Class Incremental Learning in image classification

Laura Gabriele, Linda Ludovisi, Riccardoluigi Trinchieri
Politecnico di Torino
Corso Duca degli Abruzzi, 24 - 10129 Torino, Italy
{s278439, s278221, s278170}@studenti.polito.it

Abstract

The aim of this paper is to experiment some algorithms in the field of the class incremental learning (IL) for classification of images. First, an analysis which compares different approaches (fine-tuning, Learning without Forgetting, iCaRL) was done in order to understand pros and cons of each method, resulting in iCaRL being the best performing one. Then, ablation studies on iCaRL model concerning changing the losses and the classifier were done. Finally, another study regarding the role of exemplars in iCaRL was done in order to understand why they are so important and their influence on the results. All the experiments were conducted using CIFAR-100 dataset.

1. Introduction

In real-world applications, the need of learning tasks incrementally is becoming more relevant. Different data may be gathered at different time and it is not always possible to ensure the possibility to retain previous data in their wholeness, due to memory or legal issues. With an incremental learning approach, the learning steps are performed continuously by involving new data over time and without training the model from scratch. The main weakness of this types of algorithms consist in the tendency of the network to forget previously learned information when trained on new data, despite how well it could handle the previous task: this phenomenon is known as *catastrophic forgetting*. If it was possible to acquire new knowledge without causing the aforementioned effect, neural networks could be improved under many aspects (e.g. memory requirements, computational time, etc.).

The scenario under analysis, in this project, is *Incremental Class Learning* in image classification: make a previously trained network capable to learn and predict unseen classes. Over the course of the last few years, different methods were introduced to reach this goal: Learning without Forgetting [4] and the state of the art iCaRL [8] are some of the most relevant. LwF introduced the usage of a *distillation loss*, which brought huge improvements, enabling the network to retain the previously learned knowledge when trained for a new task, while iCaRL underlined the importance of feeding the network with a small amount of previous data to remember previously learned distributions and the importance of a network detached classifier.

In the following sections the iCaRL model will be discussed and compared to fine-tuning and Learning without Forgetting. Possible modifications will be analyzed to give a reliable explanation of its performances.

2. Method

In this section the components of the aforementioned methods are described: the shared components (the dataset settings and the network architecture), the main components of iCaRL (Section 2.1), the main components of fine-tuning and LwF models (Section 2.2).

Dataset. The dataset used in these experiments is the CIFAR100. This dataset contains 60.000 32x32 images belonging to 100 different classes, divided into train and test set, having 500 training images and 100 testing images per class. The learning procedure consisted in randomly splitting the dataset in 10 batches of 10 classes, performing the training sequentially over batches, until all classes are learned.

Architecture. All methods make use of a *convolutional neural network*: the 32-layers Resnet (the so-called Cifar Resnet, as described in [1], since it matches the size of the images in CIFAR10 and CIFAR100). The network is considered a feature extractor, followed by a fully connected layer having as many output nodes as classes observed at each iteration.

The output nodes corresponding to a certain class y will be referred (Section 2.1, 4.1) with $y \in \{1, ..., s-1, s, ..., t\}$, where nodes going from $1, ..., s-1$ refer to the previously learned classes and nodes going from $s, ..., t$ are the ones of the new classes. The output of the network, which corresponds to the result of forwarding the data into the feature extractor layers and the fully-connected, is denoted as $g(x)$.

Data Preprocessing. All the images are transformed before being forwarded to the net. Both training and test images are normalized with the mean and standard deviation of the CIFAR100 dataset. Data augmentation is also applied to the training set, with a $padding = 4$ and a 32×32 crop is randomly sampled from the image or its horizontally flipped version.

2.1. iCarl Implementation

To approach this Incremental Learning problem the iCaRL [8] model was implemented. This model tries to avoid catastrophic forgetting by taking the following precautions:

- It uses the *deep neural network only for feature extraction* while delegating the *classification task to a detached classifier*.
- It stores some previous data, which will be called *exemplars* in our scenario. The amount of memory assigned to this task is fixated and decided a priori. This data is used for classification purposes, but also during training phases: *rehearsal of previous experience* helps the network to remember previous data distribution over time.
- It uses a *distillation loss* in addition to the standard classification loss, to avoid drastic changes in the network weights when training on new data.

