Incremental Learning in Image Classification

Edoardo Lardizzone
s278561@studenti.polito.it

Alessandro Nicolini
s271226@studenti.polito.it

Alessandro Palladini
s277450@studenti.polito.it

Abstract

Extending the knowledge of a model is an open problem on the road to artificial intelligence. The scope of this project is to fully understand the issues of the Incremental Learning approaches, and consequently trying to manage them, including the catastrophic forgetting, by implementing and analyzing three different existing baselines. We further propose two variations of our own that could overtake some of the problems of the techniques used.

1. Introduction

Incremental Learning is a Machine Learning paradigm that allows an existing model to continuously receives new unseen data which it has to be trained on, and be able to being adjusted in order to predict both old and new classes. Training a system on a dataset that includes all the classes seen can be unfeasible in many real-world scenarios *e.g* most of the time the data comes in a stream or the data on which the model was created are not available anymore. A central issue in incremental learning is the *catastrophic forgetting*: when adding new data the model forgets features of old classes and it is not able to recognize pre-learned classes anymore.
To overtake this issue a lot of papers and studies has been made: in this project we develop three of these approaches and some modifications of these ones and finally we propose a solution that we think could improve the results.

2. Approach

In this section we describe the implementation details of our work, the architecture of the deep neural network and the dataset that is partitioned during training for incremental learning.

2.1. Dataset

The dataset we use is the CIFAR-100 dataset available at https://www.cs.toronto.edu/~kriz/cifar.html. It is composed of 60000 thousand images of 32x32 pixels belonging to 100 different classes (600 for every class), 50000 are used for the training and the remaining ones for the testing. We implement a class that is able to create the dataset by receiving a random seed in order to randomly chose the classes added at each new task.

2.2. Network

For feature extraction and classification we use a ResNet-32, a deep convolutional neural network. The novelty of residual network architectures [3] is the presence of skip connections which help in addressing the problem of vanishing gradients, allowing the construction of deeper networks. We consider the network up to the last layer as a *feature extractor*, $\varphi : \chi \to \mathbb{R}^d$. The features are extracted with an output of 64x1 which is given to the last fully connected layer, initially used as a classifier. The fully connected is made of 100 outputs from the beginning even if it goes against the fact that the number of classes is not known apriori. We know that it's not general, but it's easier to handle and it's what iCaRL did. The weights are adjusted by gradient descent.

2.3. Workflow

To simulate the incremental learning flow we split the training set in 10 batches containing the images of 10 classes at a time, and tune the network on the images of these classes. The incremental training phase is done repeatedly using the same network, that receive each time only the current batch of images. The training images are augmented by applying a random horizontal flip with a 0.5 probability and a random crop of 32x32 with a padding of 4 pixels on the board of the image, this is done in order to improve generalization capability of the model. All the work were made using PyTorch framework and run on Google Colaboratory, which gives a tot of free GPU time with very limited capability, making some experiments impossible to be tried. The code is available at https://github.com/alessandronicolini/IncrementalLearning.git.

2.4. Settings

The setting used for the baselines is the same adopted in the iCaRL paper [2]: a batch size of 128, an initial learn-

ing rate equal to 2 which is multiplied by 0.2 at epochs 49 and 63 and a number of epochs of 70. We use BCEWithLogits both as classification and distillation loss and a SGD optimizer. In order to mitigate the sources of randomness each baseline experiment is ran 3 times with different random seeds, for each new task, represented as the number of classes seen up to that task, we report an error bar with the average of the test accuracy measured at the end of each task as mean value, and their standard deviations as error bar. All the other experiments apart from the three baselines are executed only once due to lack of resources, but to guarantee that they are comparable the seed is fixed.

3. Baselines

The three initial techniques we develop are: simple fine-tuning, Learning without Forgetting and iCaRL.

3.1. Joint training

We run this experiment that does not respect the typical constraints of the incremental learning setting. Indeed, at each new task the model is trained on all the data provided to the net up to that point. The only purpose of this experiment is to estimate the upper bound limit of measured performances given the number of seen classes. For the reasons previously explained plus the only representative purpose of this experiment it is executed only once.

3.2. Fine Tuning

Fine-tuning is done because it highlights the catastrophic forgetting effect. The model is randomly initialized and it is trained on the first 5000 images batch of 10 random classes for 70 epochs and then tested on the corresponding test images of these 10 classes, then at time 2 the second batch is imported and the model previously trained is used as starting point, after this training the test is made on both the testing images from the first and second batch. The forgetting

