Incremental learning in image classification: an ablation study

Benedetto Irene
PoliTo: s276200
s276200@studenti.polito.it

Panariello Michele
PoliTo: s276977
s276977@studenti.polito.it

Vitaletti Davide
PoliTo: s269462
s269462@studenti.polito.it

Abstract

Incremental learning is a learning paradigm in which a deep architecture is required to continually learn from a stream of data. In our dissertation, we perform an ablation study on the milestone algorithms of the incremental learning field. We start from a high level description of such algorithms: namely, we discuss the approaches of fine-tuning, Learning without Forgetting, and iCaRL. Our study then focuses more in-depth on the iCaRL framework, which makes use of the techinques of knowledge distillation and exemplars hearding to preserve past knowledge acquired by a deep neural network, while simultaneosly training it on new classes.

In our work, we amend several modifications to the origial iCaRL algorithm: specifically, we experiment with different combinations of distillation and classification losses and introduce new classifiers into the framework. By inspecting the behavior of the newly introduced elements, we seek to better understand how the model benefits from each of its individual components and what its possible weaknesses are.

We subsequetly propose three extensions of the origial iCaRL algorithm that attempt to tackle such issues, and we verify their effectiveness. For a fair comparison, we perform our tests on CIFAR-100, as used in the original iCaRL paper. Besting iCaRL in terms of prediction accuracy proved to be quite difficult: we discuss the reason why our proposals were not able to improve the algorithm's performance, and we illustrate other interesting insights resulting from our experiments.

1. Introduction

Human learning is naturally incremental. What is commonly conceived as "learning" is a continuous act of gathering experiences, processing incoming information and enriching knowledge accumulated in the past with new one. However, in the machine learning setting, it is often assumed that all the data a model will ever need to learn from is available from the very start of the algorithm's lifetime. While this assumption can sometimes be reasonable, some applications may require machine learning models to grow over time. More specifically, in the field of computer vision, we wish for a model to recognize new categories of images without forgetting the previously learned ones; in other words, we would like to design a machine learning algorithm that is capable of being trained on a continuous stream of data and "incrementally" learn to classify new classes of images while also maintaining the memory of classes seen in the past. We call this scenario class-incremental learning.

In [1], Rebuffi et al. proposed a list of 3 properties an algorithm should fulfill in order to qualify as class-incremental:

- it should be trainable from a stream of data in which images of different classes occur at different times;
- it should capable of classifying all the classes observed so far;
- its memory usage should be bounded, or it should at least grow very slowly with respect to the amount of classes seen by the model.

While the first two points constitute the definition of incremental learning itself, the third one rules out trivial solutions to the problem, such as continuously storing all training samples and re-training a model from scratch as soon as new classes of data are available. This last approach is commonly referred to in the literature as "joint training".

So far, joint training has constituted a performance upper bound on all proposed incremental learning methods. However, there are instances in which the joint approach is simply not feasible: for example, when past data is no longer available, or when a complete re-train of a classifier would be overly expensive in terms of computation workload. This motivates the search for incremental algorithms which are able to infer knowledge from the new data available and retain past knowledge without having to rely on the entirety of the old training samples.

All algorithms that attempt to accomplish the task of incremental learning must face the problematic phenomenon commonly referred to as *catastrophic forgetting*. With the

term *catastrophic forgetting* we mean the scenario in which, once an incremental model learns new classes, it experiences a performance drop in classifying old ones: in other words, it starts forgetting past knowledge.

In the last few years, several methods have been proposed to counteract this phenomenon: in the next sections, we present three of them that are well known in the literature and address this issue in different ways. All of said algorithms respect the three aforementioned principles of incremental learning and succeed in the task to various degrees. We focus our dissertation on the iCaRL learning framework [1], currently considered among the state-of-the-art in incremental learning and used as baseline for several recent proposals in the field [2] [3].

The dataset we employ for our experiments is CIFAR-100 (the same used in [4]), which is composed of 32 x 32 pixel images belonging to 100 classes (we further describe it in section 3). We simulate an incremental training scenario by dividing the dataset in mini-batches of 10 classes each and training our models on the resulting 10 mini-batches in an incremental fashion. We call the training on a single batch of 10 classes an "incremental training step". After each incremental training step, we test the model's performance on all the classes it has been trained on so far; that is, after the k-th incremental training step, we test on $10 * k$ classes, and compute various statistics which we subsequently illustrate and discuss. For a fair and complete analysis, we run all the proposed experiments three times, in which we randomize the distribution of labels across mini-batches. For each incremental training step, we then report mean and standard deviation of the three results. For consistency across different experiments, the three runs are always performed on the same three random seeds.
Our source code is available at
`https://github.com/irenebenedetto/MLDL-incremental-learning-project`.

2. Related work

The experiments and methods proposed in this dissertation relate to:

- The original iCaRL algorithm presented in [1], in which we try to tackle the issue of training dataset imbalance between old and new classes by proposing several techniques presented in section 8.3. We also discuss different variations over iCaRL which involve several different losses and classifiers, in the attempt to improve the overall performance of the original algorithm;

- SupportNet as described in [2], from which we take the support vector-based herding strategy and couple it with a SVM classifier to propose a variation of the original iCaRL framework.

- The *ensemble of specialists* described in [5], which we modify and adapt for the incremental learning setting; we merge the resulting model with the pre-existing iCaRL framework. We also propose three variations loosely based on the *inference with the ensemble of specialists* presented in the paper;

- The knowledge distillation procedure [5]. Our proposal of a "2-in-1" distillation model (see section 8.2) tries to harness the advantage of a distillation procedure by performing a simultaneous double distillation into a single model: one to preserve the old knowledge on previous classes, one to consolidate the knowledge acquired on novel classes.

3. The dataset and data preparation

The CIFAR-100 dataset is composed of images of 32 x 32 pixels from 100 different classes. We use 50000 images for the training set and 10000 images for the test set. We uniformly distribute classes between train and test data: that is, for each class, we have 500 training examples and 100 test examples. We divide the dataset in 10 mini-batches, each containing 10 classes. During each training phase, the network learns features of images that belong to the new mini-batch. Across different runs, classes of images are randomly shuffled across batches to ensure an unbiased evaluation of the presented methods. After the shuffle, we perform an internal re-mapping of the class labels of CIFAR-100, so that the model is always presented classes with labels incrementally growing from 0 to 99. However, we do preserve the link between the original label y_o and the re-mapped label y_m: this allows the conversion from y_m to the original, human readable form of y_o, which will be employed in several visualizations throughout this analysis.

The test dataset is incrementally updated with each training phase: at the i-th incremental training step, its size is equal to $100 * 10 * i$, and it contains 100 examples from each class the model has been trained with so far.

4. The baselines

4.1. Fine-tuning

The first baseline considered is the fine tuning approach briefly described in [6]. The deep architecture mentioned is composed of a set of shared parameters θ_s in its inner layers; the outer layers instead comprise a set of class-specific parameters for previously learned classes θ_o and for new classes θ_n. When training on a batch of new incoming classes, θ_s and θ_n are optimized for the new classes, while θ_o are kept frozen. This way, the parameters of the inner

layers and the parameters of the new fully connected layers are updated according to the knowledge acquired from the new classes, and therewith knowledge of past classes should be preserved. As outlined in section 4.4, this approach did not prove to be robust against the catastrophic forgetting phenomenon and achieved the worst performances of the three baselines.

4.2. Learning Without Forgetting

Our second baseline is the Learning Without Forgetting (LwF) method described in [6]. This is the first algorithm we encounter in our dissertation to make use of the so-called *knowledge distillation* procedure, as described in [5]. Learning Without Forgetting operates similarly to the previously illustrated fine-tuning method, but uses knowledge distillation to prevent catastrophic forgetting by trying to preserve the responses of previously trained parameters θ_o to new training samples. Simultaneously, a classification loss is applied to the new parameters θ_n, to instruct them in recognizing new classes. Therefore, the main difference with the previous approach is that in Learning Without Forgetting the parameters associated to past classes are not frozen; rather, they are updated according to the output produced by a past configuration of the network. Even though the images for the new classes may provide a poor sampling of the past classes, in practice this technique proves to be quite effective in preserving old knowledge. The losses proposed in [6] are:

1) Cross entropy loss for the classification task:

$$L_{new}(y_n, \hat{y}_n) = -y_n * \log \hat{y}_n$$

Where the $\hat{y}_n$ is the softmax of the output and y_n is the one-hot encoding of the ground truth label vector.

2) Knowledge distillation loss, suggested by Hinton et al. [5].

$$L_{old}(y_o, \hat{y}_o) = -H(y'_o, \hat{y}'_o) = -\sum_{i=1}^{t} y'^{(i)}_o \log \hat{y}'^{(i)}_o$$

Where t is the total number of old labels, $\hat{y}'_o$ and y'_o are respectively the temperature softmax outputs of current and past versions of the network corresponding to old classes.

The classification loss encourages the model to perform correct predictions on new classes, while the distillation loss attempts to preserve the outputs of the network's previous configuration. This way, the network is trained to learn new images and, at the same time, to emulate the behavior of the past version of the model, which is supposed to correctly classify the old labels.

4.3. iCaRL

Finally, we introduce iCaRL, the state of art for class incremental learning algorithms. Compared to the previous two techniques, iCaRL introduces two novel aspects: the use of exemplars of past classes and a nearest mean of exemplars classifier (NME) for classification. For each learned class, the model stores some items of that class which are used in subsequent training steps and in combination with the NME at classification time. At any given time, no more than K exemplars are stored. If the current number of classes learned by the model is N, then the number of stored exemplars for each class (which we call "exemplar set") is $m = K/N$.

Once the model's feature representation is updated on new training classes, two routines are called to manage the exemplar sets: one selects the exemplars for new classes (the so-called *herding* procedure) and one reduces the size of the exemplar sets of previous classes to fit within the K upper bound. iCaRL proposes a herding which selects the m items of a class which, when summed, best approximate the overall class mean in feature space. When an exemplar set must be reduced to size $m' < m$, the m' elements that best approximate the mean are again selected amongst the already present exemplars. Because the exemplars are initially stored in order of how relevant they are in reconstructing the original mean, it is sufficient to take the first m' elements of the exemplar set.

4.3.1 The Nearest Mean of Exemplars classifier

iCaRL's classifier of choice is the Nearest Mean of Exemplars classifier (NME), which makes use of the deep representation of the exemplars stored in the model. To predict a label y^* for a query image x, iCaRL computes the means of all the exemplar sets as $\mu_i = \frac{1}{|P_i|} \sum_{p \in P_i} \phi(p)$ where P_i is the i-th exemplar and $\phi(p)$ is the feature representation of the i-th example p of P_i. To assign y^*, we compute the feature mapping $\phi(x)$ of the query image and we find the exemplar mean that is closest to that value:

$$y^* = \underset{i}{\operatorname{argmin}} ||\phi(x) - \mu_i||$$

Notice that what we call *nearest class mean* throughout this dissertation is not the actual class mean of all samples of a class, but rather the mean of the class according to the current exemplar set available for it.

4.3.2 Classification and distillation loss

During training, iCaRL also makes use of a classification loss in conjunction with a distillation loss to preserve previously acquired knowledge. To learn the feature representation employed by the NME classifier at test time, iCaRL uses the activations of a fully connected layer passed through a sigmoid function. Output neurons associated to old classes are encouraged to emulate the behavior of their former version, prior to the incremental learning step; new

neurons are trained to correctly classify new images. The overall loss function is:

$$l(\Theta) = -\sum_{(x_i,y_i)\in D}[\sum_{y=s}^{t}[\delta_{y=y_i}\log g_y(x_i) + \delta_{y\neq y_i}\log(1-g_y(x_i))] + \sum_{y=1}^{s-1}[q_i^y \log g_y(x_i) + (1-q_i^y)\log(1-g_y(x_i))]] \quad (1)$$

Where D is the augmented training set composed by the currently available training examples $(X^s \dots X^t)$ together with the stored exemplars $(P_1 \dots P_{s-1})$; x_i is the i-th image that belongs to D; $g_y(x_i)$ is the network output given x_i for the y class; q_i^y is the output of the old copy of the model given x_i for the y class; y_i is the ground-truth class label of x_i ; $\delta_{y=y_i}$ is the Kronecker delta. When a new batch of classes arrives, iCaRL constructs a training set composed of the current training data and the stored exemplars.

Retaining exemplars and mixing them with the incoming data allows the network to review and consolidate past knowledge. While the binary cross entropy loss used in iCaRL achieves remarkable performances, it is not the only possible choice: a wide part of our dissertation will be dedicated to experimenting with different losses and analyzing how they affect the model's effectiveness and behavior in general.

4.4. Results

We now compare the classification performance across each incremental training step of the following models:

- Finetuning;
- Learning Without Forgetting;
- iCaRL with Nearest Mean of Exemplars;
- Hybrid1, a version of iCaRL that performs the classification with the outputs of the fully connected layers.

The hyperparameters used in training are the ones provided in [1]. Figure 1 shows the test accuracy across incremental training steps using the four different approaches listed above.

The fine-tuning approach achieves the worst results, as it is not able to prevent the catastrophic forgetting phenomenon. The behaviour of this algorithm is displayed in the confusion matrix in fig. 2: fine-tuning is essentially unable to remember past classes and only outputs predictions relative to the last batch of learned classes. fig. 3 and fig. 4 represent the confusion matrices obtained using the Hybrid1 and iCaRL methods.