Training. The training phase is made of three sequential steps: first, the feature extractor is trained using both the new data available at the current step and all the exemplars, updating how the network extracts features from images. Then, since the amount of memory for previous data cannot exceed the fixated threshold, the previous exemplars sets are reduced in size to make room for exemplars belonging to new classes. Finally, new exemplars sets are constructed for the new learned classes. Exemplar sets always share the same size, which is given by *m* where:

$$m = \frac{amount\_of\_memory}{number\_of\_learned\_classes}$$

Constructing Exemplar sets. New exemplars are chosen according to the algorithm described in iCaRL paper [8]. Exemplars sets are *prioritized lists* since, during construction, the image best representing the original class mean is firstly chosen, then, the second most important and so on. This makes the reduction task extremely easy since it consists in *discarding the last N elements*. All the operations are computed on features extracted by the convolutional layers of the network (features are *L2-normalized*, likewise the results from any operation on features), but the images are stored as the originals, since the feature extraction process is continuously changing as new classes are learned.

Loss. The loss used for both classification and distillation is *Binary Cross Entropy*, and it takes as inputs the sigmoid of the current network outputs for classification, as well as the sigmoid of current and previous network outputs for distillation. Before starting a new training step the network is copied, and its outputs are stored. Furthermore, classification loss is computed on outputs belonging to new classes, while distillation loss is computed on the remaining ones.
The loss $\mathcal{L}$ is calculated as:

$$\mathcal{L} = - \sum_{(x_i,y_i)} [\sum_{y=s}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y \neq y_i} \log(1 - g_y(x_i)) \\ + \sum_{y=1}^{s-1} q_i^y \log g_y(x_i) + (1 - q_i^y) \log(1 - g_y(x_i))] \tag{1}$$

where output nodes go from 1 to t (nodes referring to previously learned classes are the ones going from 1 to $s-1$); $g_y(x_i)$ and q_i^y are respectively the sigmoided outputs calculated from the actual and the previous network, computed on the x_i^{th} image, referring to the y^{th} class.

Classification. After the training phase only the convolutional layers of the network are considered because *feature extraction alone is needed for classification purposes*. For each class a representative point is chosen, which consists in the mean of the exemplars for that class (if the class is previously learned) or the mean of all the available data (if the class is just learned). The mean is computed on the outputs of the feature extractor and also in this case features and results of any operation on them are L2-normalized. The classifier is NME (*Nearest mean of exemplars*) and it is computing the *L2 distances* among points to be classified and means of the classes. The label assigned correspond to the one of the class mean closest to the point.

2.2. LwF and fine-tuning implementation

To see the effectiveness of the iCaRL model, it was compared with two more naive models: fine tuning and LwF [4]. The first one does not take any precaution to prevent catastrophic forgetting, while the latter is using the distillation loss, while not using a detach classifier or storing any previous data.

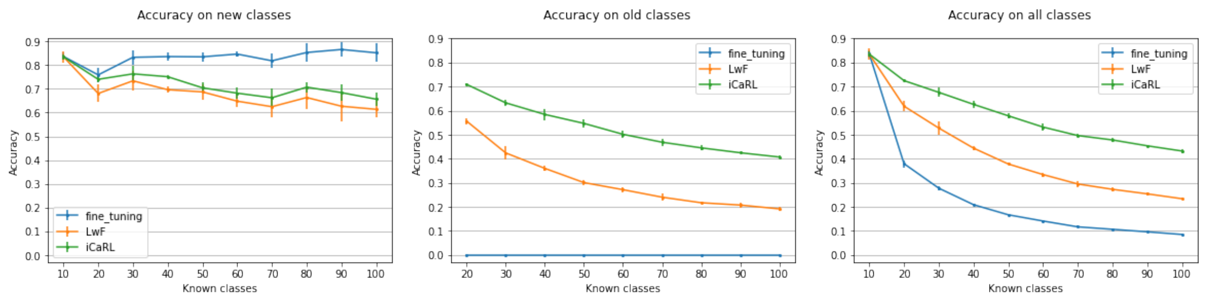

Figure 1. Experimental results of class-incremental training on CIFAR-100 performed on fine-tuning, LwF and iCarl methods: reported are multi-class accuracies across *new/old/all* classes observed up to a certain iteration. Best viewed in colors.