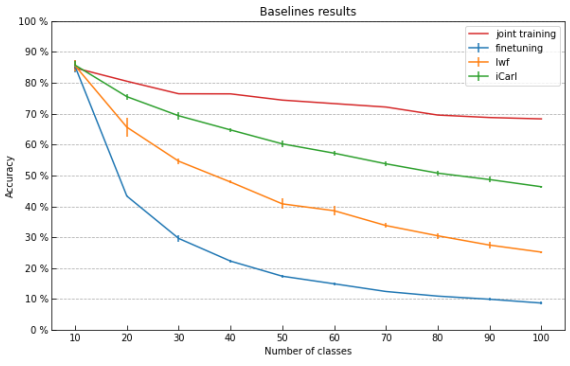


Figure 1: iCaRL multi-class accuracy for each variation

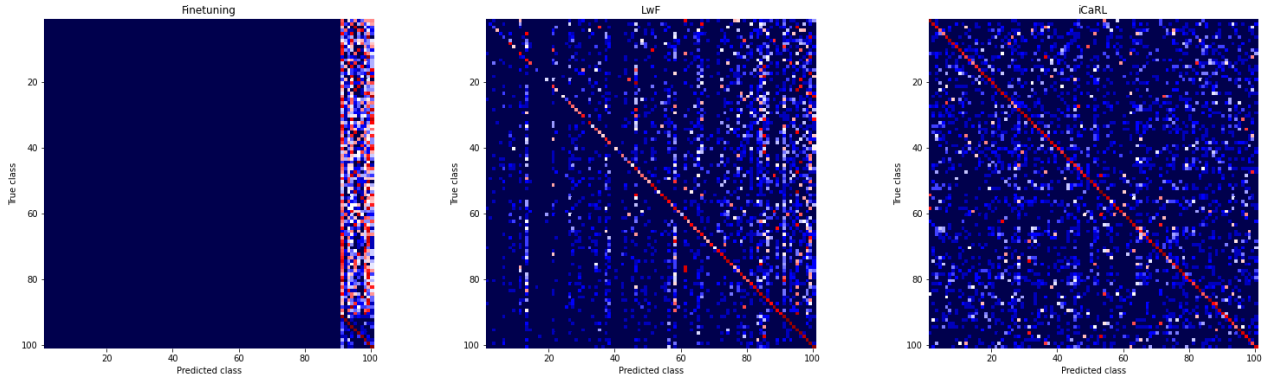
occurs because when training on new images the weight are adjusted only to better predict the second 10 classes and no information on the first 10 is given to the model. Looking at the accuracy results but especially at the confusion matrix at figure 2a it is obvious that the model is no longer able to correctly classify the classes of previous batches because no technique is used to prevent the issue.

3.3. Learning Without Forgetting

As a first step to mitigate the effects of catastrophic forgetting, we implement Learning Without Forgetting [1]. The novelty of this approach is the addition of the distillation loss. Knowledge distillation consists in transferring knowledge from the previous network to the current one. Hence, the loss used in training the net is a combination of a classification loss and a distillation loss. At each step the previous task model is saved, the classification loss is computed on the outputs which goes from the previous task last output neuron up to the end (e.g. task1 0-99, task2 10-99, task3 20-99 and so on) with the respective class label used as ground truth, while the distillation loss is computed on the previous tasks output neurons using the output of the old net feed with the new class data as ground truth. In this way the network is still able to correctly predict images from previous classes because the parameters are optimized not only on classifying the current data but also on maintaining some characteristics developed in previous training. For this particular implementation we did not use the setting used in the paper but the ones used in iCaRL in order to have a fair comparison between the three techniques. The classification and distillation loss are the same (BCE with logits), looking at figure 1 we can see that compared to the simple fine-tuning experiment we have consistent improvements from the second to the last batch finishing with an accuracy on the last set of classes that is more than doubled. Moreover, from the confusion matrix we notice that the model is still able to correctly predict some of the old images which does not happen for the fine tuned model. This can be seen looking at figure 2b where, at the tenth task, although it still tends to classify old images as belonging to new classes (red points in the top right corner), the diagonal of the matrix is well populated.

3.4. iCaRL

The last implemented baseline is iCaRL [2]. This approach uses the distillation loss of learning without forgetting but it adds two new improvements: first of all to prevent the lack of images from previous classes it creates a set of so called exemplars that are images whose purpose is to represent the distribution of the class they belong to. The total number of exemplars is fixed at K=2000, so at every step m=K/n exemplars for every class are taken (where n is the number of classes the model has received so far) and at the same time



(a) fine-tuning (b) LWF (c) iCaRL

Figure 2: Confusion matrices of baseline experiments on the incremental version of CIFAR-100. While fine-tuning is able to correctly classify only the last task classes, the Learning without Forgetting also predict old classes, still having better performances on the latest tasks, iCaRL predictions are distributed rather evenly among the classes.