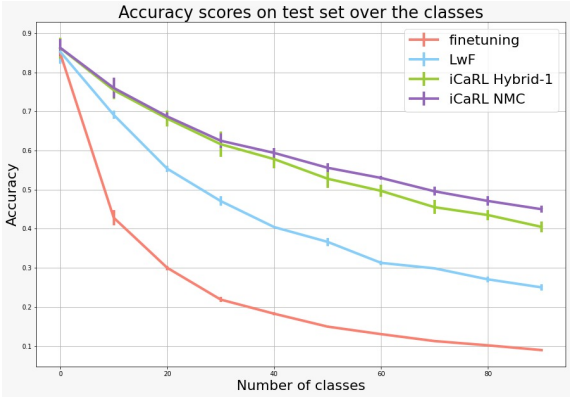

Figure 1. Baselines using iCaRL, Fine-tuning, Hybrid1, FC, LwF

The notable improvement achieved with the Learning without Forgetting algorithm (+16% on the last batch of classes) proves the crucial role of the distillation process in containing catastrophic forgetting. Hybrid1 has essentially the same configuration as LwF, with the exception of the use of exemplars for reviewing past knowledge. The introduction of this new element in the framework further mitigates catastrophic forgetting and results in a $+15\%$ accuracy on the last batch. Given the importance of exemplars, we briefly discuss the effect of different herding strategies on the overall model's performance in section 5. The introduction of the NME completes the iCaRL setup and provides a small yet noticeable performance improvement. This shows that the use of different classifiers on top of the architecture can sensibly influence the model's performance: we shall more widely discuss this topic in section 7. Table 1 summarizes the results illustrated in this section.

5. Herding

As a preliminary approach to our ablation study on iCaRL, we seek to better understand how different herding strategies affect the general performances of the framework. During the implementation of the baseline, different methods were tested to see if any useful knowledge could be obtained.

Random Herding. The simplest (and actually one of the most effective) strategy is sampling exemplars at random: for each class and for each training step, a fixed number of images of a certain class is randomly drawn from the training set to form the exemplar set of that class. A random choice exploits the natural variability of the data, which achieves good performances when combined with augmentation.

iCaRL Herding. The most natural choice for herding is the one suggested in [1]: to choose the images that, when summed, minimize the distance from the class mean. The mean is computed on all data coming from a class at training

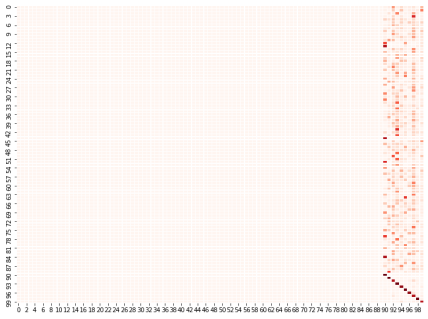

Figure 2.

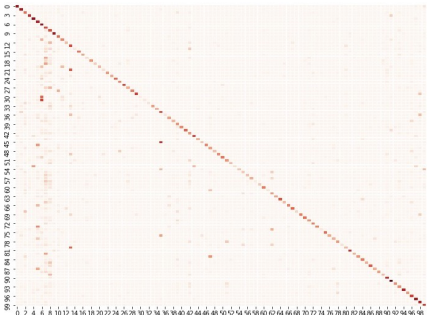

Figure 3.

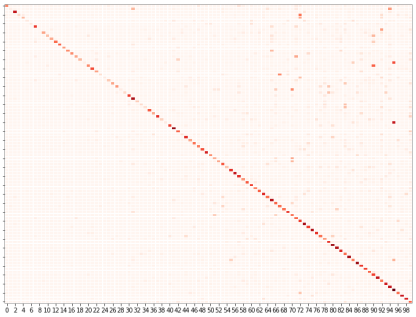

Figure 4.

Confusion matrices using Fine-Tuning (left), Hybrid1 (center), iCaRL (right).

Method	10	20	30	40	50	60	70	80	90	100
iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
Hybrid1	0.862	0.755	0.682	0.616	0.578	0.528	0.497	0.455	0.435	0.405
LwF	0.853	0.691	0.554	0.471	0.404	0.366	0.313	0.299	0.270	0.250
Fine-Tuning	0.849	0.428	0.300	0.219	0.183	0.150	0.130	0.113	0.102	0.090

Table 1. Accuracies over the number of classes using iCaRL, Hybrid1, LwF, Fine-Tuning

time. This is of course a very effective strategy, and it is the one we will use throughout this dissertation, unless where specified otherwise.

Duplicate exemplars. One option to take into account, which is not clearly specified in [1], is whether to allow duplicate images within an exemplar set or not. According to our experiments, using only distinct images as exemplars produces slightly better results in terms of classification accuracy. Indeed, selecting duplicate exemplars would be like using up the limited amount of storage at our disposal for useless information. Therefore, in our analysis, we always adopt the approach of not selecting duplicate images as exemplars.

Class means computation. Aside from herding itself, other possible variations were considered concerning exemplars. In particular, one possible alternative regards the computation of the means when using the nearest class mean classifier. The mean for each class can be obtained:

- by averaging the exemplar set of that class using the current deep feature extractor. This requires updating the class means when the feature extractor is updated, as is the approach proposed in [1];
- by computing the class mean on all training data from a specific class at training time, and storing it for later usage by the Nearest Class Mean (NCM) classifier. This has the advantage of saving computations, but relies on the assumption that the feature representation of the model will not vary too much overtime (each mean is computed once with the current version of the feature extractor).

Our experiments showed that the two methods do not differ critically in terms of classification accuracy; however, the latter seems to be slightly more performant on CIFAR-100 (though by no more than $1-2\%$). Therefore, we henceforth adopt it as our mean computation strategy. For demonstration purposes, we graphically show the performance variation of three of the possible herding strategies described so far (fig. 5):

- Setup 1: herds according to original iCaRL; exemplars are augmented; class mean is computed with all traning data;
- Setup 2: herds randomly; exemplars are augmented; class mean is computed with all traning data;
- Setup 3: herds according to original iCaRL; exemplars are not augmented; class mean is computed on exemplars only.

The test results show that there is no significant performance difference between random herding and the original iCaRL's herding; however, we do see a slight accuracy drop when computing class means only on exemplars. It is reasonable that such drop becomes evident only in later stager of training, when the model has fewer exemplars at its disposal: this of course means having less information available to infer class means, which results in a less accurate computation. In section 7.2, we also propose a further herding strategy based on selecting support vectors of a SVM as exemplars.

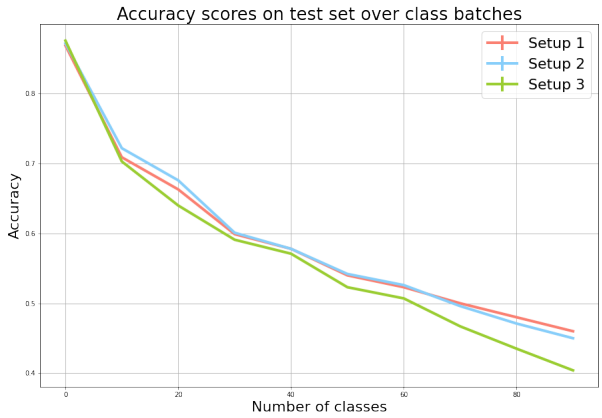

Figure 5. Accuracy comparison of different herding strategies

6. Ablation study on losses

6.1. L_p distillation loss

6.1.1 Theoretical description

To verify the effectiveness of the binary cross entropy loss within iCaRL's framework, we test a different combination of classification and distillation losses. More specifically, we employ as distillation loss the L_p-norm of the difference between feature mappings of old and new network for the same sample, raised to power p. In our experiments, $p = 1$ and $p = 2$ were used (the case for $p = 2$ is essentially a mean squared error). The goal is to align the feature mappings of the old network, which we assume to be correctly trained, with the feature mappings generated by the new network during the new phase of the incremental training. During distillation, all feature maps are normalized. Of course, not only does the new network have the task of imitating the old one: it should correctly perform classification on the new labels. Thus, a classification term is needed within the overall loss. Two possible losses were tested for this part:

- iCaRL's original binary cross entropy loss;
- A cross entropy loss.

The latter proved to be slightly more effective than the former (in terms of accuracy) when used in conjunction with a L1 and L2 classification loss. Because our main goal is to test the effectiveness of the distillation term, we keep the cross entropy as classification loss for our experiments. The overall loss therefore becomes:

$$loss(x) = -log(p_{class}) + ||\bar{f}_{old}(x) - \bar{f}_{new}(x)||_p^p$$

Where $\bar{f}_{old}(x)$ and $\bar{f}_{new}(x)$ are the normalized feature maps of the old and new model, while p_{class} is the probability predicted by the model about the current class sample. We choose to leave the power p into the norm in order to further vary the penalization generated when old and new mappings are placed farther away from each other.

6.1.2 Results

We compare the performance of each loss in two ways: firstly, by inspecting the overall accuracies across each training step; secondly, by separately analyzing the accuracy for new classes and past classes at each training step. The accuracy on old classes is computed as:

$$Acc_{old} = \frac{CF_{old}}{N_{old}}$$

where CF_{old} is the number of correctly classified elements from old classes, while N_{old} is the number of elements from old classes. Accuracy on new classes is computed analogously.

Using L2 loss yields the same results as iCaRL in early training stages; however, as time progresses, performances negatively diverge from the baseline. This seems to be due to an overfit towards new classes (see confusion matrix in fig. 6). L1 loss exhibits the opposite behavior: accuracies are lower than its counterpart during early stages of training, and higher towards the last batches of classes. Overall, the memory of the model is much more uniform with the L1 loss (see confusion matrix in fig. 7). The general trend of the two losses is illustrated in fig. 8.

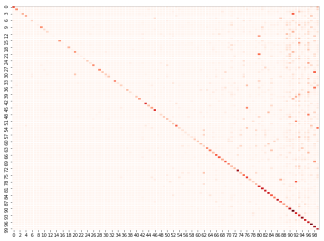

Figure 6.

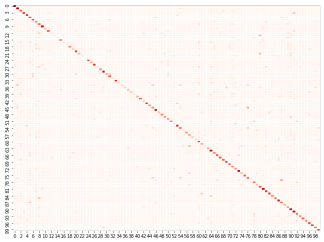

Figure 7.

Confusion matrices of iCaRL using L2 (left) and L1 (right)

We now attempt to understand the reason behind this difference in performance. It is possible to get some insight by inspecting the contribution of L1 and L2 loss to the overall loss (fig. 9): numerically, L2 is much lower than L1 at all times. This is to be expected, since we are operating with normalized features: distances will likely be below 1, and the square will decrease their value even more. This is particularly relevant since we are using the L_p losses as distillation loss between feature mappings of old and new network, which will supposedly be similar. L1 loss does not suffer from this problem, since it only considers the absolute value of distances, and its numerical value is therefore higher. As a result, the effect of the classification loss

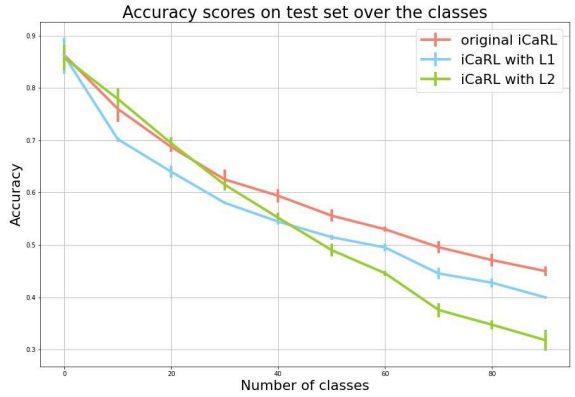

Figure 8. Accuracy across batches of classes of iCaRL with L1 and L2 distillation loss and original iCaRL

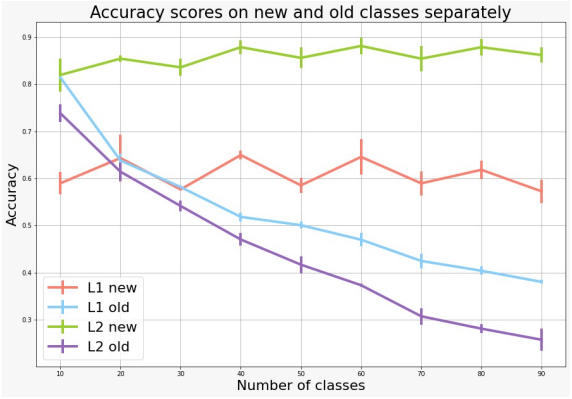

Figure 10. Accuracy on old and new classes of iCaRL with L1 and L2 distillation losses

with an L2 distillation is more pronounced: this explains the overfit towards new classes, and the fact that L2 loss is more performant with fewer classes. Conversely, L1 is much more relevant to the overall loss: consequently, the distillation process is more effective. The same conclusions can be drawn by inspecting the accuracy trends (fig. 10) on old and new classes: L2 loss always achieves high results on new classes, but is much worse than L1 at correctly classifying old classes. L1 loss behaves in the opposite way, being uniformly weaker with new classes, but more accurate with old ones (because of the more effective distillation).

Figure 9. Numerical comparison between L1 and L2 loss

6.2. Temperature softmax distillation

6.2.1 Theoretical description

We attempt to employ a different form of distillation loss based on the so-called *temperature softmax*, first proposed by Hinton et al. [5] in the context of knowledge distillation from large to small models. Given a generic input vector z, we define the temperature softmax at temperature T of $z_i \in z$:

$$q_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})}$$

When applying a normal softmax function to a vector z element-wise, we obtain a probability vector q_s whose probabilities mirror the proportions of the values found in z. A temperature softmax accomplishes a similar operation, and results in a probability vector whose values are proportional to q_s, but more flattened and uniformly distributed. In other words, a temperature softmax tends to amplify lower probabilities more than a normal softmax.