3. Related Works

The problem of catastrophic forgetting was described by McCloskey *et al.* [5] in the 1980s, as the phenomenon that training a neural network with new data causes it to overwrite (and thereby forget) what it has learned on previous data. The iCaRL [8] model proposed by S. Rebuffi *et al.*, was the main focus of our studies. The architecture of the model was adopted along with the incremental learning principles defined by the authors. As explained in iCaRL paper, the NME classifier is inspired by nearest class mean (NCM) by Mensink *et al.* [6]. This classifier represents each class as a prototype, that can be computed incrementally from a data stream, so it is particularly suited for an incremental learning setting. In our scenario means are computed on a chosen subset of the original data, due to the memory limitation inherent with incremental learning problem. The same principle of rehearsal explained in iCaRL was adopted: to update the model parameters for learning a representation, training data are extended with the exemplars from earlier classes. Also the same distillation loss proposed in iCaRL was adopted to prevent that information in the network deteriorates too much over time. As explained in iCaRL the same principle was recently presented by Li and Hoiem [4] under the name of Learning without Forgetting (LwF) , to incrementally train a single network for learning multiple tasks. When studying the role of the exemplars in the last section of this document, the consideration on the herding in iCaRL were inspired by the work of K. Javed and F. Shafait [2], which shows the ineffectiveness of this strategy, pointing out problems in its formulation and comparing it with more naive strategies such as random selection of exemplars. The Resnet32 model was used as back-bone model of all the experiments. This network is the one proposed by Kaiming *et al.* [1] to address the CIFAR-100 dataset, which images have a resolution of 32x32, smaller than the standard Resnet32 input resolution. Also the data augmentation proposed in this paper was adopted in our experiments. In the last section in order to reduce the inevitable class unbalance during training (exemplars are always way less than training data), the mixup augmentation (proposed by H. Zhang *et al.* in [9]) was adopted to oversample exemplars, without the overfitting repercussion brought by a naive random oversampling approach.

4. Experiments

Experiments were conducted over the three proposed methods, resulting in iCaRL being the best performing one. In addition, ablation studies on iCaRL method were performed.

Implementation Details. Fine-tuning, LwF and iCaRL methods were implemented with PyTorch. The obtained results are slightly inferior to the ones reported in the iCaRL paper. The number of epochs per iteration, for all the experiments, was 70. The initial learning rate was 2 and it was divided by 5 at given epochs $(43, 69)$. Moreover, a fixed memory size $K = 2000$ exemplars was set for the iCaRL configuration. Our source code and further data are available at the public repository `https://github.com/21ric/IL_project`.

As we can see in Figure 1, the most naive method (fine tuning) is better than the others at learning new data distributions, but it is completely forgetting previously learned experience. On the other hand, LwF and iCaRL are compromising between learning and remembering: they do not share the same learning performance with fine tuning but they are able to retain previous knowledge, even if the task gets harder as the number of learned classes increases. Finally we can see that iCaRL has the best overall performances: both having a network detached classifier and revisiting old experience are crucial in an Incremental Learning scenario. As it will be shown in the next sections also other classifiers yield comparable results to NME.

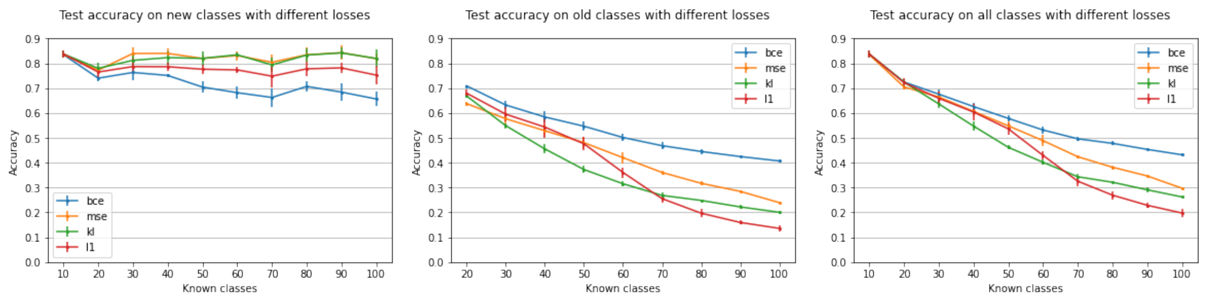

Figure 2. Experimental results of class-incremental training on CIFAR-100 performed on iCarl, using BCE as classification loss and different types of distillation losses: BCE, MSE, L1 and Kullback-Leibler are tried out. Reported are multi-class accuracies across *new/old/all* classes observed up to a certain iteration. Best viewed in colors.