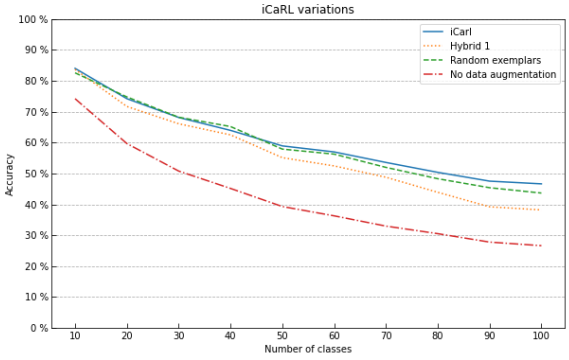


Figure 3: iCaRL multi-class accuracy for each variation

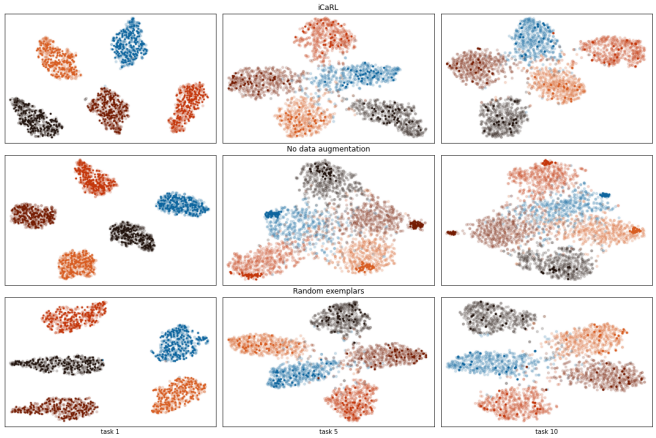


Figure 4: t-SNE results for basic iCaRL, Random exemplars and No data augmentation, at task 1, 5 and 10. Five class of the first task are chosen, the light color points show the original data as reference of the real data distribution, the deep-color points are exemplars.

the exceeding number of exemplar from previous batches is deleted. In this way every class has the same number of images in the exemplar set.

The exemplar set for a new class is made with the *herding* technique, which selects the images ordering them according their importance: the idea is to add at the exemplar set, one at a time, the images that minimizes the L2 distance between the class mean features and the mean features of the already collected exemplars (avoiding duplicates) in an ordered fashion, until the correct size is reached. This ensures that deleting the last exemplars of the old classes, the most important ones still remains.

The second technique developed to further improve the results is the adoption of the so called *nearest mean of exemplar* classifier (NME): the output of the feature extractor is passed to this classifier which measures the L2 distance between the input image features and all the exemplar sets mean features, predicting the label of the exemplar set which minimizes this distance, the exemplar sets mean features are computed and stored just before the test phase of each task, which is repeated at each task. To visualize the improvements of this classifier we also implemented the *Hybrid1* setting, where the fully connected layer is used as classifier, while keep using the selected exemplars for representation learning.

Looking at accuracy results the improvement is obvious: in fact the model is not only able to remember old features by

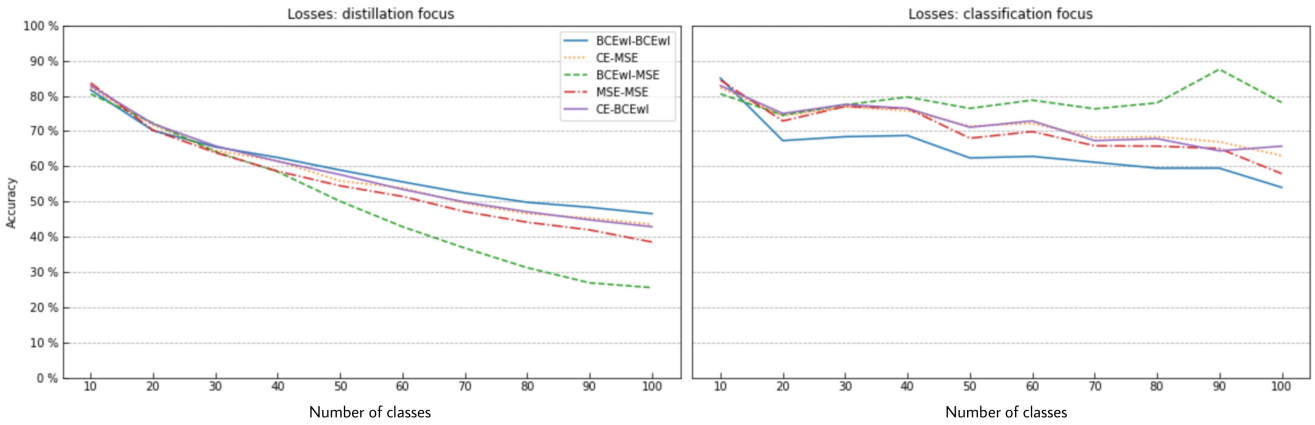


Figure 5: Accuracies for different combinations of classification and distillation loss tested on incremental test set (left) and actual test set (right)

mean of a distillation loss but it also has some data from old classes that are used at training time (the training set is a concatenation of the current batch and the exemplars set), resulting in a final accuracy that is almost the double of the LwF one.

