Incremental Learning project

Bianca Iacomussi
Politecnico di Torino
s277857@studenti.polito.it

Martina Toma
Politecnico di Torino
s276602@studenti.polito.it

Sofia Perosin
Politecnico di Torino
s269748@studenti.polito.it

Abstract

Incremental learning is a way to progressively acquire knowledge, without having all data available since the beginning. It is particularly useful in many applications where new data are accessible step by step and the re-training of the entire model is too expensive. However, this learning protocol is an open problem in computer vision and deep learning, due to the phenomenon of catastrophic forgetting, because of which previous knowledge is lost or only partially remembered. To deal with it, some efforts were made in research: from using distillation loss to keeping few exemplars from old tasks. In this work we have implemented the strategy proposed by iCaRL *paper, which is one of the most representative contributions to the research. Then, we compared our* iCaRL *implementation with* finetuning *and* learning without forgetting *methods, to better understand its effectiveness, and we made an ablation study by changing* iCaRL *loss and classifier. Finally, we present two proposals to deal with imbalance between earlier and new data: loss with weights and a second network for feature representation extraction.*

1. Introduction

Incremental Learning is a learning strategy that continuously extends model's knowledge with new input data. For example, when we travel and we see a lot of new places, landscapes, monuments, we still remember the city where we live. But this is not so easy for a machine learning algorithm, because when it tries to learn new things tends to forget the information acquired in the past. A more clear explanation is provided by figure 1. Training the model from scratch every time new data arrive is very expensive and time consuming, and above all most of the time old data are no longer available, so Incremental Learning reveals to be a suitable solution because it trains the model only with the new data. The main challenge related to Incremental Learning is the so called *catastrophic forgetting*. This problem consists in forgetting previously learned information when past data are not available, with a subsequent evident drop

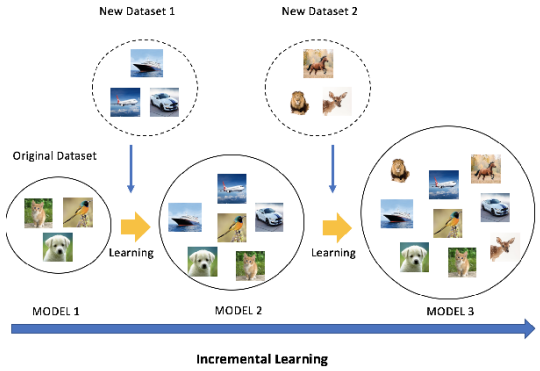


Figure 1: At first the algorithm is trained on a dataset containing only the images of initial classes, then at each incremental step new classes arrive and the algorithm should be able to correctly classify all the images in the dataset: both the old ones and the new ones. The third model has to correctly classify cats, dogs, cars, lions ..., but for sure it will perform better in the classification of the last classes seen: lions, horses and fawns.

in the algorithm performances. As a result, two challenges have to be faced [7]:

- the algorithm should maintain good performances also on the old classes,
- there should be a balancing between old and new classes.

Different models have been implemented in order to better understand how *catastrophic forgetting* could be overcome. First of all we have faced *Finetuning*: this algorithm does not implement any strategy to remember what was learned in the past, so it's completely affected by *catastrophic forgetting*. It is used as a baseline to measure the improvements made by other methods: *Learning without Forgetting* [8] and *iCaRL* [6] which aim is to maintain performances on old tasks. *Learning without forgetting* makes use of a mechanism that allows to preserve the knowledge of previous classes: in addition to the classical classification loss, it computes a distillation term which indicates how much

knowledge about old classes is lost: the lower the loss the better old classes are remembered. A further improvement is made by *iCaRL*, which combines *Learning without Forgetting* enhancement with a smart strategy: the most representative observations for each class seen so far are saved and subsequently used in the entire process: both in the training and in the classification part. After having implemented the papers mentioned, we have tried to make some modifications to *iCaRL* regarding the losses and the classifiers used. The final step consists in proposing an *iCaRL* variant taking into account some limitations of the algorithm.

