foreach $x$ in $\mathcal{P}$ do
    Pass through the old network and store output
      $q_{x,t-1}$
end
foreach $f$ in $F$ do
    Pass through last FC of old network and store
      output $q_{f,t-1}$
    Pass through last FC of new network and store
      output $q_{f,t}$
end
$q \leftarrow q_{x,t-1} + q_{f,t-1}$
Train network with distillation loss
```

more resources and computational power and used a bigger network that can provide more accurate features.

5.3. Issues

Generate reliable information is not an easy task: a necessary condition for this to be feasible is to have stored representative knowledge to solve properly the problem. In this section, we explain which kind of problems have occurred during exploiting the generative approach.

One of the main iCaRL limitations is certainly the inevitable bias that arises between the knowledge held up to a certain moment and the information related to the new incoming classes, the latter characterized by a much higher magnitude than the former. This issue affects the herding system as much as the generation of synthetic features since both try to feed the network with data based on previous representation carried out by ResNet feature extractor. Although this effect is mitigated by the use of special losses such as *distillation loss*, it arises without any solution when the representation of previous knowledge changes through the time when the system is trained on incoming new data. Consequently, it is not possible to generate features consistent with those extracted from the system during training since our WGAN can be trained only after each batch of new classes.

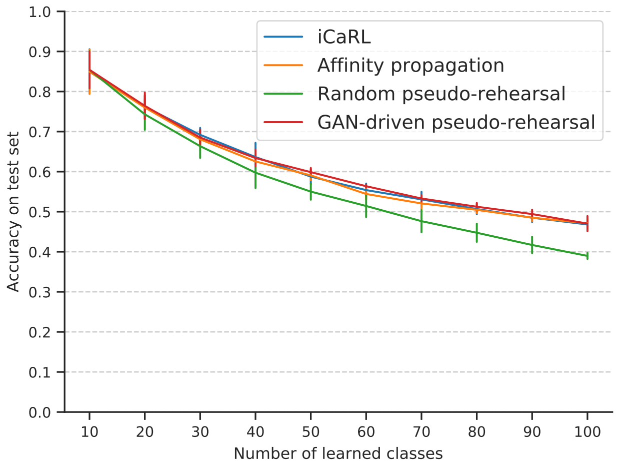


Figure 3: Incremental learning results (accuracy) per batch with settings of Table 2.

Another issue that we had to face during GAN training is the so-called *mode collapse*: a sort of overfitting phenomenon for which the generator, stuck in a local minimum, collapses and produces limited varieties of samples since it found a way to fool the associated discriminator and is not able to learn anything during the training phase. Some tricks have been proposed to overcome the problem [17], as feature matching and minibatch discrimination, but they lead to longer training time and do not guarantee optimal performances. To face this issue we tried to exploit different WGAN settings:

- Changing discriminator and generator learning rates in order to preserve the auxiliary network from unwanted behavior and unbalancing between two antagonist agents.
- Implementing a loss to force the auxiliary network to produce synthetic features that are comparable with the real ones.

Other solutions that were not exploited in this work to improve in a more formal way the system are discussed in Section 6.

5.4. Ablation study

Table 2 shows the results that we obtained with our variations compared to iCaRL.

The random pseudo-rehearsal approach gave us comparable results to iCaRL in the first batches, but lower in final ones. The reason could be that having a small subset of exemplars, especially in the last batch, the random features

Table 2: Incremental learning results (accuracy, $\mu \pm \sigma$, $n_{\text{samples}} = 3$) of our variations compared to iCaRL on *CIFAR100* dataset with an increment of 10 classes.

Model	Mean accuracy	1st batch accuracy	10th batch accuracy	Gap 1st to 10th batch
iCaRL	0.6076 ± 0.1210	0.8540 ± 0.0422	0.4681 ± 0.0004	45%
Affinity Propagation	0.6032 ± 0.1197	0.8493 ± 0.0452	0.4712 ± 0.0087	45%
Random pseudo-rehearsal	0.5653 ± 0.1426	0.8543 ± 0.0378	0.3897 ± 0.0062	54%
GAN-driven pseudo-rehearsal	0.6110 ± 0.1184	0.8540 ± 0.0375	0.4703 ± 0.0155	45%

vectors are less similar to the real ones. A possible improvement is discussed in Section 6.