To better understand the behavior of this method, we made some variations to the baseline iCaRL as it was proposed, we tried to see what happens when the last fully connected is used as classifier (*Hybrid1*), when no data augmentation is made on the training set in the training phase and when exemplars are randomly chosen. Looking at figure 3 is quite clear that the basic iCaRL setting is the best one, particularly remarkable is the multi-class accuracy drop without using any data augmentation on the training set. In implementing iCaRL we did not perform any validation during training to select the best model of the 70 epochs, and data augmentation seems to have a major role in guarantee a good generalization capability of the model even if not validated. This is even more clear in figure 4, which highlights how exemplars of the first task suits the corresponding data distribution which change task after task. While the Random exemplar experiments yields similar results to the basic iCaRL, in the no data augmentation experiments the exemplars tend to collapse in the same region, they are no longer representative of the class distribution they should approximate. Looking at figures 1 and 2c, it is obvious that the presence of the exemplars and the use of a NME classifier are a big improvement especially in the last tasks where the model is still predicting the 46% of the whole test set, while by the confusion matrix it is evident that the model is not over-predicting the last classes but these predictions are instead evenly distributed for the 100 classes.

4. Ablation Studies

For the following studies we were not able to have 3 run using 3 different seed for every method so we did the comparison only on one seed to be fair, even if not fully reliable (due to training shuffle and to the fact that only one set of batches is considered) the aim of this studies is to show the various changing in these different settings.

4.1. Different Losses

We use three different sets of losses for classification and distillation comparing them to the BCE-BCE setting of iCaRL. For every set of losses, other than the accuracy on the whole test set to check which set has the better remembrance of old classes, we also check the accuracy on the test considering only the images of the current batch to see which set is the more adapt in learning new classes.

4.1.1 CE-MSE

As a first try we change the classification loss to a Cross Entropy loss defined as:

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{1}$$

where M is the total number of classes, $y_{o,c}$ is a binary value which takes 1 if the prediction of observation o is correct 0 otherwise and $p_{o,c}$ is the probability of the observation o to be of label c. The distillation loss is instead a Mean Squared Error loss also called L2:

$$\sum_{o=1}^{N} (y_o - y_{o,t})^2 \tag{2}$$

where y_o is the output of the fully connected layer for an observed value o after a softmax operation and $y_{o,t}$ is the

ground truth label expressed as a vector with 1 in the correct position and 0 elsewhere. We notice looking at figure 6 that this set is the one that gets nearer to the iCaRL results, the problem of this configuration is in learning new classes: going on with the batches the accuracy on the singular test set decreases constantly.

4.1.2 BCE-MSE

Another set used for comparisons is BCE-MSE, which is similar to the first one but with a Binary Cross Entropy as classification loss. In this case the ability to remember old classes has decreased a lot, resulting in the worst combination for the incremental learning approach, but on the other side the learning of new classes has highly improved being constant during the 10 steps.

4.1.3 MSE-MSE

This particular combination was the one that gave the worst results when testing only on the actual batch, having an accuracy of nearly 57%. This probably explain the outcomes on the whole test set that were quickly dropping while going on with the batches.

4.1.4 CE-BCE

Finally to have a set where BCE was used as a distillation loss like in iCaRL we used CE as classification. The particularity of this combination is the presence of an high overfitting (accuracies near to 1 on the training set) that did not reflect on the test set even if it almost reaches the iCaRL results.

4.1.5 Conclusions

While testing four different combinations we understand that the combination used by iCaRL is the best at distilling knowledge, even if confronted only on one seed, and it could maybe be overcome by using losses that we do not know.

4.2. Different Classifiers

iCaRL uses a so-called NME (variant of NCM) as a classifier, it stands for Nearest Mean of Exemplars and the functioning is the same of NCM: at every step the mean value of the features extracted by the net is computed for every class (while for the old classes it is computed on the exemplars) and when classifying an image, the output of the feature extractor is taken and confronted with these means, the output of the classifier is the class whose exemplar mean is the nearest to the features of the image by computing a L2 distance. To make a confront on which classifier is the best we tried three different algorithms, K-NN, MLP and SVM.

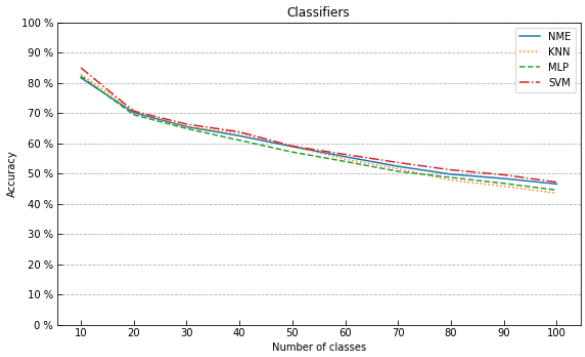


Figure 6: Accuracies for different classifier on 10 classes per time

4.2.1 Implementation

For reason of GPU availability and time we did not perform a complete hyperparameter search but instead we only tried few values and taken the ones which gave the best accuracy values on the test set. The training of the models is made only on the exemplars in order to have balanced dataset and not incur in misclassification due to unbalanced data.

4.2.2 K-NN

K-NN is a geometrical classifier algorithm (it can also be used for other machine learning problems but we use it as a classifier). It acts on a set of features, that in our case is the 64x1 vector, the output of the feature extractor. The features of the training images are used to train the K-NN classifier: for every point in the space the k nearest training point are taken, the label which is more present amongst these *k* points is the label which this point will be classified as. We tried different values for k (obviously odd values) obtaining that for our specific seed k=5 was the best number of point to take in consideration.