4.1. Ablation studies

In this section the components of the iCaRL method were analyzed to demonstrate their impact on the final performance. Different types of distillation losses were tried out, along with different types of classifiers. All the ablation studies were performed with the fixed memory setup ($K = 2000$ exemplars in total).

First, different losses were evaluated with an experiment using three different techniques for calculating the **distillation loss** function: *L1Loss()*, *MSELoss()* and *KLDivLoss()*. All the losses were performed using the same LR value as the one suggested in iCaRL paper [8].
Both L1 and L2 loss (MSE loss) were applied among outputs obtained from the old network and the outputs of the new one, with a softmax after the last FC layer. These losses are not supposed to outperform BCE, due to the fact that both L1 and L2 are suited for *regression problems*, expecting an input value that ranges from $-\infty$ to $+\infty$. In our particular case, the input values are probabilities that range from 0 to 1. So misclassification errors are not sufficiently penalized, with respect to the heavy penalization of the BCE loss.

- *L2 loss*: it is minimizing the sum of the square of the differences between the estimated values ($g_y(x_i)$) and the target values (q_i^y):

$$\mathcal{L}_{dist} = \sum_{(x_i,y_i)} \sum_{y=1}^{s-1} (g_y(x_i) - q_i^y)^2 \tag{2}$$

  where $g_y(x_i)$ and q_i^y have the same meaning as in Eq. (1).

- *L1 loss*: It is minimizing the sum of the absolute differences between the estimated values ($g_y(x_i)$) and the target values (q_i^y):

$$\mathcal{L}_{dist} = \sum_{(x_i,y_i)} \sum_{y=1}^{s-1} |g_y(x_i) - q_i^y| \tag{3}$$

  where and $g_y(x_i)$ and q_i^y have the same meaning as in Eq. (1).

The decision to try both L1 and L2 loss was due to the fact that they look at the errors in two different ways: while L1 is simply calculating the absolute difference between two values; L2 squares this difference, resulting in a smaller value because in this particular case input and target $g_y(x_i)$ and q_i^y are probabilities.
Also, the Kullback-Leibler divergence loss was calculated among outputs obtained from the old network (with a softmax after the last FC) and outputs of the new one (with a log-softmax after the last FC). The decision of trying this particular loss was due to the fact that both L1 and L2 losses are not properly suitable with probabilistic values unlike KL. KL-loss, in fact, requires two probabilities resulting in a good alternative to MSE loss (as stated in [3]).

- The *Kullback-Leibler* divergence loss is essentially the expectation of the log difference between the probability of data in the original distribution (logits of previous network) with the approximating distribution (logits of the actual network).

$$\mathcal{L}_{dist} = \sum_{(x_i,y_i)} \sum_{y=1}^{s-1} q_i^y (\log q_i^y - \log g_y(x_i)) \tag{4}$$

  here, $g_y(x_i)$ and q_i^y have the same meaning as in Eq. (1), with the only difference that here on the input parameter ($g_y(x_i)$) is applied a log-softmax, and on the target parameter (q_i^y) is applied a softmax. In fact, both of them have to be probabilities.

Looking at the Figure 2 , we can see that using BCE as distillation loss is still the best possible choice, as it is the one

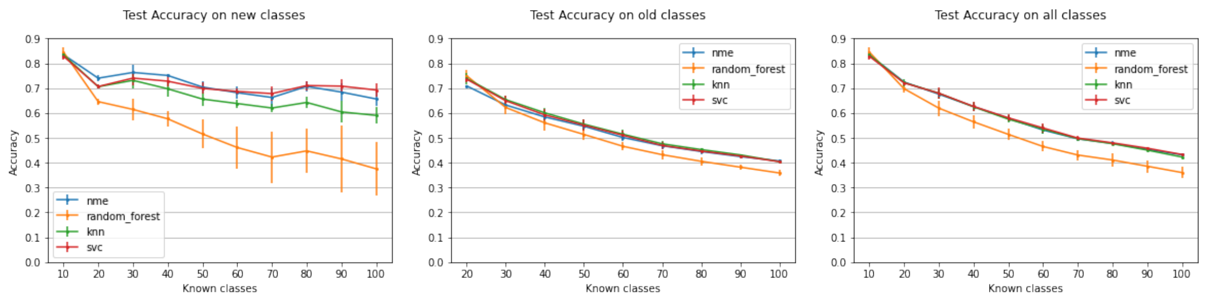

Figure 3. Experimental results of class-incremental training on CIFAR-100 performed on iCarl, using different classifiers: NME, Linear SVC, KNN and Random Forest. Reported are multi-class accuracies across *new/old/all* classes observed up to a certain iteration. Best viewed in colors.