If vector z are the activations of the last layer of a neural network, a temperature softmax can be used to emphasise the "hidden knowledge" acquired by the model by highlighting prediction probabilities that would have been flattened otherwise. This retains information not only about which class the network would have predicted for a certain input, but also which classes are closest to the predicted one in terms of probability. We therefore attempt to use temperature softmax as a distillation term instead of the binary cross entropy as follows. A copy of the old network is fed with a training sample x and we compute its softmax at a given temperature T; we then pass x through the current version of the network and also compute the softmax of the resulting activations at the same temperature T. A normal cross entropy is then computed using the old network's softmax as target:

$$L_{dist}(x) = -q_{old} \log(q_{new})$$

Where q_{old} and q_{new} are the temperature softmax at some temperature T of the old and new network's outputs respectively. The classification loss applied, in this case, is a standard cross entropy loss which uses as target the one-hot encoding of the ground truth labels y_{oh}:

$$L_{class}(x) = -y_{oh} \log(s_{new})$$

Where s_{new} is the softmax of the current network's output (only considering neurons associated to old labels), this time at temperature 1. The total loss for a sample x is

$$L_{tot}(x) = w_d L_{dist}(x) + w_c L_{class}(x)$$

Where w_d, w_c are weights to fine-tune the loss terms. Notice that, with this configuration, the classification loss also takes into account labels coming from past exemplars.

Temperature T, w_d, w_c are hyperparameters: we found the best working configuration to be $T = 2$, $w_d = 1.5$, $w_c = 0.5$.

6.2.2 Results

The temperature distillation loss is remarkably effective within iCaRL framework: at the right temperature, performances are overall very close to those of iCaRL's original configuration (fig. 11). The temperature distillation loss proves to be suitable in an incremental learning setting: this does not come as a surprise, as the loss we employed in this experiment is essentially a slightly modified version of the one originally used in Learning Without Forgetting [6].

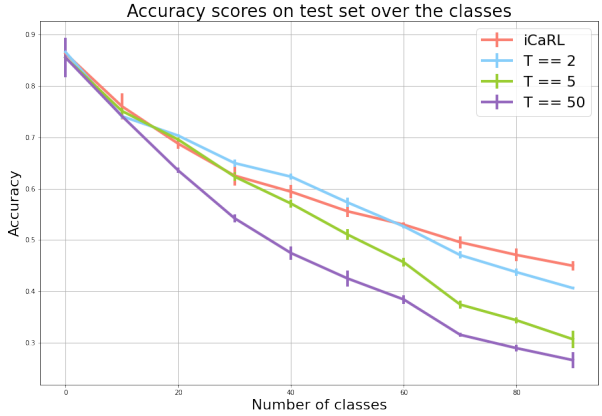

Figure 11. Classification accuracy with temperature softmax distillation

This is a very important insight on the relevance of exemplars within the iCaRL's framework: when using a loss similar to LwF with added exemplars, performances are almost equal to those of the original iCaRL's. It is evident that the exemplars play a crucial role in the task of incremental learning, regardless of the loss: they allow for a supervised review of past classes that would otherwise be remembered only by means of "unsupervised" distillation. Still, the temperature softmax setup seems to be slightly more unbalanced towards new classes in the long run (fig. 12) with respect to iCaRL (fig. 4); thus, the absolute best configuration remains the distillation based on binary cross entropy.

To get a further sense of what is happening, we confront accuracies separately for old and new classes. We compute

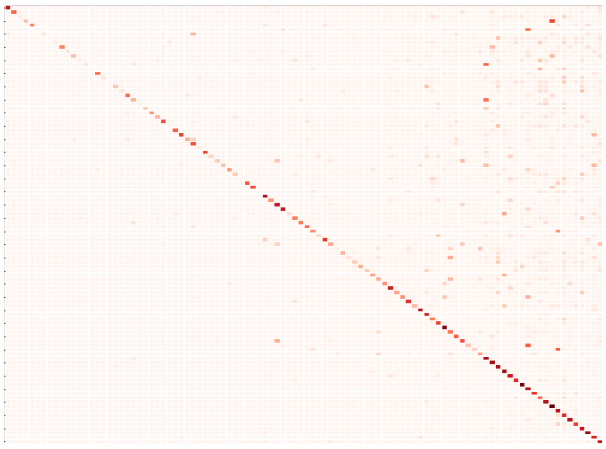

Figure 12. Confusion matrix at 100 classes with temperature softmax distillation (T=2)

the accuracy on new classes as the number of correctly classified samples from the last batch of training classes over the total number of samples from those classes. The accuracy on old classes is computed analogously. We discover that, with respect to the binary cross entropy, the distillation effect of the temperature softmax is stronger during early stages of training and weaker on the last batches (fig. 14). A more refined (possibly adaptive) weighting of the distillation term might help counteracting this phenomenon.

As regards the correct temperature for distillation, it appears that excessively high temperatures tend to degrade the distillation effect even more quickly (fig. 15); we settle on $T = 2$, which is in line with how the temperature softmax is used in [6] and [5].

Fig. 13 highlights the effect of different temperatures on the probability returned by the softmax function: the higher is the temperature, the "flatter" probabilities are. Therefore, with an excessive temperature, this loss levels off the old network's probabilities, which consequently transmit less information in distillation process. This is particularly significative in the long run, when probabilities tend to distribute uniformly across a wider range of classes. The results of this sections are summarized in table 2.

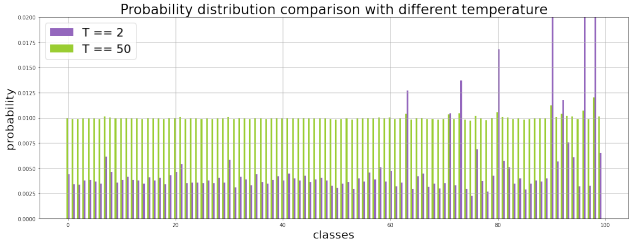

Figure 13. Probability distribution of the same input image after a softmax at different temperatures

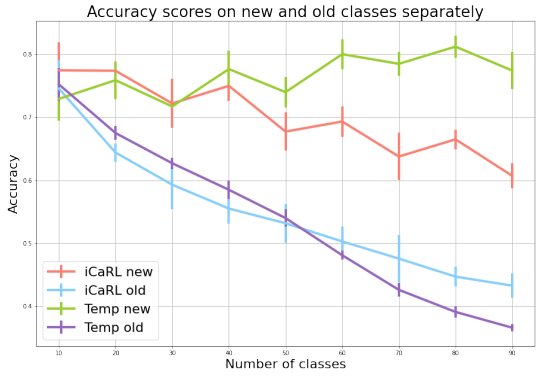

Figure 14.

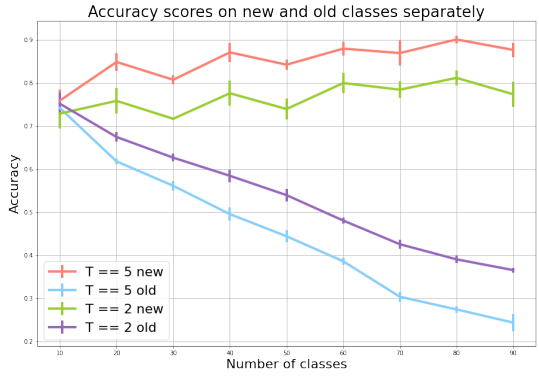

Figure 15.

Separate accuracy on old and new classes of iCaRL and temperature softmax (right); separate accuracy on old and new classes with softmax distillation at different temperatures (left)

Method	10	20	30	40	50	60	70	80	90	100
iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
T=2	0.866	0.741	0.703	0.650	0.623	0.573	0.526	0.471	0.438	0.407
T=5	0.855	0.751	0.695	0.623	0.571	0.511	0.457	0.375	0.344	0.307
T=50	0.855	0.741	0.635	0.542	0.475	0.425	0.384	0.316	0.290	0.266

Table 2. Accuracies over the number of classes using iCaRL and temperature softmax method with differnt T values

6.3. Soft Nearest Neighbor Loss

As previously discussed in section 4.3, iCaRL performs classification with a nearest mean of exemplars, which assigns labels to an input image according to which class has the closest mean to it in the feature space. Therefore, feature representation of images becomes particularly relevant for the classification task: intuitively, the more exemplar sets are clustered in the feature space, the more effective the nearest mean classifier will be. To verify whether this assumption is correct or not, we introduce an additional component to iCaRL's loss: the Soft Nearest Neighbor loss proposed in [7]. We first describe the loss from a theoretical standpoint, and subsequently analyze the obtained results.

6.3.1 Theoretical description

We adopt the Soft Nearest Neighbor (SNN) loss in order to attempt to more tightly amass the clusters formed by exemplars in the feature space. This result is achieved by taking into account their relative position as training progresses, and computing a loss based on that information. In the words of the authors of [7], the Soft Nearest Neighbor loss is designed to reduce entanglement between manifolds of classes in high dimensional space. In general, this means that any sample of a given class is closer to samples of that same class than to samples of different classes.

In practice, to obtain the loss, we perform the following computations. Given a point x_i in the feature space, we compute the distance between:

- x_i and all other points belonging to its same class;
- x_i and all other data points.

All these distances are then squared, divided by a temperature T and passed through an exponential with a negative sign. This way, it is possible to construct a softmax with the distances of x_i from points of the same class with respect to the distances of x_i from all other points. This quantity can be interpreted as follows: let x_i be a data point of label y_i of a training batch. We randomly sample a point from the same batch with a probability proportional to its distance from x_i. The SNN loss can be seen as the probability of sampling a neighboring point of the same class of x_i. We take the negative log of such probability and compute this contribution for all points x_i of a batch. The average of such value across the batch is the Soft Nearest Neighbor loss:

$$l_{sn}(x, y, T) = -\frac{1}{b} \sum_{i \in 1..b} log(\frac{\sum_{j \in 1..b, j \neq i, y_i = y_j} e^{-\frac{||x_i - x_j||^2}{T}}}{\sum_{k \in 1..b, k \neq i} e^{-\frac{||x_i - x_k||^2}{T}}})$$

As previously mentioned, this loss introduces the temperature T as new hyperparameter. This value controls the

relative importance given to the distances between pairs of images: the higher the temperature , the lower all distances between points become; consequently, the distances between widely separated points have larger influence on the loss. In other words, the loss becomes stricter, since widely separated points influence it significantly.

As the authors point out, minimizing the Soft Nearest Neighbor loss does not necessarily mean collapsing clusters to a single point: a minimization of this loss can be achieved by creating a wide enough separation between class manifolds.

As previously stated, the nearest class mean classifier assigns a label according to the closeness of an input image with a mean of a class in the feature space. By using Soft Nearest Neighbor loss, the model is encouraged to generate close feature mappings for images that belong to the same class. Our intent is to aid the nearest class mean classifier in the decision by obtaining more compact and widely distributed exemplar sets. When using the Soft Nearest Neighbor loss, we simply add it as an additional term to the already present loss before computing gradients. The loss is computed batchwise, with no distinction at training time between past exemplars and current training images.

6.3.2 Results

In this analysis we experiment with this loss in three different models:

- iCaRL model;
- iCaRL with a KNN classifier;
- iCaRL with a SVM classifier.

The last setting will be separately discussed in section 7.2. In our experiments, we use temperature $T = 2$. We first focus on comparing the performances of standard iCaRL as described in [1] against iCaRL with an added SNN loss. We first visually compare the difference between feature mappings by inspecting the t-SNE representation of the final 100 exemplars sets after training iCaRL normally (fig. 17) and with the Soft Nearest Neighbor loss (fig. 16).

The separation among the highlighted classes is particularly evident in the configuration in which the Soft Nearest Neighbor loss is added. To get numerical confirmation of that, we consider each set of exemplars in the feature space as a cluster and we compute the silhouette score for the overall dataset. The silhouette score is commonly employed as a measure of effectiveness of a clustering function: it indicates how close objects are to members of their own clusters, and how far from elements of other clusters, as a number in the range $[-1, 1]$.

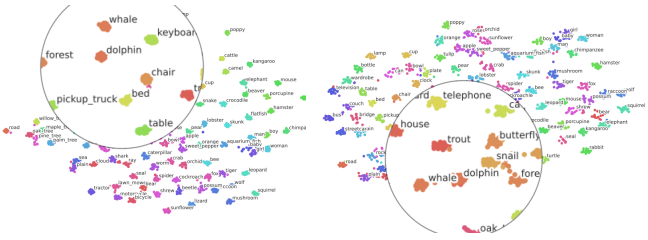

Figure 16. Figure 17.

Comparison of t-SNE of 100 exemplars set with and without Soft Nearest Neighbor loss (left and right representations, respectively). Exemplar sets of same classes are highlighted.

Numbers confirm our thesis: the silhouette score at the end of the training process is 0.21 for iCaRL + Soft Nearest Neighbor and about 0.16 for plain iCaRL.

However, contrary to expectations, accuracy score does not noticeably improve despite the increased separation between classes. Different temperatures do seem to make major differences, unless for very high values (such as $T = 10$): however, in this case, changes in performance are only negative (fig. 18)

A possible explanation of this fact might be that the number of exemplar sets is particularly small, especially in the last training step, and the separation among old classes is computed among a limited number of exemplars. In other words, having so few exemplars with respect to the new classes might make the contribution of this loss too small. Our tests have shown that, indeed, the Soft Nearest Neighbor loss starts off at about 0.1 at epoch 1 and does not diminish to less than 0.08-0.07 at epoch 70. It seems that, with this setup, there simply is no room for much improvement: the model cannot find a way to cluster the exemplar sets more tightly than this. That said, it could be plausible that by rebalancing the dataset between exemplars and new classes at training time, the contribution of this loss may be more relevant to the training process. We will further discuss this possibility in section 8.3.