4.2.3 MLP

Multi Layer Perceptron is an artificial Neural Network used to classify data that are not linearly separable. It is composed by at least three layers: one input layer, one or more hidden layer and one output layer. The particularity of this networks is that every node share a weight $w_{i,j}$ with every other node of the adjacent layers. The activation function is a non linear function and the weights are updated by gradient descent using L2 as loss. For our case we used a constant learning rate of 0.001 using an *adam* optimizer with 200 epochs of training and only one hidden layer, using as activation for the hidden layer the ReLu function.

4.2.4 SVM

Support Vector Machine is a family of geometrical algorithm useful for classification tasks. The features are placed in a plane,in this case a plane of 64 dimensions, and the algorithms tries to separate them by means of hyperplanes. The typical approach when classifying on multiple classes is the *one vs all* that is, one class is separated from all the others and the procedure is repeated for every class; typically data are not fully separable and so the approach is soft: some data are let to be on the wrong side of the hyperplane that is constructed by maximizing the distances between the points and the plane. When data are not linearly separable it is used the so called *kernel trick* by which data are mapped in new feature spaces where they are more easily separable. For our case we had a quick hyperparameters optimization which led to the use of a rbf kernel (*radial basis function*) and C=1 (C is the parameter responsible for the cost of misclassification, i.e., points that are on the wrong side of the hyperplane)

4.2.5 Conclusions

The result are not fully reliable due to no changes of the random seeds, but it is clear that all the algorithms have good performances on the set of features created by the ResNet, in particular it seems that the SVC was even better than the NME used by iCaRL but as aforementioned this could not lead to the conclusion of SVM being the best classifier for this method.

5. Beyond the baselines-Our proposals

We think that the best thing to improve in this workflow is the quality of the data used to remember old classes, the so called *exemplars*. As a matter of fact they are chosen via herding: at a specific task m images are selected as the most representing of their class. The problem for us is that they are the best for the current step but going on with the steps are they still the best? This concept is better highlighted in figure 4. Obviously we could every time get new exemplars not only for the current classes but also for the classes already seen by computing again the mean of every class but this would go against the fact that training images are not available anymore. We tried different approaches in order to upgrade the quality of the exemplars.

5.1. Collage exemplars

The first and easiest method we tried in order to make exemplar artificially is to combine four images in a collage which will constitute a unique exemplar sample, as reported in figure 8. The intuition behind this proposal is try to make images which could give more information about the class they represent letting the network extracting more features. In this way we could have chance to enhance the performances on the current task but we do not address the problem of the class distribution changing over tasks (9), which instead would need a way to update the old exemplars in order to better fit the current class distribution without any other old data, however it is worth verifying the goodness of the collage idea. In order to have comparable results we fixed the seed at the same value for each experiment. Apart from the iCaRL baselines results, simply identified as *iCaRL* in figure 7, we also recorded the results related to three different collage experiments:

- **Resized**: when it comes to chose new exemplars, $4 \cdot m$ images for each new class are randomly selected, with the constrain that no image is repeated among the collages, to maximize generalization. Then the collage making process starts: the chosen images are sequentially grouped four by four and each group is exploited to make a single collage exemplar which is resized to have an image size of 32x32. The aim is to generate 2000 different collage exemplars 32x32, within the memory constraint, but this is not possible in the first task because we have 5000 images, so choosing 4 different images for each collage results in creating less then 2000 collage exemplars but only 1250. From the second task on is possible to collect the maximum number of possible collage exemplars.

- **Random crop**: with the *Resized* experiment making the resize transformation the original resolution of the images is reduced, in order to see if this could affect the experiment we tried to make the collage exemplars combining random 16x16 patches of the selected images.

- **Center crop**: same procedure of *Random crop* experiment but now the patches are not randomly chosen but the central area of the image is systematically selected.

Looking at the multiclass accuracy for each experiment in figure 7 we can see how the *Random crop* one yields the best results among the experiments. Selecting random patch of the images probabably helps the net in generalizing better, but this happens only for the current task classes, indeed a common trend between the experiments is an important drop with respect to the baseline *iCaRL*. We tried to explain this behavior looking at the t-SNE dimensionality reduction of the feature space, from 64 to 2 dimensions (figure 9) made for 5 classes of the first task, looking at their evolution for task 1, task 5 and task 10, where the net should be able to classify respectively, 10, 50 and 100 classes: the lighter points represent all the data we have for each class, as an approximation of the class distribution, while the deep color points are the exemplars. From figure 9 we can see how since the first task, the exemplars of the three collage