with the highest overall accuracy. As expected, L1 and L2 loss have bad performances because their application makes sense if the outputs of the network are a real-valued function of the input images. In our scenario, L1 and L2 loss give a penalty to misclassified samples which is not severe enough. This results in a prevalence of the classification loss during training: as we can see in Figure 2, accuracy on new classes is better than the default case, meanwhile the accuracy on old classes suffers from an heavy performance drop. When considering L1, L2 losses, to avoid the problem related to the limited range of the inputs, the last softmax layer could have been removed but this can lead to extremely high value of the loss, which may bring to divergence. In the context of classification, using a sigmoid or softmax activation function in the output layer along with a cross-entropy loss, allows the model to learn faster and in a more robust way, while softmax in addition to L1 and L2 lead the loss to have many local minima, which is an unwanted scenario: the reason why the softmax function was used was related to the fact that removing it lead the network to divergence.
On the other hand, KL divergence is used to measure the divergence of the two probability distributions. The KL loss was first introduced as an alternative to L2. Looking at the results, L2 loss seem to behave better than KL, but BCE is still the best performing one as a distillation loss.

With an high degree of certainty, each of these loss configurations is not optimized, because different hyperparameters are required for each configuration and, in order to find these hyperparameters, an exhaustive tuning must be performed. With optimal parameters KL should be comparable to BCE and L1, L2 would suffer from smaller performance drops.

Other experiments were conducted using different types of **classifiers**: the classification based on the Nearest Mean of Exemplars was replaced with *Linear SVC*, *K Nearest Neighbors* ($K = 3$) and *Random Forest*. All the classifiers were trained on the features of the exemplars generated by the features extractor layers of the network. Training a classifier in this way resulted in an unbiased classifier (exemplar sets share the same cardinality).
We decided to test these three classification models because of their differences in separating the sample space:

- *KNN* is a distance-based algorithm, it draws boundaries that are jagged regions of the space and assigns the labels according to the majority voting of the K nearest neighbors of the sample.

- *Linear SVC* is a classification algorithm which draws linear boundaries dividing the space.

- *Random Forest* is suited for classification tasks where the feature space is separable by axes orthogonal to the dimensions.

Figure 3 shows that KNN, SVC and NME applied on the features of the exemplars work better than random forest. KNN, being a similar technique w.r.t. NME, performs worse maybe because classifying very high dimensional points makes the model more sensitive to noise, while the averaging effect of NME helps in the stability of the classifier.
The random forest classifier is clearly not good for our task, leading to the conclusion that our points can't be classified with boundaries which are orthogonal. On the other hand, Linear SVC gives good accuracy scores, leading to the conclusion that a linear separation fits pretty well on our data.

5. Our Considerations

Authors of iCaRL made three contributions: *herding algorithm for exemplars selection*, the use of *NME classifier* and a particular *distillation loss*. Particularly, we discuss and analyze the first contribution and, starting from that, we make some considerations regarding the use of exemplars.

5.1. Random exemplars selection

iCaRL uses the herding algorithm for selecting exemplars that approximate the true class mean. We did some experiments on the different random splits, both using herding algorithm and without using it: it was replaced with a random exemplars selection. We noticed that there was no significant difference in the accuracy testing iCaRL model with or without herding. Also, K. Javed and F. Shafait [2] did the same experiment, showing no big differences among the two methods. This could be explained understanding how the features extractor layers work: at each training phase the feature map of the images forwarded to the net changes along with the iterations. This leads to the phenomenon that exemplar images chosen to give a good approximation of the class mean at increment i may not to give a good approximation at increment $i + 1$, $i + 2$, etc. if the feature extraction is changing significantly.

Based on this assumption, in the following section we will try to focus more on other aspects regarding exemplar sets.