7. Ablation study on classifiers

7.1. Cosine normalization layer

7.1.1 Theoretical description

One of the main problems of the incremental learning setting is the imbalance between new training samples and exemplars of past classes during training. Such issue can be tackled in several ways: one possibility would be trying to "resample" past exemplars by generating them with different techniques, or attempting to estimate their distribution and sampling from it (we explore both these possibilities on

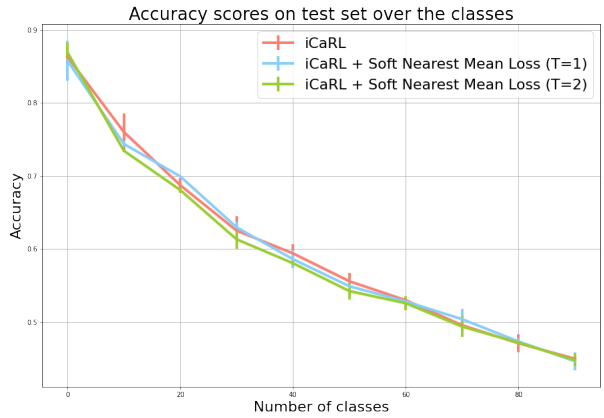

Figure 18. Accuracy comparison of iCaRL with and without SSN loss

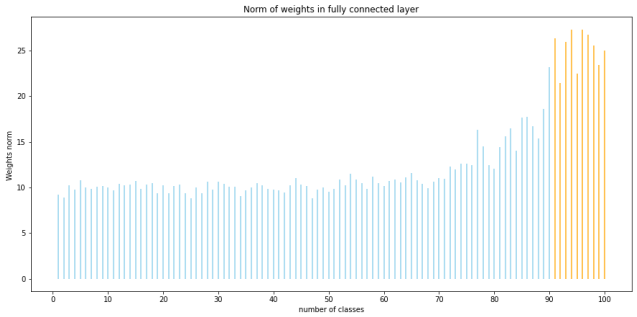

Figure 19. Norms of weights in the last fully connected layer after 10 incremental training steps

section 8.3). Hou et al. [3] instead propose a method based on the normalization of the weights of the last fully connected layer. This approach is motivated by the fact that, according to the paper, the model will often assign weight vectors of higher norms to the most recent classes in the last fully connected layers; similarly, biases associated to new output neurons will tend to be higher. We verify the former claim by inspecting the weights of the last fully connected layer of the network after running a full 10-batch training in the standard iCaRL setting. More specifically, we plot the L_2-norm of the vector of weights associated with each output neuron. Indeed, the last 10 classes seem to consistently have vectors of higher norms. Fig. 19 displays the weight norms after 10 steps of incremental training.

We now attempt to verify if a rebalancing of the network weights is beneficial for the incremental learning of the model. To do so, we replace the normal fully connected layers of the network with a different classifier: the cosine normalization layer proposed in [3]. This classifier computes the probability $p_i(x)$ that a sample x belongs to class i using a modified softmax:

$$p_i(x) = \frac{exp(\eta \langle \bar{\theta}_i, \bar{f}(x) \rangle)}{\sum_j exp(\eta \langle \bar{\theta}_j, \bar{f}(x) \rangle)}$$

Where $f(x)$ is feature mapping of sample x, θ_i are the weights of the last fully connected layer associated with class i, and $\bar{f}(x)$ and $\bar{\theta}_i$ are their normalized counterparts. η is a learnable scalar which controls the magnitude of the scalar product between $\bar{f}(x)$ and $\bar{\theta}_i$, which is constrained in [-1,1] because of the normalization. Notice that, with this classifier, we normalized both the weights of the fully connected layers and the feature mappings of the convolutional layers. This method does not take biases into account: this way, we simultaneously tackle both problems stemming from imbalanced training.

7.1.2 The Less Forget Constraint loss

We combine the action of this new classification layer with:

- a normal cross entropy loss (which employs the probability described above) as classification loss;
- the *less forget constraint* proposed in [3] as distillation loss, specifically for this classifier;
- the *inter-class separation* loss, again proposed in [3].

Overall, we refer to this setup as *Less forget constraint loss*. The less-forget constraint encourages the new network to produce mappings similar to the old network by exploiting the fact that a cosine similarity between two normalized vectors is at most 1:

$$L^G_{dis}(x) = 1 - \langle \bar{f}^*(x), \bar{f}(x) \rangle$$

Where $\bar{f}^*$ is the feature mapping of the old network. Inter-class separation loss encourages separation between clusters of feature-mapped vectors of different classes. Specifically, it attempts to separate the mapping of new incoming classes from the already established mappings of the old classes by at least a margin m. For each old sample x_o and its label y_o, we take the K neurons belonging to new classes that activate the most when receiving x_o. We encourage the activation value of each of these K neurons to differ at least m from the activation of the proper neuron associated to y_o. These activations are of course computed by using the cosine-normalization layer, but without taking into account the η scalar. This results in the following loss:

$$L_{mr}(x) = \sum_{k=1}^{K} \max(m - \langle \bar{\theta}(x), \bar{f}(x) \rangle + \langle \bar{\theta}^k, \bar{f}(x) \rangle, 0)$$

K and m are hyperparameters. In our experiments, we use the values suggested by [3]: $K = 2$ and $m = 0.5$. The overall loss of a batch of size N is therefore computed as:

$$L = \frac{1}{|N|} \sum_{x \in N} (L_{ce}(x)) + \lambda L^G_{dis} x)) + \frac{1}{|N_o|} \sum_{x \in N_o} (L_{mr}(x))$$

Where N_o is the number of old exemplars in the batch, and λ is a weight factor for the distillation loss. [3] proposes to adaptively change this weight according to the current number of old and new classes:

$$\lambda = \lambda_{base} \sqrt{|C_n|/|C_o|}$$

Where λ_{base} is an hyperparameter of the model, $|C_n|$ and $|C_0|$ are the number of new and old classes, respectively. In our experiments, we found the best working value to be $\lambda_{base} = 2.5$. Alongside this combination of loss and classifier, the normal iCaRL herding strategy is employed. For a more detailed ablation study, we simultaneously test the classification performances of the cosine normalization layer and of the standard nearest mean classifier used in the original iCaRL architecture.

7.1.3 Results

We evaluate the performance of the cosine normalization layer by comparing the accuracy returned after each training step with the results obtained with two different configurations of iCaRL:

- Nearest Mean Classifier + binary cross entropy loss (i.e. the original iCaRL setup, as described in [1]);
- Nearest Mean Classifier + Less Forget Constraint loss.

In terms of raw accuracy, the cosine normalization layer seems to be able to stand up to iCaRL only during early training stages; on later steps, performances tend to drop. The attempt to reduce bias toward new classes by employing a combination of a custom classifier plus an ad-hoc loss does not accomplish the task, and instead ends up accentuating the overfit on the new incoming classes. This is made particularly evident by the confusion matrix after the last stage of training (see fig. 4 and fig. 20).

The main issue with this setup seems to be the Less forget constraint loss: this is suggested by the fact that performances drastically drop even by using such loss alongside iCaRL's classifier of choice, the NMC.

We now try to get a deeper insight of the reason behind the unbalance between old and new classes and the consequent worsening of the classification results. We compute the feature mappings of the exemplar sets after the training at various training steps and visually inspect them with t-SNE. While old classes are somewhat homogeneously distributed and well separated, newer classes always seem to be placed within a single, confused cluster (fig. 22). Such

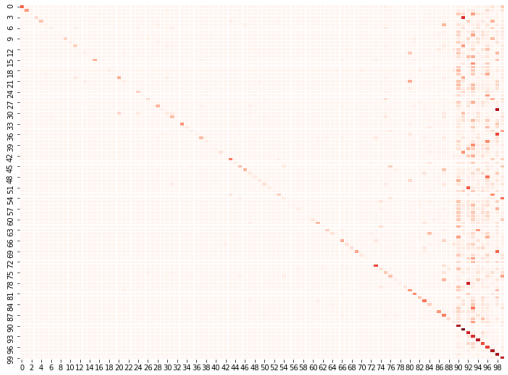

Figure 20. Confusion matrix of the Cosine Normalization Layer after 10 incremental training steps

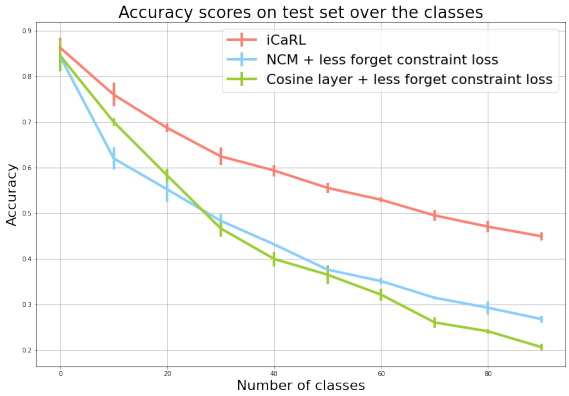

Figure 21. Accuracies using the Cosine Normalization Layer

cluster subsequently dissolves as newer classes are shown to the model. Because this phenomenon only happens to new classes, we hypothesize that it might be caused by an excessive contribution of the inter-class margin loss, whose task is to push feature mappings of new classes away from the mappings of old classes. This may have the effect of amassing all new classes in a single heap, which dissolves overtime as the classes are not affected by the inter-class separation anymore. We therefore perform another round of training with the same setting as before, except for the inter-class separation loss, which we remove completely. With this change, new incoming classes do not clump in a single area anymore (fig. 23); however, the overall separation between all classes seems to diminish, and classification performances do not noticeably improve. The accuracies resulting from this training setup are illustrated in fig. 21, and summarized in table 3 along with all other results from this section.

Method	10	20	30	40	50	60	70	80	90	100
iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
NMC + less forget constraint loss	0.843	0.620	0.553	0.484	0.432	0.377	0.352	0.315	0.293	0.269
Cosine layer + less forget constraint loss	0.845	0.700	0.583	0.467	0.400	0.366	0.322	0.261	0.242	0.207
NMC + lfc loss, no interclass	0.820	0.556	0.520	0.459	0.445	0.400	0.361	0.292	0.238	0.206
Cosine layer + lfc loss, no interclass	0.837	0.653	0.539	0.443	0.393	0.351	0.332	0.281	0.256	0.226

Table 3. Summary of accuracy scores of iCaRL with Cosine Normalization Layer variations

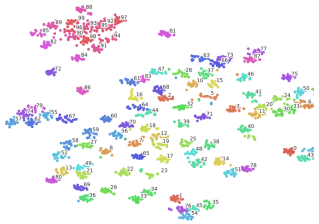

Figure 22.

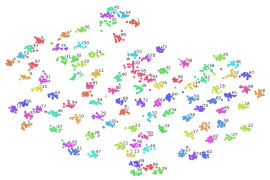

Figure 23.

t-SNE of 100 classes with and without inter-margin separation loss (left and right representation, respectively).

7.1.4 Conclusion

With the use of the cosine normalization layer, we attempted to see if a normalization of feature mappings and of the weights in the last fully connected layers could tackle the imbalance between old and new classes during the incremental training procedure. In doing this, we adopted a configuration of losses proposed in [3], specifically crafted to operate on normalized feature vectors and fully connected weights. Not only did it not achieve the goal, but it ended up sensibly worsening the performances of the classification task itself. We conclude that the cosine normalization layer is not a worth alternative to a nearest class mean classifier, which achieved superior results coupled with either the *less forget constraint* loss and the original iCaRL's loss.

7.2. SVM-iCaRL

Support Vector Machines (SVM) are supervised learning models that perform classification by constructing splitting hyperplanes that separate data points of different classes. The algorithm attempts to maximize the margin of the hyperplane: that is, the distance between the hyperplane and the closest points of each class. If the hyperplane we seek is of the form $\langle w, x\rangle + b = 0$, and m is the number of data points, the maximum margin problem can be formulated as:

$$\underset{(w,b):||w||=1}{\operatorname{argmax}} \min_{i\in[m]} |\langle w, x_i\rangle + b|$$

s.t. $y_i(\langle(w, x_i\rangle + b) > 0$.

Or, equivalently:

$$\underset{(w,b)}{\operatorname{argmin}} \frac{1}{2}||w||^2$$

s.t. $y_i(\langle(w, x_i\rangle + b) > 1$.

The formulation above is relative to the *Hard-SVM* problem, which has a solution if and only if the data are linearly separable. Only a subset of the training data points actually define the separating hyperplane: we call these points *support vectors*. In this sense, the support vectors could be seen as the points that the algorithm deems to be the "most important" since they are the only ones that must be kept in order to recompute the hyperplane, should the need arise. The hard margin assumption can be relaxed to yield the *Soft-SVM* problem, which can be formulated as follows:

$$\underset{w,b}{\operatorname{argmin}} \frac{1}{2}||w||^2 + C\sum_i \xi_i$$

s.t.
$y_i[\langle x_i, w\rangle + b] \geq 1 - \xi_i$
$\xi_i \geq 0$

The minimization problem introduces slack variables ξ_i in order to allow classification mistakes: the penalty paid for committing these errors is proportional to C, an hyper parameter chosen according to the data distribution. For nonlinear classification, it is possible to apply the *kernel trick*, which allows the algorithm to find the maximum margin hyperplane in a higher dimensional space: we replace all dot products between data points with a function $K(x, x') = \langle\phi(x), \phi(x')\rangle$ associated to a nonlinear mapping ϕ. This is equivalent to projecting all data points into a higher-dimensional space. We hope that, in this new setting, the data points will be linearly separable. Once projected back to the original feature space, this will result in a nonlinear split.