experiments tend to concentrate around a restricted area instead of being sparse in the whole class distribution, such as in the baseline *iCaRL* case, this trend is less marked in the *Random crop* experiments. This results in having exemplars which are not well representative of the class distribution, and this initial error sums to the catastrophic forgetting phenomenon we tried to tackle so far, encountered over the tasks: by the end of the training using these kind of exemplars means completely misclassify the old classes, being them completely unrepresentative of the class distribution (see how the exemplars in task 10 are completely out of the approximated class distribution). In our opinion, even though the exemplars were created by grouping four different images which should have contained all the information to recognize a given class, the model learns features that not particularly suit to the original class distribution. The collage exemplar setting is definitely not a good choice. Looking at figure 7 we can notice how despite the fact the patches are randomly chosen, we are likely to select parts of the image that focus on the subject, due to the fact the subjects are a major part of the image. Following this line one could think a way to completely remove just the background of the initially selected images, using them as exemplars, to see if this brings improvements, but we did not have enough time to explore this possibility. The other direction we explored is instead trying to generate synthetic exemplars, which is explained in the following section.

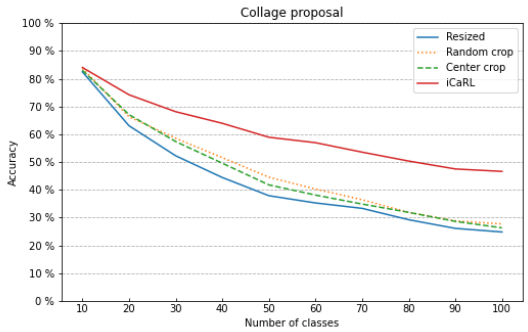


Figure 7: Multi class accuracy for each experiment made with collage exemplars.

Figure 8: Three collage exemplars created in the Random crop experiment, for the class poppy.

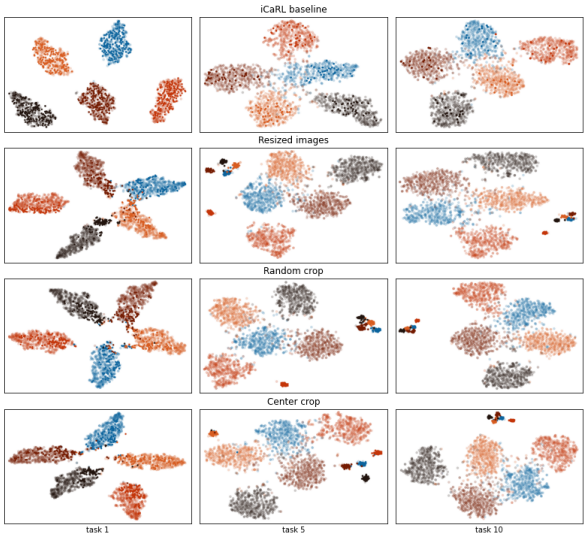


Figure 9: t-SNE reduction to visualize both data and exemplars in the 2 dimensional space for each experiment.

5.2. Synthetic exemplars

Inspired by a paper ([4]), we discovered and studied the *DeepInversion* approach. Our aim is to synthesize images from the image distribution used to train a deep neural network: this is done by inverting the trained network, called teacher network, to synthesize class-conditional input images starting either from random noise or from original images extracted from our training set.
Formally, given a trained network it is possible to obtain synthetic images ($\hat{x} \in \hat{\chi}$) containing similar low and high-level features as x ($x \in \chi$), the real set of images. The initial workflow is the one of Deep Dreaming which is used to create images given a trained network, beginning from random noise by optimizing:

$$\min_{\hat{x}} L(\hat{x}, y) + R(\hat{x}) \tag{3}$$

where $L(\hat{x}, y)$ is a loss computed between the output of the fed image and the label we want to create an image of (e.g a Cross Entropy Loss) and $R(\hat{x})$ is an image regularization term useful to create images that are not totally unrealistic defined as:

$$R_{DD}(\hat{x}) = \alpha_{TV} R_{TV}(\hat{x}) + \alpha_{l_2} R_{l_2}(\hat{x}) \tag{4}$$

in this way it penalizes the total variance and the loss of the image $\hat{x}$. In this case the optimizer will not act on the model parameters but instead will modify the inputs of the net in order to minimize this sum. Deep Inversion extends this concept by changing the regularization term in order to further decrease the dissimilarities between the feature statistics of the created images and the synthesized ones,

minimizing the distance between feature map statistics for x and $\hat{x}$. Assuming the feature statistics follow the Gaussian distribution across batches, the feature distribution regularization term can be formulated as:

$$R_{feature}(\hat{x}) = \sum_l ||\mu_l(\hat{x}) - E(\mu_l(x)|\chi)||_2 + \sum_l ||\sigma_l^2(\hat{x}) - E(\sigma_l^2(x)|\chi)||_2 \quad (5)$$

where μ_l and σ_l^2 are respectively the mean and the variance of the batch-wise mean and variance terms used for normalization in the l^{th} Batch Normalization layer, this computation is easy to made since these two values are stored in these layers and are easily obtainable. Hence, the BN layer capturing the channel-mean and variance during training allows to estimate the expectation in eq. 5:

$$\begin{aligned} E(\mu_l(x)|\chi) &= \text{BN}_l(\text{running-mean}) \\ E(\sigma_l^2(x)|\chi) &= \text{BN}_l(\text{running-variance}). \end{aligned} \quad (6)$$