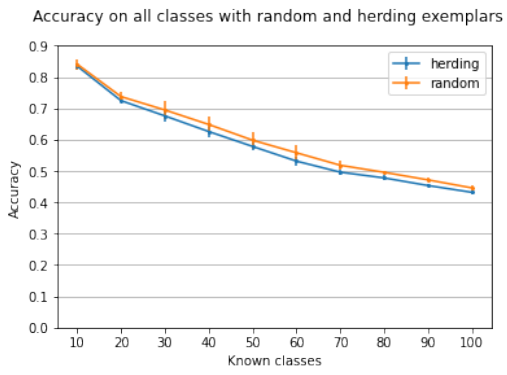

Figure 4. Experimental results of class-incremental learning on CIFAR100 performed on iCarl on three different splits, using herding selection and random selection for exemplars. Best viewed in colors.

The plot in Figure 4 shows a little improvement in using random selection of exemplars: this could be explained taking into account the fact that these experiments are performed on just three different splits, so they are very dependent of the particular data that are forwarded to the net. After all, the performances are quite the same.

5.2. Changing memory threshold

Further experiments involving iCaRL method were performed, with a variation of the memory threshold. In fact, parameter K (memory size) was changed with different values in order to understand if reductions or increases of the memory used to store the exemplars could improve the performances of the model.

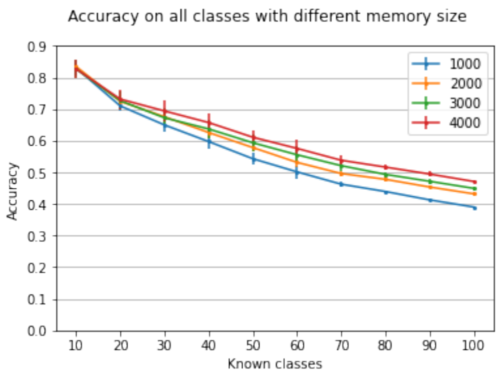

Figure 5. Experimental results of class-incremental learning on CIFAR100 performed on iCarl on three different splits, using K = [1000, 2000, 3000, 4000] exemplars. The plot shows that as the memory size increases, the final accuracy tends to increase, even if with slighter improvements. Best viewed in colors.

Figure 5 shows the increase in accuracy as the memory size increases. $K = 4000$ exemplars is the best configuration, as we expected, but we can also notice that there is not an important difference in performances between $K = 3000$ and $K = 4000$ exemplars. Gaining accuracy at the expenses of doubling the memory size is not a good decision. An observation that could be made looking at this plot is that giving more memory space to the exemplars leads to better results, while giving more memory space to previously learned classes do not lead to better performances (as shown in Section 5.3).

5.3. Configuration of exemplars

Another experiment was performed in order to figure out how much the number of exemplars per class influenced the performances. A fixed number of exemplars was chosen to be reserved for each class without considering the memory limit of $K = 2000$ exemplars, then the accuracy on each of the old 10-classes batch was computed at the end of the whole training. This experiment was performed with 20,40 and 60 fixed exemplars per class and the results were compared with the ones obtained with iCaRL. A bar plot was printed such that the final accuracies on each old group could be compared.

As expected, the bigger the number of exemplars per class is, the better the model is able to remember each group. Also, the absence of a trend in the plot of each group shows that *iCaRL has no intrinsic bias* towards or against classes that it encounters early or late during learning. It is possible to observe that, regardless the size of available memory, the accuracy on the older group of classes is not necessarily the smallest one, in fact, there are youngest groups of classes that have a lower final accuracy.

Another important fact was shown: with iCaRL, the number of exemplars per class varies with the total number of learned classes in order to equally distribute the memory resources, so the first group starts with 200 exemplars per class and ends up with just 20 exemplars per class. If we compare the results of iCaRL with the results of 40 fixed exemplars per class, we can notice how they are very similar to each other. If we look at the final accuracy on the older group obtained with iCaRL and to the one obtained by maintaining 40 exemplars per class, we can see that they are almost the same. With iCaRL, the first group starts with 200 exemplars, then decreases in cardinality to 100 exemplars, 66, 50 and so on; in the second case the first group has, for at least the first iterations, a much smaller number of exemplars. The difference between the initial number of exemplars in these two cases (200 vs 40) is 160 exemplars in favour of iCaRL. It seems that not even starting with an high number of exemplars is relevant.