7.2.1 Theoretical description of SVM-iCaRL

We take inspiration from [2], which presents a method that addresses the problem of catastrophic forgetting by merging deep architectures and Support Vector Machines. The model depicted in fig. 24 illustrates the functioning of our version of an SVM-iCaRL hybrid. The main changes we apply are the following:

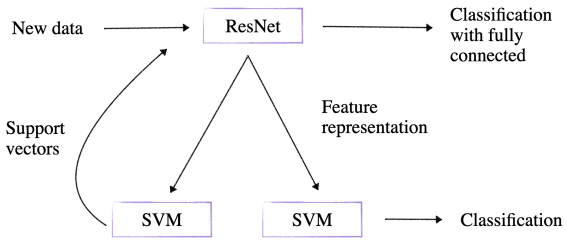

Figure 24. SVM-iCaRL architecture schema

- we use a SVM as classifier, instead of a NCM;
- to increase the synergy between feature extractor and SVM, we propose a different herding strategy based on selecting support vectors as exemplars.

Thus, for each batch of new classes, two distinct SVMs are trained within the framework: one for herding, and one for classification. For herding purposes, the SVM is trained on past exemplars plus all data coming from new classes: the set of support vectors SV_i selected for class i becomes the new exemplar set for class i. If $|SV_i| > m$ (where m is the current maximum size of an exemplar set), then we sample from SV_i until we have m support vectors. Instead, for classification, we only train the SVM using the exemplar sets: according to our experiments, this seems to yield better results than training on exemplars plus all data from the current batch, possibly because of the balance of the dataset. Both SVMs are trained on images mapped in the learned feature space by the network's feature extractor. Training and distillation of the CNN are performed like in iCaRL's original framework. The deep architecture is trained with the new incoming images and the images selected as support vectors for the past classes. The network is divided in two parts:

- The mapping function $\phi(\cdot)$, which includes all layers apart from the last fully connected;
- The softmax layer, i.e. the last fully connected layer of the architecture.

As remarked in [2], the advantage in selecting support vectors is that, despite the small size of the exemplar set, the support vectors are able to preserve the distribution of the old data quite well. We add that, intuitively, support vectors might be seen as the most informative points to train a model on. Indeed, the support vectors alone determine the decision rule of a SVM: further points outside the support vector set would not influence the decision rule of the algorithm. Thus, it is reasonable to think that these points are the most meaningful to plug into a network for distillation purposes.

In [2], the authors propose the use of the fully connected layer as classifier; in our analysis, we compare the performances of both a fully connected layer and a support vector machine on top of our SVM-iCaRL architecture.

7.2.2 Addition of Soft Nearest Neighbor loss

In our SVM-based architecture, we maintain the same classification and distillation losses originally used in iCaRL. To further enhance the performance of the final SVM classifier, we also experiment with the use of an additional contribution to the loss given by the Soft Nearest Neighbor loss, which we already described in section 6.3. Because the goal of this loss is to reduce entanglement between class clusters, its contribution would intuitively be beneficial for a SVM, producing wider margins between classes. We therefore introduce the Soft Nearest Neighbor loss in our SVM-based iCaRL variation, with a temperature of 2.

Its effects on the class separation can be visually inspected in fig. 28, which we discuss in the following section.

7.2.3 Results

In this section we compare the results in terms of accuracy obtained with:

- iCaRL implementation, as baseline;
- Our version of SVM-iCaRL hybrid with fully connected layer as classifier;
- Our version of SVM-iCaRL hybrid with a SVM classifier.

Overall, our SVM-based variation is not able to stand up to the performances of iCaRL (fig. 25). Between the fully connected layer and the SVM, the former achieves slightly higher scores. Again, we delve deeper into the results to get a better sense of the inner workings of our SVM-iCaRL.

We first inspect the accuracies achieved by our SVM-iCaRL setup on old and new classes separately, using the SVM as classifier; to have a baseline, we compare them to accuracies on old and new classes achieved by the original iCaRL (fig. 26). The SVM is consistently accurate on new classes, while it tends to forget old classes more quickly than the original iCaRL. This phenomenon is also visible from the confusion matrix of the SVM (fig. 27). The distillation loss is likely not the problem, since it is the same as iCaRL's.

The issue might be in the herding strategy: inspecting the distribution of exemplars via t-SNE shows that the exemplar sets are not very compact (fig. 28). This can be numerically confirmed by considering the exemplar sets as clusters in the feature space and computing the overall silhouette score

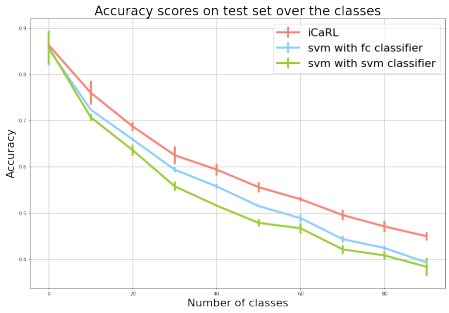

Figure 25. Accuracy scores across each class batch of SVM-iCaRL

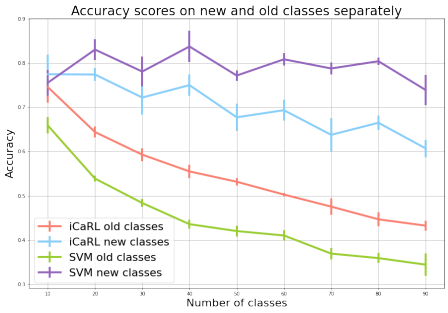

Figure 26. Separate accuracy on old and new classes of SVM-iCaRL

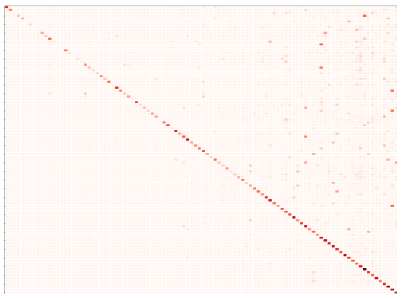

Figure 27. Confusion matrix on 100 classes of SVM-iCaRL using SVM as classifier

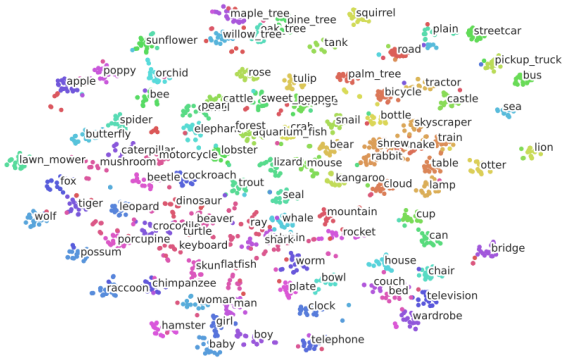

Figure 28. t-SNE visualizaion of 100 feature mapped exemplars set by herding using support vectors

of the clusters. The 100 exemplar sets computed by means of support vectors achieve a silhouette score of 0.10, while the ones obtained with the normal iCaRL herding have a silhouette of about 0.16. Since a higher silhouette means more tightly packed clusters, it seems that the normal herding strategy is overall more effective at keeping exemplars together in the feature space. This does not come as a surprise since, as previously illustrated, support vectors are not supposed to be clustered in a single point: by definition, they are the data points that are close to the margin. Thus, one possible explanation of the performance drop may be that, contrary to our expectations, support vectors do not generalize the overall data distribution so well.

In all our experiments illustrated so far, a linear kernel was used for both SVMs within the framework. Further testing has shown that employing nonlinear kernels, such as RBF, brings no concrete improvement in terms of accuracy. Such result is certainly not anomalous: the network itself is already introducing a deep nonlinearity by means of the feature extractor, and a further nonlinear mapping on top of that will plausibly not affect the classification performances critically. A summary of the results discussed in this section can be found in table 4.

7.3. kNN-iCaRL

In this section, we propose a variation of the iCaRL model that uses the k-Nearest Neighbor algorithm (commonly referred to as "kNN") to classify images. kNN is a supervised learning algorithm which can be summarized as follows. Given an item x to classify and a set of training points:

- we establish a parameter K and a distance metric;
- we find the K nearest training points from x according to the chosen metric;
- we assign to x the label that most frequently appears among the K nearest neighbors of x.

In other words, to classify a new data point, we take a decision based on the labels founds in its surroundings. To know the K closest elements to x, we need to compute the distance from x to each training example: the most commonly chosen metric is the euclidean distance, which we also employ in our experiments. The main feats of the kNN algorithm are its "automatic" nonlinearity and its intrinsically incremental nature. The latter aspect is surely the most interesting in our setting: indeed, the original iCaRL framework uses a Nearest Class Mean classifier, which may be seen as a 1-NN applied on class means in feature space. We shall too follow this approach by applying a kNN in feature space. We further discuss our implementation in the next section.

On the other hand, the downsides of this algorithm are the amount of data which we need to store (normally, the entirety of the training set) and its computational cost at classification time ($O(n)$, where n is the total number of examples). The interest in this classification approach stems from the fact that objects of the same class are of course closely mapped in the feature space and, often, class clusters appear well separated from one another. In this scenario, a kNN could operate well, and perhaps prove to be even more robust than its mean-based counterpart NCM. However, by

Method	10	20	30	40	50	60	70	80	90	100
SVM iCaRL (SVM classifier)	0.858	0.707	0.636	0.558	0.516	0.479	0.467	0.422	0.409	0.384
SVM iCaRL (FC classifier)	0.857	0.716	0.652	0.584	0.543	0.503	0.484	0.439	0.422	0.396
SVM iCaRL (SVM classifier, RBF kernel)	0.886	0.709	0.637	0.541	0.497	0.454	0.450	0.401	0.383	0.363
SVM iCaRL (SVM classifier, poly kernel)	0.861	0.660	0.599	0.517	0.417	0.387	0.352	0.319	0.291	0.277

Table 4. Summary table of SVM-iCaRL

using neighbors instead of class means, we introduce a further hyperparameter in the framework: the K value, which will considerably influence the model's performance. If K is too small, we may consider too few points for the classification and make the classifier sensitive to noise (that is, we overfit). Instead, with an excessively high K, the classifier may take into account too many images across different class regions and consequently underfit. We will discuss all the aforementioned aspects in the following sections.

7.3.1 kNN-iCaRL theoretical description

Our proposal consists in replacing iCaRL's original nearest class mean classifier with a kNN. Incremental training is carried out as in standard iCaRL, and most other aspects of the framework are left untouched. After updating the feature representation for an incoming batch of classes, we fit the kNN on top of the architecture using the feature mapping of the currently available exemplar sets. Fitting the classifier solely with exemplars prevents possible dataset imbalancing that may affect the performance of the classifier itself. This way, given a sample x_i to classify and its feature mapping $\phi(x_i)$, the kNN computes the euclidean distance from the mapping of all exemplars, and the label is chosen among the k-nearest feature representations of $\phi(x_i)$. We keep the same herding strategy proposed in [1]: this ensures that the average feature vector over any subset of exemplars is a good approximation of the true class mean vector. This results in exemplar sets more concentrated around a single point, and consequently aids the classifier in the decisional process. To compute the distance between the mapped images we use the euclidean distance:

$$d(\phi(x_{new}), \phi(x_i)) = \sqrt{\sum_{i=1}^{n} (\phi(x_{new}) - \phi(x_i))^2}$$

$\forall x_i \in \{P_1 \cup P_2 \cup ... \cup P_n\}$

Where x_{new} is the new image to be classified and x_i is the i-th image from the union of all the exemplar sets. For the hyperparameter K, we experiment with two different approaches:

- keeping K fixed at all times;
- adaptively change K according to the current size of exemplar sets.

Moreover, to further accentuate the separation between clusters of images, we again introduce the Soft Nearest Neighbor loss (which we already discussed in section 6.3) as an additional contribution to iCaRL's binary cross entropy loss. For our experiments, we fix the temperature of the Soft Nearest Neighbor loss at $T = 1$.

7.3.2 Results

We now present the results obtained with our kNN-based variation. As a preliminary test, we run two experiments with $K = 5$ and $K = 10$. While $K = 5$ shows a very slight performance degradation, the results achieved with $K = 10$ closely mimic the ones achieved by standard iCaRL, both in terms of total accuracy (fig. 30) and accuracy separately computed for old and new classes (fig. 31). As previously explained, a kNN chooses a label according to the majority class of the neighbors of a point, while the decision of a nearest class mean classifier depends on the closest class mean. Therefore, we can assert that the behavior of a kNN is quite similar to that of iCaRL's Nearest Class Mean classifier, and it does not come as a surprise that kNN and NCM perform similarly.