2. Related work

Many recent studies have been made about incremental learning with deep neural networks. However, such a problem is not new in machine learning, because before the hype of deep learning, scientists realized incremental learning methods exploiting for instance linear classifiers, ensemble classifiers and nearest neighbors algorithm [7]. Nevertheless, since deep neural networks have become a pillar in computer vision, in particular Convolutional Neural Networks for image classification, it is normal to focus our discussion on incremental learning of deep neural networks.

Originally proposed by Hinton et al. to transfer information between diverse networks, knowledge distillation [4] is largely used in a slightly different way by the incremental learning methods presented here. In fact, they use the distillation within a single network in order to prevent knowledge acquired on old tasks from deteriorating by adding new ones. That is, distillation loss is a way to escape from catastrophic forgetting. One of the first important applications of this approach was made by Li and Hoiem in Learning without Forgetting [8]. The latter uses a variant of cross-entropy loss to preserve the knowledge. A significant improvement with respect to Learning without forgetting is the introduction of a memory to store elements of old tasks. Different works exploited this idea by implementing it in different ways.

Rebuffi et al. introduced a very effective solution to IL, *iCaRL*, which makes use of a combination of nearest-mean-of-exemplars classifier, prioritized exemplars selection and distillation loss. They made a careful analysis of *iCaRL*, presenting the algorithm in detail and showing that it outperforms the baselines of *Learning without Forgetting* and fixed representations on Imagenet LSVRC and Cifar-100 datasets. [6]

Wu et al. concentrated their efforts in large scale incremental learning and presented a strategy to tackle data imbalance problem and so to "alleviate" the instability between old and new classes. In fact, as the number of classes increases, there might be similar classes and the degradation applies also for this reason. Because of the number of images belonging to the last task is higher compared to those of the exemplars of the previous classes, there is a bias in favour of new classes in the last fully connected layer of the CNN and they have solved this problem introducing the *BiC*, the Bias Correction method. In practice, they added, after the last FC layer, a linear model that is a bias correction layer, to correct the bias. In this procedure, they used a small subset of exemplars which is excluded from training and used only for bias correction. The training procedure is composed by two stages: convolutional layers and fully connected layer are trained and then frozen in order to estimate the bias parameters. Distillation loss and softmax cross entropy loss for the classification loss were used. [7]

Hou et al. aimed to eliminate the problem of imbalance between previous and new tasks with a unified classifier, incorporating cosine normalization, less-forget constraint and inter-class separation. Cosine normalization lets all classes have the same importance avoiding "imbalance magnitude", that happens when the values of the weight vectors belonging to new classes are higher than the old ones. Less-forget constraint is a new loss added to the classification and distillation ones, it is needed to maintain the geometric orientation of the old classes without being subjected to the "deviation" caused by unavailability of old classes. Furthermore it's used an adaptive coefficient which takes into account the number of new and old classes. Inter-class separation is based on a margin ranking loss and aims to have the new class as distant as possible from the old ones, solving the "ambiguities" problem when there are similar classes. [5]

Belouadah and Popescu proposed an Incremental Learning with *Dual Memory*. Their neural network has a fixed architecture and an additional second bounded memory, in which initial class statistics are saved because it is better if data are modelled at the moment they arrive. Statistics are then reused to rectify the prediction scores of past classes. This rectification was proposed as an alternative to the NME classification by *iCaRL*, to compensate for the bias toward new classes. In addition, the loss is computed as the sum of distillation and classification terms, the latter computed by cross entropy. Different experiments were performed by varying the cardinality of the exemplar set and by avoiding the use of the distillation loss term, the so called vanilla finetuning. [2]

3. Implementations

3.1. Implementation details

The technical details of our implementations are presented in this section. Our source code is available at `https://github.com/SofiaMartinaBianca/Incremental-Learning`

3.1.1 Dataset

The dataset used is *CIFAR-100*, it consists in 100 classes, each of them contains 600 images: 500 training images and 100 testing images. The resolution of the images is 32×32. To simulate an incremental learning protocol we divided the total classes in 10 groups; this subdivision allows to add 10 new classes at each incremental step. To better compare results, the same fixed split is used in all models, discussed next.