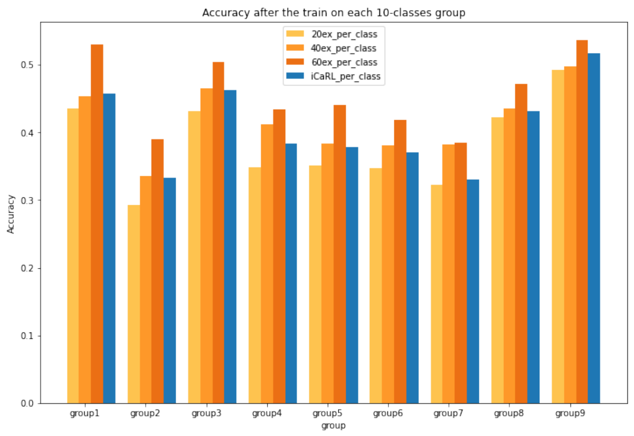

Figure 6. Experimental results of class-incremental learning on CIFAR100 performed on iCarl and modified versions of it, storing respectively 20, 40 and 60 exemplars per class since the beginning of the training. Experiments are performed on three different splits. Best viewed in colors.

5.4. Undersampling the training set

As we can see from previous sections, iCaRL shows a bias towards new classes. This bias is not caused by the classifier but by the feature extractor. In fact, the training dataset is mostly made of samples belonging to new classes (this is not avoidable in a class incremental setup, in which all previous data cannot be retained). Increasing the number of exemplars brings to an overall improvement because this unbalance is mitigated, but by doing this, the memory threshold is exceeded. In this section we try to completely remove this unbalance by proportionally undersampling the training data (all classes, new and old, have the same cardinality). We call it $iCaRL_{us}$ for simplicity (us stands for undersample).

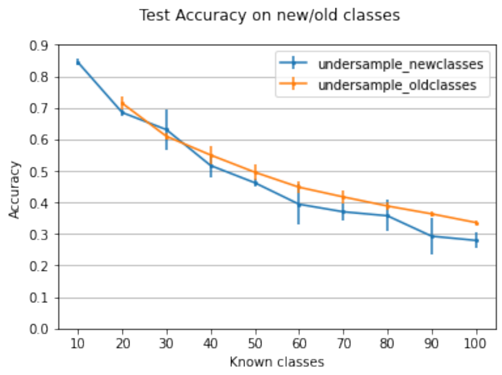

Figure 7. Experimental results of class-incremental learning on CIFAR100 performed on $iCaRL_{us}$ on three different splits. Accuracies on the old and the new classes are plotted in a way that the blue line corresponds to $iCaRL_{us}$ performances on new classes and orange line corresponds to $iCaRL_{us}$ old classes. Best viewed in colors.

We can see from Figure 7 a reduction of the aforementioned bias along a general drop in performance as the number of iteration increases. This happen due to the fact that the last classes are modeled with way less samples, compared to the default $iCaRL$. In general, tossing this amount of data is not a good practice, since theoretically classes are best modeled when mostly all data available is used. In the next section we propose a mirror approach to reduce the bias, oversampling the exemplars instead of undersampling the training data.

6. Our variation

Based on the observations made in the above section and on the considerations made in iCaRL paper [8] regarding hybrid2 (the one which uses the exemplars for classification, but does not use the distillation loss during training), the results are not so different. Instead, LwF uses distillation but no exemplars at all, and we can see the big improvements of iCaRL w.r.t. LwF. The main explanation of why iCaRL works better than LwF is not for herding (as we already explained), not for NME (K. Javed and F. Shafait [2] showed that you can learn an unbiased classifier), but for the *rehearsal of previous learned experience*, which helps the network to remember previous data distribution over time. Using a combination of distillation loss and exemplars is a good way to address the catastrophic forgetting problem.

Different methods for compensating class unbalance are: oversampling/undersampling [7] the training data, introducing class weights or class aware loss functions, or thresholding (which solves the classification bias problem). Inspired by the work of [9], we used *mixup augmentation* on exemplars, in order to improve accuracy on old classes. To see the effectiveness of this method we compared it with a more naive version of *oversampling*.

Oversampling. Exemplars are added 4 times to the dataset (6000 points more).