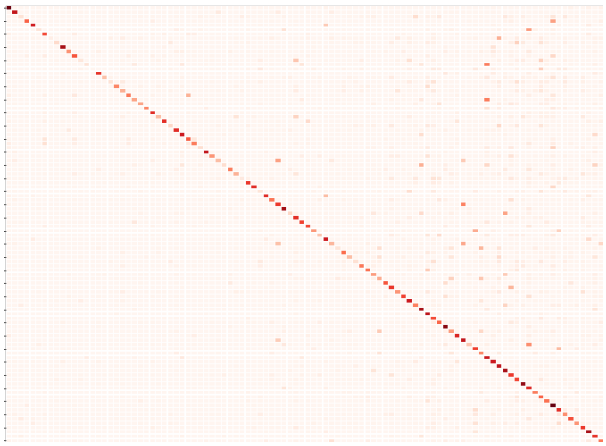

Figure 29. Confusion matrix on 100 classes using kNN-iCaRL setup with $K = \frac{3}{4}m$

Method	10	20	30	40	50	60	70	80	90	100
iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
KNN $k = 10$	0.885	0.757	0.689	0.633	0.601	0.554	0.539	0.497	0.470	0.451
KNN $K = 3/4 * m$	0.851	0.741	0.687	0.629	0.589	0.552	0.529	0.486	0.462	0.437
KNN $K = 3/2 * m$	0.871	0.723	0.673	0.609	0.579	0.540	0.522	0.475	0.454	0.437

Table 5. Accuracies over the number of classes with different K values using the kNN method

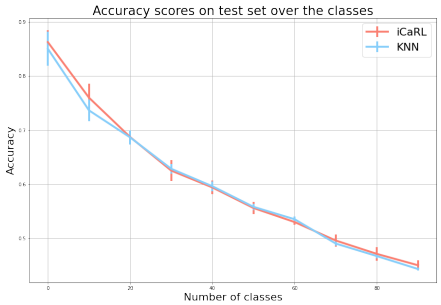

Figure 30.

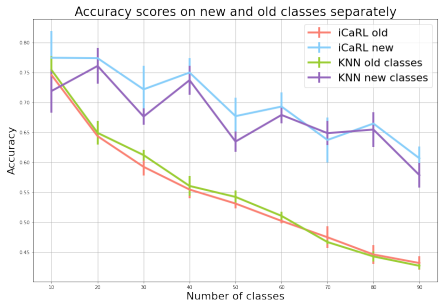

Figure 31.

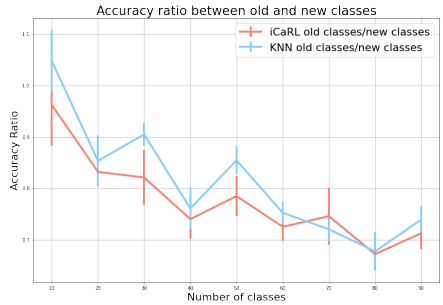

Figure 32.

Accuracy scores across each class batch of kNN-iCaRL using $K = 3/4 * m$ (left);
Separate accuracy on old and new classes of kNN-iCaRL using $K = 3/4 * m$ (center);
Ratio of accuracies on old and new classes of kNN-iCaRL using $K = 3/4 * m$ (right).

We now attempt to vary the value of K according to the size of an exemplar set. Let x_i be a point that belongs to a certain class y_i, and let N_{x_i} be the set of its K nearest neighbors. Because the model increasingly handles more and more classes, and therefore smaller and smaller exemplar sets, it is reasonable to assume that the number of points in N_{x_i} that belong to any given class y decreases at each incremental training step. Thus, we set K to be equal to a fraction of the number of exemplars for each class, in order to take into account the reduction of the exemplar sets. Firstly, we attempt to use $K = \frac{3}{4}m$, where m is the size of the exemplar sets at the current training stage. Again, this result seems to be in line with the original iCaRL configuration, both in terms of overall accuracy and in balance between old and new classes, as shown in the confusion matrices (fig. 4 for iCaRL and fig. 29 for kNN).

For later batches of classes, this is understandable: with this setup, we end up with $K = 15$ nearest neighbors at the last incremental training step, which is reasonably close to the case of K fixed to 10. Indeed, the classification accuracies reflect this result. However, accuracy scores for fixed K and variable K are quite similar even in earlier batches of classes. This might actually be taken as a sign that the feature extractor is working properly: even with a small K, the classes are so well separated that there is no evident sign of overfitting.

We then opt for a more extreme approach and set $K = \frac{3}{2}m$. That is, we intentionally take into account more points than those contained in a single exemplar set. Such an attempt is motivated by the assumption that fitting the whole appropriate exemplar set within the neighbor set of the query point would mean ensuring the correctness of classification. In other words, because at any given time there are only m exemplars of a single class, including all m exemplars of the "right" class within the neighbor set would mean having at most $\frac{1}{2}m$ of any other spurious class, and thus ensuring the right outcome of the classification task. However, our tests show that, while not worsening the performances of the model, using $K > m$ does not bring any real improvement either. Accuracies are consistent with other values of K, and the kNN does not show any noteworthy behavior.

To see whether different losses would be more beneficial to the functioning of a kNN, some tests were also carried out by employing the L2 loss as distillation loss instead of a binary cross entropy (as discussed in section 6.1). However, the results were similar to the ones previously obtained with the L2 loss: a drop in accuracy and a visible overfit in new classes. This is a further confirmation that kNN and NCM essentially behave in the same way, even in the reaction to different losses.

In all of this, we do find one slight benefit in using a kNN with respect to a NCM: on the long run, the former seems less biased towards new classes than the latter. The difference is subtle, but consistent: it is possible to verify it by computing the ratio between accuracy on old classes

and accuracy on new classes at each incremental training step. The closer this ratio is to 1, the more the classifier is balanced between old and new classes. As shown in fig. 32, kNN appears to be slightly less biased towards new classes on the long run, albeit not by a critical amount.

In conclusion, a Nearest Class Mean classifier and a kNN behave in roughly the same way when combined with the rest of the iCaRL framework; this is a totally reasonable result, justified by the similar inner workings of the two classification algorithms. Results illustrated in this section are summarized in table 5.

8. Modifications to the framework

8.1. EnsembleCaRL

In [1], Rebuffi et al. formally establish three points that an algorithm must respect in order to be considered class-incremental; the last of such points imposes limitations on the memory footprint of the model. More specifically, in the iCaRL framework, it is assumed that a limited number of exemplars K can be retained at any given time during training. Such an assumption can be motivated by the fact that the entire training dataset may be too cumbersome in terms of storage, or that past training data was simply deleted or lost. The former motivation seems to be the least probable, as storage support constantly gets less and less costly as technology progresses. We therefore propose a variation of the architecture in which we relax the bound of having limited memory and we allow our storage consumption to grow linearly as new classes arrive. However, we also stay true to the assumption that, indeed, old training data may be unavailable. In other words, we only allow the model to grow in complexity as new classes are presented to it, without ever storing more than K exemplars.

Starting from this idea, we propose an approach based on ensemble methods. Ensemble modeling is a well-established technique for improving prediction accuracy in a machine learning algorithm: it allows to average out noise which would otherwise negatively affect a single model. This leads to the idea of spawning minor *specialized models* as new classes arrive, and using them to assist the main iCaRL architecture at classification time. We draw inspiration from the concept of *specialist model* proposed in [5], where specialists are used to aid the main model in classifying classes that are often confused. More specifically, we propose to enrich the already existing iCaRL framework as follows:

- Each time a new batch of classes arrive, along with the normal training procedure, we clone a version of the current network and fine-tune it on the new incoming classes without any form of distillation. This model becomes the so-called *specialist model* for the current training labels;
- At test time, a preliminary prediction is performed by the main model; this prediction is subsequently refined by taking into account the opinion of the appropriate specialist(s) following one of many possible protocols we later describe.

We expect that this will bring a general improvement of classification accuracy, especially in later stages of training. At the cost of using more memory, we retain clone models that are assigned the specific task of remembering certain classes without the bargain to hold onto the older classes. As [5] suggests, because the specialists are clones of the main model, they can leverage on already learned low-level features: they are therefore trained with a lower learning rate and with fewer epochs than then main network. The loss we employ is the same as iCaRL's, without the distillation term (see section 4.3). The other hyperparameters for specialist training are:

- Learning rate = 0.2;
- Weight decay = $5e-5$;
- Momentum = 0.9;
- Gamma = 0.5;
- Milestones = $10, 20$;
- Batch size = 32;
- Epochs = 30.

The problem now shifts to how to adequately merge the main model's prediction with the specialists' opinions. We propose three possible protocols: we first present them in algorithmic form, and subsequently comment on their effectiveness and the choices that brought to their implementation.

8.1.1 *Hard opinion* protocol

For a given test sample x and its feature mapping $\phi(x)$, we compute the top K nearest class means from $\phi(x)$ like we would when using the normal iCaRL classify method. K is an hyperparameter: in our experiments, $K = 3$. We denote as L the set of labels obtained this way. For each $l_i \in L$, we take the specialist associated with l_i and compose the set of involved specialists S (we exclude repeated specialists, if any). The sample x is then forwarded through each specialist $s_i \in S$. We apply a softmax to each activation vector $out_i = s_i(x)$, obtaining the probability distribution $p_i = Softmax(out_i)$. We call $p_{i,max}$ the maximum probability found in p_i. The position of that probability within p_i is the label that the specialist would predict for x: we call this the *response* r_i. The final prediction for x becomes the r_i associated with the highest $p_{i,max}$. In other words, the

specialist that is more certain of their prediction sentences the definitive classification (hence the name of the protocol).

8.1.2 *Soft opinion* protocol

Given a sample x, and its feature mapping $\phi(x)$, we construct the set L as illustrated in the previous point. This time, we sort L by how far $\phi(x)$ is from the class mean of each predicted label – that is, we sort the labels in S "by importance". l_1 is now what we would normally predict by using a nearest mean classifier. For $i = 1, 2...K$, we feed the specialist associated to l_i with x and again pass its response though a softmax to obtain the distribution p_i. The prediction that the specialist would make for x is $l'_i = \text{argmax}(p_i)$. If that prediction coincides with the starting label l_i, then the prediction l_i is confirmed. Otherwise, we step to the next value of i and repeat. The number of considered labels K is an hyperparameter of the model. In our experiments $K = 3$.

If no match between specialist and main model is found, we can decide to give the ultimate choice to the specialists (and fall back to the *hard opinion* protocol) or to the main model (and return l_1 as final prediction). In other words, for each plausible label found by the main model, in order of importance, we ask the opinion of the appropriate specialist. We confirm the first potential label which main model and specialist agree on. The main difference with the previous protocol is that, instead of giving the same importance to all specialists' opinions, here we attempt to give them different priorities according to the prediction of the main model.

8.1.3 *Mixed opinion* protocol

We also propose a third approach which seeks to further balance specialists' and main model's opinions, based on finding the intermediate probability distribution amongst the ones produced by all models. We again take inspiration from [5]: given a sample x to classify, we seek the probability distribution q^* such that

$$q^* = \underset{q}{\text{argmin}} \, KL(p^g, q) + \sum_{s_i \in S} KL(p^{s_i}, q)$$

Where

- KL is the Kullback-Leibler divergence;
- p^g is the softmax output of x when forwarded through the general model;
- p^{s_i} is the softmax output of x when forwarded through the specialist s_i.

The main difference with the previous two protocols is that here we do not use the normal Nearest Mean Classifier on top of iCaRL's architecture, but rather we employ its fully connected layers. This makes it possible to uniformly blend the main model's opinion with the ones produced by the specialists: indeed, the KL divergence between two distributions can be seen as a measure of distance between two probability vectors. This time, we consult the specialists associated to the K most likely classes predicted by the main model for the given query. K is an hyperparameter of the model; in our experiments, $K = 3$. Experiments showed that the best results are achieved by taking into account each specialist's opinion at most once, i.e. even if a specialist is associated with more than one label within the top K, we add its contribution to the equation above only once. The result q^* is the distribution that sits midway from the probabilities obtained from all models. The final predicted label is $\text{argmax}(q^*)$.

8.1.4 Results

The following configurations were evaluated and compared to the original iCaRL architecture:

- Ensemble of specialists with hard opinion;
- Ensemble of specialists with soft opinion;
- Ensemble of specialists with mixed opinion.

As previously mentioned, $K = 3$ for all protocols listed. The first protocol achieves the worst performance out of the three. Though it is barely noticeable, hard opinion slightly decreases the accuracy of the model, especially in the last batches. Further inspection showed that this is probably due to wrong specialists being more "convinced" on their wrong predictions more than the correct specialists: that is, a wrong specialist may return a higher probability for a wrong label than the correct specialist does for the right one.

It is possible to verify this assumption with the following experiment. We take image x_i of true label y_i, in an instance in which it is correctly predicted by the main model; and we then extract the second and the third most likely label returned by the main model. We gather the specialist associated to label y_i and the specialists of the other two labels. Because the labels of specialist models do not overlap, the last two specialists have never actually seen an instance of y_i. We feedforward x_i into the three specialists and extract the hard opinion (i.e the label associated with the highest probability) returned by these models: in some cases the probability associated to a incorrect label returned by a wrong specialist is higher than the one associated with y_i returned by the correct specialist. We display this misbehavior in fig. 33: even though the correct label is 16, the

specialist that handles classes from 10 to 19 returns a probability lower than the one obtained from the specialist of classes 90-99 (which is not supposed to be able to classify images of class 16). In short, the problem of this configuration seems to be the excessive attention given to one single specialist, which may sometimes be wrong.

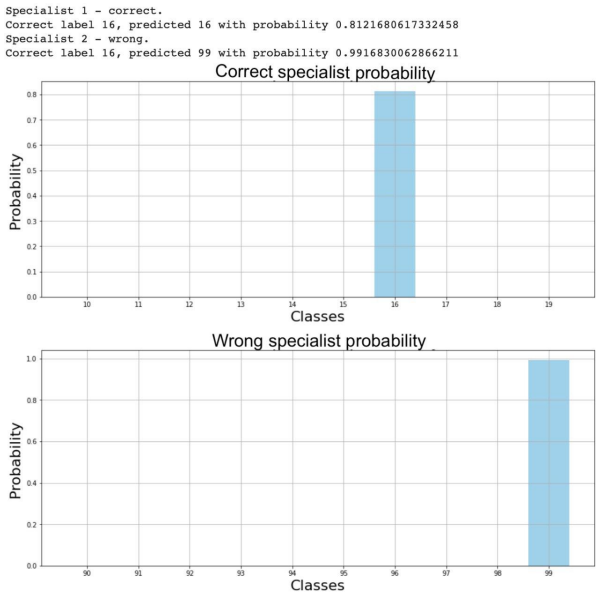

Figure 33. Output probabilities of two specialists for sample x of true label 16: the correct specialist (above) predicts label 16 with 81.2% probability, while an unrelated specialist (below) predicts label 99 for the same sample with 99.1% probability. This leads to an incorrect classification

The *soft opinion* protocol is designed to address this problem: this time, we prioritize more the main model's opinion. The specialists do not make a prediction on their own: we use them to confirm or reject a prediction proposed by the main model. This approach re-aligns classification performances with those of iCaRL, but does not bring any additional improvement either (fig. 34).