Putting the blocks together, the image regularization term becomes:

$$R(\hat{x}) = R_{DD}(\hat{x}) + \alpha_F R_{feature}(\hat{x}) \quad (7)$$

where α_F is a weighting factor.
Obviously not only the coherence with the class is important for exemplars, we also have to consider the diversity of these images: we should not create similar images many times because, otherwise we would have redundant images repeated in the exemplars set. In order to obtain diversity among the synthetic images we go on and implement the *Adaptive DeepInversion* which enable the use of a student network (used alongside the teacher network that is used to synthesize images): the authors of the paper introduce the student mainly to mimic the prediction and feature extraction of the teacher network by distillation knowledge, but we will not exploit this functionality but rather, when creating the images, the student allows the algorithm to try to enhance a teacher/student disagreement, that implies diversity. This competition between the two networks is weighted by an α_A factor. Putting all the blocks together, $R(\hat{x})$ in eq. 3 becomes:

$$R(\hat{x}) = R_{DD}(\hat{x}) + \alpha_F R_{feature}(\hat{x}) + \alpha_A R_A(\hat{x}) \quad (8)$$

where

$$R_A(\hat{x}) = 1 - \textbf{JS}(p_T(\hat{x}), p_S(\hat{x}) \quad (9)$$

with

$$\textbf{JS}(p_T(\hat{x}), p_S(\hat{x})) = \frac{1}{2}(\textbf{KL}(p_T(\hat{x}), M) + \textbf{KL}(p_S(\hat{x}), M)) \quad (10)$$

where $\textbf{KL}(\cdot)$ is the *Kullback-Leiter Divergence*, $p_T(\hat{x})$ is the output distribution of the teacher network (student if s) and $M = \frac{1}{2}(p_T(\hat{x}) + p_S(\hat{x}))$ is the average of the two distributions.

5.2.1 Our Settings

Given our goal, to enhance the utility of every single exemplar, we exploit a sequence of different setting and evaluate the performance and utility associated to each one. Particularly, when a new batch of classes comes, we first train our standard network used for classification and then use a copy of it as a teacher, and another ResNet32 (randomly initialized) as a student. We had to split the synthetization phase in 3 chunks, with the classes are equally distributed among them, to not incur in CUDA memory errors.

- **Random Noise:** as done by the authors of the paper we generate images starting from random noise. In details, at any task we synthesize a number of images, belonging to the actual batch of classes available, and store them as they would have been extracted from the real training dataset, we basically construct a synthetic exemplar set, and consequently reduce the existing one. Unfortunately the results don't match with the authors'. We suppose that our net, trained for relatively few epochs and moreover on a low resolution small dataset, composed of 500 images per class, doesn't make possible for the net to properly learn the underlying features and be able to reproduce them through DeepInversion algorithm, as shown in figures 11a, 12a.

- **Random Images:** We figure out that starting from random noise penalize too much the accuracy. We moved to sample m (where m is K divided by the number of classes seen so far) images randomly from the training set, and start from them instead of random noise. Our hope is to highlight through DeepInversion (Adaptive is not needed) the key features of any batch of images of any class. The results are still not satisfying.

- **iCaRL exemplars:** This time the exemplars are taken not randomly, but with the herding technique as explained in section 3.4.

Results We found that 300 epochs, $\alpha_{tv} = 2.5 \cdot 10^{-5}$, $\alpha_{l2} = 10^{-8}$, $\alpha_f = 5.0$ and $\alpha_c = 10.0$ work best for our specific case.

Comments First of all we tried to start from random noise, when inverting the net and creating images we noticed that, no matter how many iterations we made in order to synthesize them, there was still too much noise that could have been confused the model, so we tried to begin from existing images taken from the dataset noticing that in this case the noise was created in areas where there were not features important for the classification. What the algorithm was doing was isolating the parts of an image that are of importance when extracting characterizing features, as you

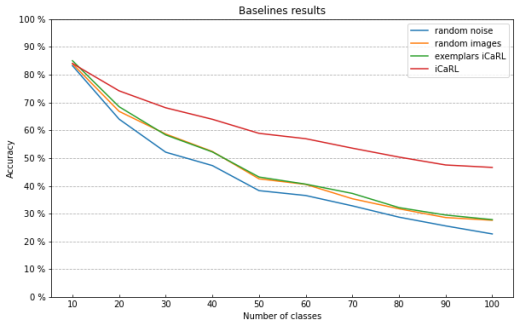
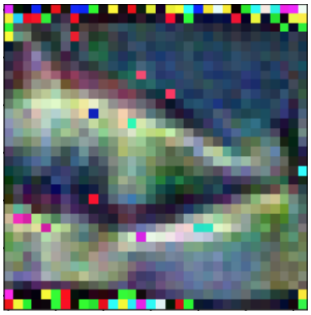
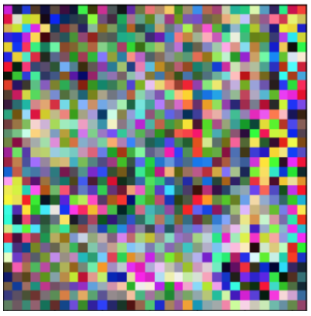