Mixup. 5850 points are added to the dataset. Each point is a weighted linear combination ($w = 0.6$) of two randomly selected exemplars. This augmentation is done when a batch of images is going to be loaded to the network. From the exemplars in that batch we are creating 150 additional points (the number of image batches, at each epoch, is 39).

$$\begin{aligned} \tilde{x} &= wx_i + (1-w)x_j \\ \tilde{y} &= wy_i + (1-w)y_j \end{aligned} \tag{5}$$

Here, x_i and x_j are two random exemplars, generating the new $\tilde{x}$ sample; while y_i and y_j are the respective labels, generating the new label $\tilde{y}$.
The new label is not used in classification, while in distillation we are using the previous output of this newly made sample, instead of combining the previous outputs of the components (the output of a linear combination is not supposed to be the combination of the output of the components, differently from what happens when using a label as a target).

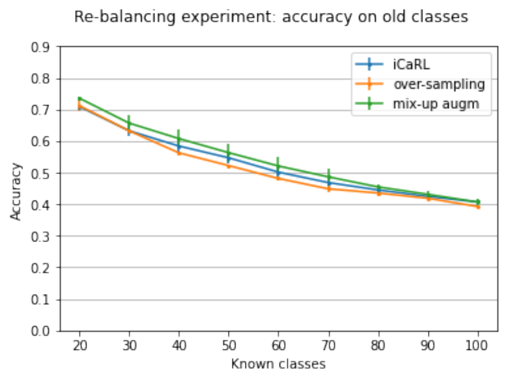

Figure 8. Experimental results of class-incremental learning on CIFAR100 performed on standard iCaRL, iCaRL with oversampling the exemplars and iCaRL with mixup augmentation performed on the exemplars. Accuracies on the old classes are plotted. Best viewed in colors.

As we can see from Figure 8, the oversampling approach did not lead to an improvement of the performances on the old classes even if it allows the model to train with more exemplars. This behaviour derives from the fact that oversampling causes overfitting because of the re-usage of the same exemplars.
Data augmentation is known to be an efficient method to overcome overfitting because it increases both the number of training images, starting from the original ones, and the variance explained by their distribution at the same time. Mix up is a particular kind of data augmentation: it allows the model to train with a bigger number of exemplars without leading to overfitting because, differently from oversampling, it does not use exact copies of exemplars.
In fact we can see that the mix-up augmentation provided a higher accuracy on the old classes w.r.t to both oversampling ad iCaRL.

Also, reported are average accuracies on new/old/all classes performed by iCaRL, iCaRL with oversampling, iCaRL with mixup augmentation: we can notice that, on the overall accuracy, $iCaRL_{mixup}$ is still the best performing one.

Method	New	Old	All
$iCaRL$	71.8	52.4	58.3
$iCaRL_{over}$	76.3	52.6	59.1
$iCaRL_{mixup}$	70.7	**54.0**	**59.2**

Table 1. Average multi-class accuracy on iCIFAR-100 for different modifications of iCaRL, on new/old/all classes.

7. Conclusion

We considered mixup, a data augmentation technique which generates points that are a linear combination of the exemplars data. We introduced it as a possible strategy to mitigate class unbalance, which is one of the main weakness of iCaRL. Due to the nature of the IL setting, this class unbalance will always be present and can only be reduced. In fact, even adding a big amount of samples, the class unbalance was only slightly reduced. Oversampling could have been made proportionally but this would have slowed down the training significantly. In addition samples are generated from gradually less images and the improvements start to disappear as the number of iterations increases. We think that a better approach to storing exemplars could be building a generative model capable to reproduce previous data distribution when needed. This could overcome memory limitations, erasing the need of storing any data.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[2] K. Javed and F. Shafait. Revisiting distillation and incremental classifier learning, 2019.

[3] O. Kosheleva and V. Kreinovich. Why deep learning methods use kl divergence instead of least squares: A possible pedagogical explanation, 2017.

[4] Z. Li and D. Hoiem. Learning without forgetting, 2016.

[5] M. McCloskey and N. J. Cohen. *Catastrophic interference in connectionist networks: The sequential learning problem.* 1989.

[6] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In European Conference on Computer Vision (ECCV), 2012.

[7] A. More. Survey of resampling techniques for improving classification performance in unbalanced datasets, 2016.

[8] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning, 2017.

[9] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. Mixup: Beyond empirical risk minimization, 2018.