6. Future work

The experiments we have presented are only the starting point for implementing iCaRL and improving its ability to perform incremental learning. Below we present some research ideas encountered during the development of this project that inspired us or that we would have liked to implement in case we had more resources or time available.

Improve SVM as classifier In Section 3.1 we proposed come variations of iCaRL that used different classifiers, eventually with different losses like the SNN loss in Section 3.2.3. Besides tuning, one of the problems of low values of accuracy using the SNN loss and SVM could be the training phase. We used the standard network in the training phase (with the SNN loss) and we changed the last fully connected layer with the SVM only when testing. A problem of this approach is that despite the SNN loss is trying to separate classes, we are training the network knowing that we will have a fully connected layer at the end. In the end, we use an SVM on a network that is not optimized for SVMs. A solution could be to use the SVM when training the network as done by Tang [19].

Improve the SNN loss The SNN loss could be improved too. Since it is sensible to the batch size (the higher the better) we may have better performances increasing it more than what we tried given our computational resources. We may also try to increase the number of epochs since the loss did not converge with 150 epochs, which is the maximum number of epochs that we tried.

Improve features generation In Section 5.3 we discussed the problems we faced with the random generation of the images (random pseudo-rehearsal). We improved the method by generating features instead of images. The main problem of this approach is that, unlike images, features must be computed at each epoch, since the feature representation changes. For this reason, while with images we could store the mean and standard deviation of all the images the first time, with features we need to compute them with the exemplars. Better results may be obtained keeping more exemplars for the generation process. In this case, generating better features having more exemplars, the performances may be higher than the iCaRL version with the same number of exemplars.

Face the so-called *semantic drift* Class-incremental learning of deep networks sequentially increases the number of classes to be classified. During training, the network has only access to data of one task at a time, where each task contains several classes. During this process, the network is subject to a sort of ”drift” effect that leads previous knowledge to be represented biased in favor of the information associated with the new classes. In particular, it would have been interesting to carry out a study on the variation of the ”semantic” representation of classes involved in incremental learning, drawing inspiration by Yu *et al.* [22].

Improve generative models Our experiments prove the need to make the generative model able to learn incrementally as much as the main system. The necessary condition to refresh the representation of previous knowledge over time is to be able to generate synthetic information without being affected by the learning process. To this end, it would be possible to:

- Exploit new losses to distill and preserve knowledge from the precursor models to the current ones.
- Use more complex architectures and more suited for this purpose.

7. Conclusion

In this work, we implemented a working iCaRL instance and exploited personal methods to improve its weaknesses. We acquired an in-depth knowledge of the incremental learning problem, exploring different methods and techniques to address it from different scenarios and propose our personal solutions to improve the exemplars selection and to deal with the case of unavailable exemplars. We understood the problem’s importance in the modern world and had the opportunity to work on the state of the art solutions in a widely researched topic.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[2] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[3] D. Fontanel, F. Cermelli, M. Mancini, S. R. Bulò, E. Ricci, and B. Caputo. Boosting deep open world recognition by clustering. *arXiv preprint arXiv:2004.13849*, 2020.

[4] N. Frosst, N. Papernot, and G. Hinton. Analyzing and improving representations with the soft nearest neighbor loss. *arXiv preprint arXiv:1902.01889*, 2019.

[5] C. He, R. Wang, S. Shan, and X. Chen. Exemplar-supported generative reproduction for class incremental learning. In *British Machine Vision Conference (BMVC)*, 2018.

[6] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[7] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019.

[8] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[9] Z. Li and D. Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[10] X. Liu, C. Wu, M. Menta, L. Herranz, B. Raducanu, A. D. Bagdanov, S. Jui, and J. van de Weijer. Generative feature replay for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 226–227, 2020.

[11] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[12] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2624–2637, 2013.

[13] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[14] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[15] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[16] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[17] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans, 2016.

[18] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay, 2017.

[19] Y. Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[20] C. Wu, L. Herranz, X. Liu, Y. Wang, J. v. d. Weijer, and B. Raducanu. Memory replay gans: Learning to generate images from new categories without forgetting. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 5966–5976, Red Hook, NY, USA, 2018. Curran Associates Inc.

[21] Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019.

[22] L. Yu, B. Twardowski, X. Liu, L. Herranz, K. Wang, Y. Cheng, S. Jui, and J. v. d. Weijer. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6982–6991, 2020.