Figure 10: DeepInversion results

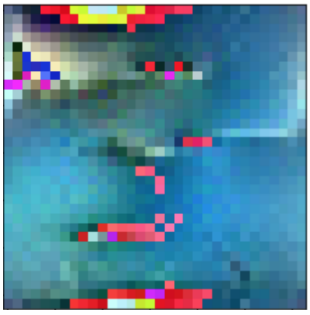
(a) Original shark image taken from CIFAR-100



(b) Same image after deep inversion



(c) Shark image created starting from random noise with our network



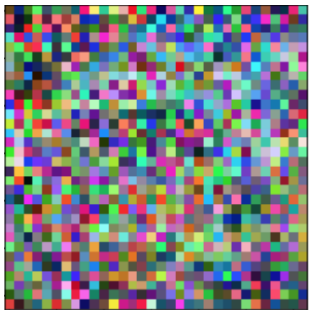
(d) Shark image created starting from random noise using a pretrained network

Figure 11: Four figures belonging to the ”shark” class

can see in figure 12b, where only the face and the hands of the kid are visible. The accuracy was a little higher in this case since the present features were similar to the ones the test data have, differently from the images created from random noise, and so we tried by applying the deep inversion on the exemplars taken by herding but this did not lead to a significant improvement.

(a) Original boy image taken from CIFAR-100

(b) Same image after deep inversion



(c) Boy image created starting from random noise with our network

(d) Boy image created starting from random noise using a pretrained network

Figure 12: Four figures belonging to the ”boy” class

6. Conclusions

This work expoited some of the incremental learning frameworks developed for multi-class incremental learning, such as Learning Without Forgetting and iCaRL, perfectly replicate the authors' works [1] [2]. Moreover it studied different combinations of classification and distillation loss functions and alternative classifiers.

Despite the fact we did not get good results in our proposals, we are firmly convinced that the key strategy to overcome the iCaRL limitations is to focus on the way exemplars are generated. We started by trying to manually create collages of images as possible exemplars, a second direction we tried to explore but we did not report, because of the unreliability of the results probably due to implementation problems, was to treat the exemplars as parameters to be optimized, in order to maximize their prediction capability on the current task data they should represents (the idea comes from [5]), while the third option, represented by *Deep Inversion*, could be a way to work around the problem because, even if we tried to fit it to the exemplar framework, it does not actually need any old data, but only the information contained in the batch normalization layers of the trained network, meaning that we could potentially create as old class images as we need having a balanced number of samples for each class. Concerning this last method, we

still suppose that it could bring advantages in the incremental learning setting. To validate this idea, we apply this it to our work because in the original paper, where the work has been made using as teachers pre-trained networks on ImageNet, the authors show how the synthetic images encapsulate meaningful features of each classes. Starting from this consideration, firstly we tried to fine-tune a pre-trained network on each batch containing 10 classes, extrapolate from this net the synthetic exemplars, and transfer these new images as exemplars in our workflow. Even if this could be a good idea, using pre-trained networks collide with the principles of incremental learning, and furthermore these synthetic images contains features that our networks has never seen. We moved away from pre-trained networks, trying to obtain good looking images that can embed as many important features as possible in a fixed amount of images. We tried many other approaches, such as training a second ResNet32 only on the current images, that were duplicated and augmented in order to have more images and variety as possible. Although the effort, all the trials didn't lead to significant results. We suppose that the main disadvantage for us is the dataset used: indeed, having only 500 images to train the model on, and being these images only 32x32 pixels the feature extraction is difficult and not totally appropriate. Still, we decided to report the more direct approach to apply this work to our task: synthesize the exemplars using our own network. In this case, going on with the batches and having to minimize the loss also on the old images the model was not good enough on new classes. The differences can be seen in figure 12.
Of course there could be other ways to improve the results by changing the way to distill the old knowledge but given the resources we had we preferred to concentrate only on an ablation study on the exemplars and the information they bring. Incremental learning is a real practical and challenging problem and a hot research topic, there are plenty of possible ways to cope with it already proposed in the literature, but there still not exist the perfect solution.

References

[1] Z. Li, D. Hoiem. Learning without Forgetting. In *ECCV*, 2016. 2, 9

[2] S. A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, 2017. 1, 2, 9

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[4] H. Yin, P. Molchanov, Z. Li, J. M. Alvarez, A. Mallya, D. Hoiem, N. K. Jha, J. Kautz. Dreaming to Distill: Data-free Knowledge Transfer via DeepInversion. In *CVPR*, 2020 7

[5] Y. Liu, Y. Su, A.A. Liu, B. Schiele, Q. Sun. Mnemonics Training: Multi-Class Incremental Learning without Forgetting. In *CVPR*, 2020 9