Indeed, one could say that the scenario in which this method is supposed to surpass iCaRL's flaws is somewhat rare: *soft opinion* will outperform the original algorithm only in cases when the correct label is actually within the top K prediction of the NCM, and the corresponding specialist is able to identify it. Only then this method actually differs from iCaRL; in all other cases, the two algorithms converge to the same outcome. Thus, we conclude that *soft opinion* seems to suffer from the opposite problem of *hard opinion*: too much relevance is given to the NCM alone, minimizing the advantage of the ensemble method.

With the mixed method, we seek a balance between the two previous approaches by adequately mitigating both specialists' and main model's opinion. Finding an equidistant

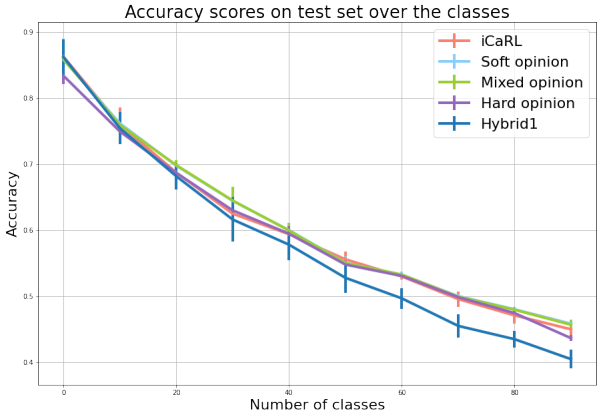

Figure 34. Accuracy of EnsembleCaRL across batches of classes with *hard*, *soft* and *mixed opinion* protocols

probability distribution across all predictions allows to take into account all classifiers' opinions simultaneously and results in slightly improved performances - sometimes higher than standard iCaRL, though by no more than 1% (fig. 34).

Interestingly enough, this method does not use a NCM, but performs the prediction by using the final fully connected layer of all models. In other words, the configuration of the mixed opinion is closer to what [1] calls by the name of Hybrid1. From this standpoint, our method sensibly improves its original counterpart, achieving up to 5% more accuracy in later training batches (fig. 34). This, of course, comes at the price of making use of more memory.

A summary of the results illustrated in this section can be found in table 6.

8.2. FamilyCaRL

It is plain by now that one of the main issues in incremental learning is handling the bias that an incremental model might develop in predicting old or new classes. In the other sections, we have presented several methods that attempted to tackle this problem. In section 7.1, an ad hoc classifier that rebalances weights in the last fully connected layer was proposed; in section 8.3, the rebalancing operation is carried out by means of exemplar generation and augmentation of the training data. In this section, we propose a further approach to the rebalancing problem based on the decoupling of the training and distillation procedures.

Instead of rebalancing the data, we focus on achieving equal relevance between the old version of the network (trained on past classes) and the new version of the network (trained on the lastly available data). We do so by distilling both models in a new network simultaneously. More specifically, for a given batch of classes C_t to learn at training step t, we do the following:

- We create a new instance of the CNN used by the

Method	10	20	30	40	50	60	70	80	90	100
Accuracy iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
Accuracy hard	0.832	0.749	0.681	0.612	0.584	0.557	0.532	0.496	0.473	0.440
Accuracy soft	0.877	0.763	0.697	0.638	0.595	0.556	0.534	0.500	0.480	0.463
Accuracy mixed	0.879	0.759	0.698	0.638	0.594	0.556	0.534	0.499	0.479	0.460

Table 6. Accuracies over the number of classes using iCaRL, Hard variation, Soft variation, Mixed vatiation

model. This can be done by either initializing it with random weights or cloning the current version network. The latter option is preferred to preserve prelearned low level features, as suggested in [5] with regards to knowledge distillation. We call this new instance of the network the *parent model*;

- We train the parent model on C_t only, without the using exemplars; during training, we encourage the network not to shift the feature representation too much by applying the Less forget constraint introduced in section 7.1.2:

$$L(x) = 1 - \langle \bar{f}_{new}(x), \bar{f}_{old}(x) \rangle$$

Similarly, any classification loss can be used; in our tests, we use binary cross entropy, the same as iCaRL's. For comparison purposes, we also keep the same training hyperparameters.

We now have a model trained on the new classes only (the *parent model*) and a model trained on old classes only (which we may fittingly call *grandparent model*). Ideally, both models should have similar underlying feature mappings. After performing the training of the parent model, a set of exemplars for each class is selected from C_t. The herding strategy could be any; in our tests, in order not to discriminate between feature mapping of parent and grandparent, we stray from iCaRL's original method and simply select exemplars at random. In section 5, we show this approach does not hurt performances in any noticeable way. Finally, we generate a third CNN by cloning the parent, to obtain what we call a *child model*. The last step of training consists in simultaneously distilling *parent* and *grandparent* into *child*; the distillation process is carried out only using exemplars, to avoid unbalancing of the training data for the *child*. At the next incremental training step, the current *child* model will take on the role of *grandparent*. The whole process is summarized in the figure 35.

For each exemplar x and each label i, we compute:

- The output $g_i(x)$ of the grandparent when forwarding x (including a final sigmoid)
- The output $p_i(x)$ of the parent when forwarding x (including a final sigmoid)

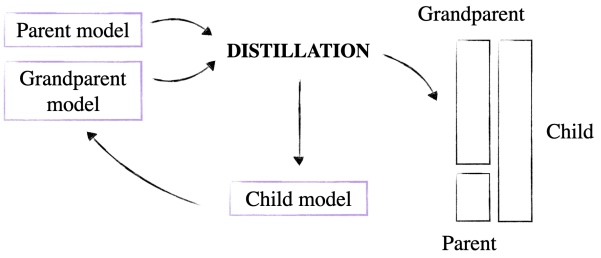

Figure 35. FamilyCaRL architecture schema

- The output $c_i(x)$ of the child when forwarding x (including a final sigmoid)

The distillation contribution of the parent is given by a binary cross entropy only computed on its labels (i.e. the labels of the incoming batch of classes):

$$L_p(x) = \sum_{i \in C_t} -p_i(x) \log(c_i(x)) - (1 - p_i(x)) \log(1 - c_i(x))$$

Analogously, the grandparent contributes to the distillation only on its labels (i.e. the past labels):

$$L_g(x) = \sum_{i \in C_j, \forall j < t} -p_i(x) \log(c_i(x)) - (1 - p_i(x)) \log(1 - c_i(x))$$

The overall distillation loss becomes:

$$L_{dist}(x) = L_g(x) + L_p(x)$$

For the sake of comparison, our FamilyCaRL uses the same classification and distillation losses as iCaRL (apart from the Less Forget Constraint). However, more refined choices might be made to adapt the framework to a more specific problem. Likewise, any classifier could be used on top of the architecture; again, for comparison purposes, we use the same as iCaRL's, a Nearest Class Mean classifier.

Because the distillation process is decoupled from the training on C_t, different hyperparameters for classification and distillation can be used. In our tests, we found the following configuration to work reasonably well: 60 epochs in total and a less aggressive initial learning rate of 0.9, which

we step down to a third every 30 and 50 epochs. More fine-grained tuning may lead to even better results.

The distillation from parents to child is completely unsupervised. Further experimentation, which we lack the time to carry out, might reveal some benefits in enriching the exemplar set with images not belonging to any previously seen class (in [5], Hinton et al. propose the idea of performing distillation with an unlabeled *transfer set*).

8.2.1 Results

We compare the results of our proposal (which, again fittingly enough, we refer to as FamilyCaRL for conciseness) with the original iCaRL baseline with the following metrics:

- Total accuracy across batches of classes;
- Accuracy score separately on old and new classes;
- Ratio of separate accuracies on old and new classes, to test which method is most balanced between old and new classes. This is simply computed as Acc_{old}/Acc_{new}. The closer to 1 this ratio is, the less the algorithm predictions are biased towards old or new classes.

Overall, our method can stand up to iCaRL fairly well, achieving almost the same results (fig. 36). The rate at which iCaRL and FamilyCaRL forget past classes is essentially equal; this can be confirmed by computing accuracies on old classes only (fig. 37).

Nevertheless, our method scores on average 2% less that the original iCaRL in all batches. This trend also manifests itself when computing accuracies on new classes only (fig. 37).

In exchange of a small percentage of overall accuracy, FamiliCaRL appears to be less biased than the original iCaRL towards new classes: we verify it by computing the ratio between accuracies on old and new classes at each batch. Fig. 38 illustrates that our method is closer to the value 1 (highlighted in green) across all batches, and is therefore generally less biased than iCaRL. A summary of the results discussed in this section can be found in table 7.

8.3. Training set rebalacing

The major challenge in incremental learning is of course the assumption that past data cannot be fully retained for future training steps. The practice of exemplar selection tackles this problem to a certain extent, but often results in a very unbalanced training dataset. In this, section, we try to address this issue by means of several techniques. At any training step t, the number of images seen by the model during t is equal to $500 * 10 + 2000$, where the first term represents the unseen images, 500 for each new class, and the second term is the number of total exemplars stored in iCaRL memory, a fixed amount. The number of images that composes each exemplar set depends on t and decreases overtime, since the 2000 possible images kept in memory have to be shared by an increasingly large number of classes. Therefore, the training set unbalance grows bigger overtime. Some solutions for this problem may be the following:

- Undersampling: only use a subset of the labels in excess to rebalance training. Since deep architectures usually require a lot of data in order not to overfit, this solution is avoided;
- Oversampling: artificially generate new images based on the ones available.

One of the three principles established by iCaRL [1] is that the amount of memory used must be fixed: thus, we propose different techniques in order to create new images on the fly for a given training step.

8.3.1 Sampling around the class mean

The first and simplest method we propose to generate new observations is to start from the mean image for each class label and sample points in its proximity, moving away from it by a random factor. More specifically, the mean μ_c and standard deviation σ_c for each class c is computed and a random contribution r is added to it, following the rule:

$$x_{new} = \mu_c + r\sigma_c$$

For this approach to make sense, the assumption of normality of the distribution of pixels in an image is required: this is of course a very unrealistic requirement, which immediately showed its limitations during preliminary testing. One could argue that, by Central Limit Theorem, a large enough number of images will eventually satisfy such condition; however, as previously stated, the number of exemplars available for a class decreases as the training steps increase. Although at the beginning 200 exemplars for each class are available, fewer and fewer will be retained by the model as training goes on, progressively moving away from a scenario in which we could appeal to the Central Limit Theorem. We ended up immediately discarding this naive approach and moved onto more refined techniques.

8.3.2 SMOTE technique

The second approach proposed takes inspiration from the Synthetic Minority Oversampling Technique (SMOTE). This technique generates synthetic exemplars by introducing new observations taken along the line segments joining any or all the images that belong to the corresponding exemplar set. In other words, given a limited set of images that

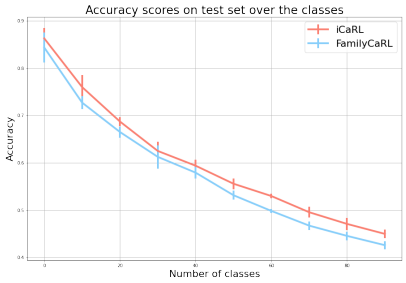

Figure 36. Overall accuracy comparison between iCaRL and FamilyCarl

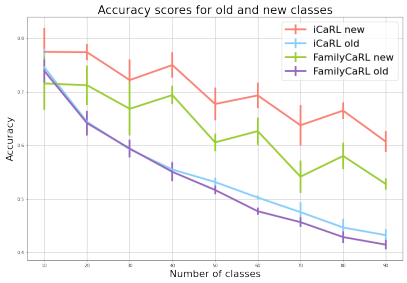

Figure 37. Old and new classes accuracy comparison between iCaRL and FamilyCarl

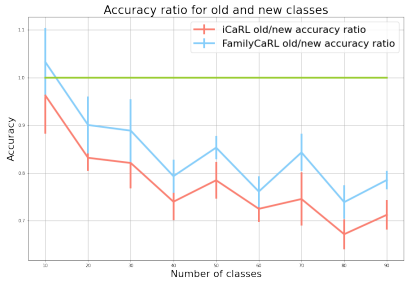

Figure 38. Ratio old/new accuracy comparison between iCaRL and FamilyCarl

Method	10	20	30	40	50	60	70	80	90	100
Accuracy iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
Accuracy FamiliCaRL	0.844	0.727	0.665	0.613	0.579	0.532	0.498	0.467	0.446	0.426
Ratio old/new classes iCaRL	//	0.96	0.82	0.81	0.75	0.78	0.73	0.75	0.67	0.71
Ratio old/new classes FamiliCaRL	//	1.21	0.9	0.89	0.80	0.85	0.76	0.85	0.75	0.78

Table 7. Accuracy scores and ratio of accuracy between old and new classes across class batches of iCaRL and FamilyCaRL

belong to the class that we wish to augment, we trace "high dimensional lines" connecting the images and we draw new samples from such lines. We use this technique to augment the exemplar sets built as described in [1], where images are chosen according to their closeness to the class mean.

Practically, rather than applying augmentation to the old exemplars, we generate new ones artificially by exploiting the information encoded in the relative position of pairs of observations belonging to the same class. In doing so, we consider the distribution of the "best exemplars" that can be extracted from the training set. These synthetic images are then subjected to the same transformations that exemplars undergo. This way we select exemplars, which are supposed to be well clustered, and among them we create a variable number of images that depends on the numerosity of the exemplar set. At the end of the process we obtain a perfectly balanced dataset, which we then use to train the network.

SMOTE has a drawback: it will generate artificial exemplars that, by construction, lie within a portion of hyperspace enclosed by the original exemplars. If exemplar sets are well separated, this can lead to improvements in the classification performances. However, in a setting in which images of different clusters overlap, SMOTE can make the situation worse, accentuating the overlapping phenomenon.

8.3.3 Network aided generation

The criticality of the previous approaches is that, in both cases, they require to make some assumptions on the distribution on the data points. We therefore try to tackle the problem from another standpoint, by making use of a deep architecture. For each old class, starting from a set of exemplars, we train a network to mimic samples from that specific class. This is done by a CNN with 3 x 32 x 32 output neurons, trained as follows:

- for an exemplar e_i belonging to class i, we generate a synthetic image x composed of gaussian noise;
- we forward x through the network to obtain $net(x)$;
- we compute the loss for a single sample as $L(x) = |net(x) - e_i|$, and backpropagate.

By doing this, we hope that the new generator network will arrange its weights so that, when given whatever random noise, will modify it to make it as close as possible to the overall distribution of an exemplar set. We train one network for each incoming class.

After training the generator networks, we can use them to generate any number n of exemplars of needed to rebalance the dataset at any time by feedforwarding n images of random gaussian noise. In our experiments, we found that these generator networks converge to somewhat of a "mean" image (which does not necessarily correspond to the actual average image of the class). This behaviour is shown in figure 39, where we display how the output of the network evolves during training for the class *apple* (the input is always gaussian noise): at the beginning the output still resembles a random vector (bottom left), but after some batches of training it takes on the shape of an apple (bottom right).

We found that the exemplars generated with this method are visually quite similar. The lack of variability may affect the effectiveness of training of a network that, on the contrary, would have to be trained on a large number of

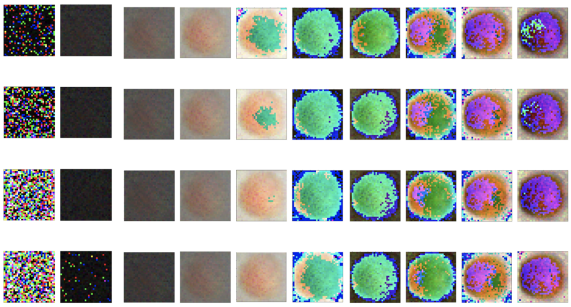

Figure 39. Evolution of a random image to an *apple* during the training of a network

different images. Indeed, results of this method were not satisfying. Moreover, this method involves training a whole network for each new incoming class: not only is this very computationally expensive, but it also employs additional memory.

8.3.4 Maximal activation of neurons generation

We seek more variability in the data and a lower memory consumption in a method that again involves the use of a deep architecture. Specifically, this time we employ the main CNN underlying iCaRL. We propose a means of generating an exemplar of class i by artificially synthesizing an image which maximizes the activation of the i-th output neuron of the network. Formally, we seek the input I^* such that:

$$I^* = \underset{I}{\operatorname{argmax}}(out_i(I))$$

Where $out_i(I)$ is the output of the i-th neuron of the network. We do this by freezing the parameters of the network (both weights and biases) and generating a starting image x of random gaussian noise.

We can then pass x to any optimizer (in our tests, we use Stochastic Gradient Descent), and iteratively update the pixels of x to satisfy the objective function (in our case, the maximization of the activation of a specific neuron). For our implementation, we minimize the following objective function:

$$L_i(x) = M - out_i(x)$$

Where M is an arbitrary parameter. For our experiments, $M = 1000$. By minimizing this difference, we of course maximize the activation that corresponds to the label for which we want to generate observations. To add further randomness to the generation, the learning rate of the SGD is set to be a random number between 0.2 and 0.5. The idea is to "change the path" along which the optimizer finds the maximal activation, and consequently generate different images. In this case, the generated exemplars do not display any distinguishable pattern or shape as before. At least, not to the human eye: to the network, however, they are clearly and evidently samples of their original class.

8.3.5 Results

In this section we illustrate the results obtained with the previously described techniques. In this scenario, not only do we consider the performance obtained in terms of accuracy, but also if the algorithm is feasible in terms of computation time.

The augmentation based on sampling around the mean is certainly the least computationally expensive method among the ones described. However, as previously hinted, this technique was anything but successful: its usage led to sensible worsening of accuracy scores, probably because of the unnatural way of generating exemplars. The model was visibly led astray by this new synthetic data, so we immediately discarded this approach.

While SMOTE was able to achieve slightly better results, the overall accuracy still experienced a drop when using it. We can visually inspect the result of the augmentation process using t-SNE: fig. 40 and 41 shows the feature representation of the data available at the second training step - that is, 10 exemplar sets of 200 samples and the 500 samples for the 10 new incoming classes. Fig. 40 shows the feature space with no augmentation, while in fig. 41 the exemplar sets have been rebalanced with SMOTE. Some overlapping is visible between the main clusters of the new classes and the augmented sets, which may explain the performance drop. Like the previous method, SMOTE is purely based on the geometrical information retained at each training step within the exemplar sets - which of course diminishes over time. The last tests highlighted that, indeed, these methods are not fit for this scenario, and trying to rebalancing the data by solely exploiting the information encoded in the exemplars' geometrical disposition is not enough.

We confirm this assumption, it is possible to check the variation of accuracy separately for old and new classes (fig. 43). Indeed, accuracy on new classes is quite stable and aligns with the original iCaRL, only with a slight low decrease at the very end of the training procedure. Instead, classification on old labels visibly drops in accuracy (with respect to the original iCaRL) when the synthetic images are introduced in the training phase. This proves that the damage introduced by SMOTE resides in the old classes alone, which are the ones involved in the augmentation process.

The numerical results described so far are summarized in table 8.

We now move to the inspection of the methods based on the use of neural networks to generate new images. The net-

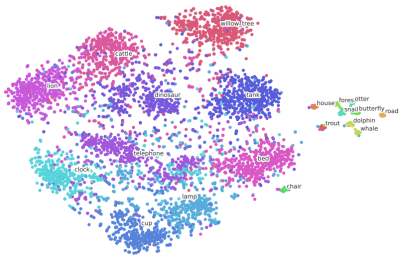

Figure 40. t-SNE representation of new classes + exemplars in the second training step without augmentation

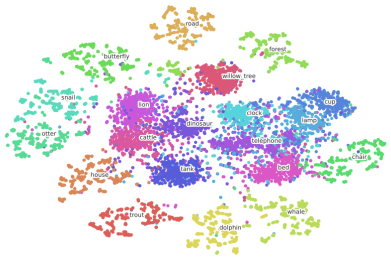

Figure 41. t-SNE representation of new classes + exemplars in the second training step with SMOTE augmentation

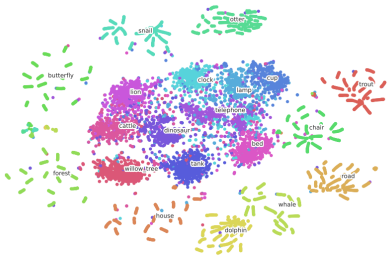

Figure 42. t-SNE representation of new classes + exemplars in the second training step with max activation

	Method	10	20	30	40	50	60	70	80	90
old	iCaRL	0.746	0.644	0.593	0.555	0.532	0.503	0.475	0.447	0.433
	SMOTE	0.686	0.542	0.470	0.417	0.378	0.336	0.293	0.250	0.230
	Network	0.711	0.630	0.558	0.528	0.506	0.450	0.381	0.357	0.331
	Max Activation	0.728	0.626	0.568	0.517	0.493	0.454	0.399	0.380	0.354
new	iCaRL	0.775	0.774	0.722	0.750	0.677	0.693	0.638	0.665	0.607
	SMOTE	0.713	0.745	0.711	0.765	0.706	0.728	0.678	0.738	0.704
	Network	0.774	0.732	0.693	0.727	0.664	0.715	0.714	0.745	0.709
	Max Activation	0.754	0.796	0.707	0.762	0.680	0.764	0.732	0.756	0.692

Table 8. Accuracies separately for old and new classes using iCaRL, Smote, Network aided and Max Activation

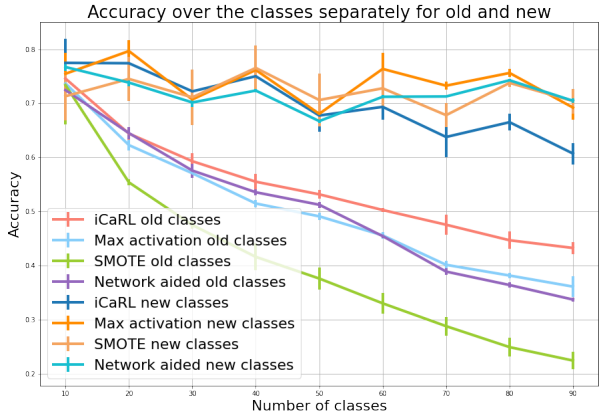

Figure 43. Accuracy trend on old and new classes for different augmentation techniques

work aided generation achieves discrete results when compared to the method described so far (see table 9). It seems that, despite still having only the exemplars available to learn from, and thus a diminishing source of information like the previous cases, the network has a good grasp of the necessary information to retain. Of course, the synthesized image is not always distinguishable to the human eye; nonetheless, the network seems to be able to get more sense out of it compared to SMOTE, as proven by the higher accuracy scores achieved.

However, this augmentation methods still has two flaws: firstly, it still slightly worsens iCaRL's performances; in the second instance, it requires a great deal of computation time, since it calls for the training of a network for each different class.

With the last method, which involves maximizing the activation of a single neuron of the main network, we partially solve the latter issue: despite being somewhat computationally expensive (it still requires a numerical maximization process for each bunch of generated exemplars), it is certainly more feasible than training a whole network for each class. This method also seemed to generate the most plausible version of synthetic exemplars: this is particularly evident from the t-SNE representation in fig. 42, in which we can see a wider gap between old classes (at the center) and new classes (arranged in the outer border). We were able to further verify the plausibility of the generated exemplars by verifying that the following two measures were numerically comparable:

- cosine similarity between artificial exemplars and class mean (in feature space);
- cosine similarity between genuine exemplars and class mean (in feature space).

Both quantities oscillated around 0.8-0.85, which is plausible since we operate with normalized features.

Because this method does not make use of the current available exemplars but of the "knowledge stored" within

Method	10	20	30	40	50	60	70	80	90	100
iCaRL	0.863	0.760	0.687	0.625	0.594	0.556	0.530	0.496	0.471	0.450
Smote	0.848	0.700	0.610	0.530	0.486	0.433	0.392	0.341	0.304	0.278
Network aided	0.856	0.769	0.685	0.642	0.578	0.547	0.499	0.432	0.412	0.381
Max Activation	0.858	0.739	0.683	0.605	0.566	0.524	0.501	0.442	0.423	0.402

Table 9. Accuracies over the number of classes using iCaRL, Smote, Network aided and Max Activation

the network itself, it does not suffer the same level of degradation as SMOTE; however, its effectiveness depends on the classification performance of the network itself. If the model tends to misclassify a lot, then the synthetic exemplars will unavoidably mimic the model's tendency to make mistakes, possibly initiating a negative feedback loop. This might explain why the accuracy score of iCaRL with maximum activation exemplars initially keeps up well with the original iCaRL, but then it slowly diverges overtime (fig. 44). This increasing discrepancy might be the negative loop taking over. Unfortunately, a complete training run with this final augmentation setup ends up scoring 4% below the original iCaRL in the final batch of classes.

Accuracy scores in each training step for all previously described methods are summarized in table 9.

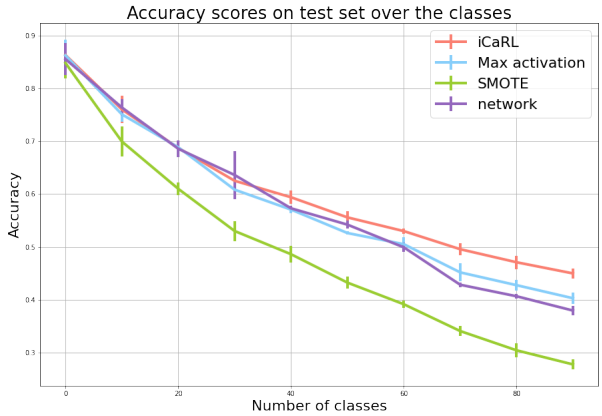

Figure 44. Accuracy comparison of different rebalancing methods

References

[1] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representaton learning."

[2] Y. Li, Z. Li, L. Ding, Y. Pan, C. Huang, Y. Hu, and X. G. Wei Chen, "Supportnet: solving catastrophic forgetting in class incremental learning with support data," 2018.

[3] S. Hou1, X. Pan, C. C. Loy, Z. Wang, and D. Lin, "Learning a unified classifier incrementally via rebalancing."

[4] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2012.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[6] Z. Li and D. Hoiem, "Learning without forgetting," 2017.

[7] N. Frosst, N. Papernot, and G. Hinton, "Analyzing and improving representations with the soft nearest neighbor loss," 2019.