3.1.2 Network

To implement the model we exploited *PyTorch* library and we made use of a *ResNet-32* specifically modified to handle *CIFAR-100* images size; in fact since the resolution is small it's more efficient to adapt the network for receiving 32×32 images, instead of resizing the images themselves. [3]. The networked presented in [3] is slightly modified in order to perform our analysis. Firstly in the forward method is possible to retrieve in addition to the scores, also the feature representations when they are needed. Then in order to have an approach as general as possible the starting output size of the network is equal to the number of initial classes, then with the arrival of new tasks the output size is incremented, paying attention to maintain biases and weights of the old neurons.

3.1.3 Hyperparameters

The same hyperparameters of *iCaRL* paper [6] are used among all the implementations: the network is trained with mini-batches of size 128, the optimizer used is *stochastic gradient descent* with momentum=0.9, weight decay=0.00001 and initial learning rate=2, which is divided by 5 after the 49^{th} and 63^{th} epoch. The training is performed for 70 epochs. The last hyperparameter, K, used only in *iCaRL* implementation, is set to 2000.

3.2. Finetuning

Finetuning works as a simple multi-class algorithm without adopting any measures to prevent catastrophic forgetting: it starts with a new network which is subsequently trained on the given first task, then it uses the parameters of the old network as a starting point to perform the second task, and so on. The weak point of this approach is that if the final network is tested on classes belonging to old tasks the performances are very poor: this is the *catastrophic forgetting* problem. In other words, what the network learns in the previous steps is useful to initialize the weights for the new task, but after the training the new weights are appropriate only for the current images, and completely meaningless for the old ones.

3.3. Learning without Forgetting

The presented *Learning without Forgetting* is slightly different from the one proposed by [8], in order to have comparable results with the ones obtained from *iCaRL*. In particular the main differences consist in:

- the choice of the loss: the original paper [8] proposed a multinomial logistic loss for the classification part and a modified cross entropy loss for the distillation part. The latter is multiplied by a loss balance weight, this parameter will favour the old task performance over the new task's [8]. Furthermore the input of the classification loss is the softmax of the output of the network, instead in *iCaRL* the sigmoid is used; for the target is applied the same approach: one hot encoding of the labels.

- the network used: instead of MatConvNet [1, 8] and AlexNet [8] we exploited a Resnet32.

- the data augmentation since the dataset used is different: our analysis is focused on CIFAR-100.

- consequently all the hyperparameters are chosen according *iCaRL* implementation.

Training. The network is trained using only the images belonging to the new classes available at each time, but to avoid *catastrophic forgetting* two losses are computed simultaneously: classification and distillation loss. The latter encourages the network to not lose the discriminative information learned previously. The final loss is minimized in order to update the network parameters: this process tries to force the network to correctly classify the new classes (classification loss) and to reproduce the scores done by the previous network for the old classes (distillation loss). The loss used is a Binary Cross Entropy Loss with Logits:
$l(x, y) = L = \{l_1, ..., l_N\}^T$
$l_n = -[y_n \cdot log(\sigma(x_n)) + (1 - y_n) \cdot log(1 - \sigma(x_n))]$
where l_n is the loss computed on the n^{th} image of the considered batch; the input of the loss, x_n, are the outputs of the current network; instead the targets, y_n, are the sigmoid of the outputs of the previous network for the images belonging to the old classes (distillation loss) while for the images belonging to the new classes the target is the one hot encoding of the label (classification loss). Classification and distillation loss are computed together because in this way the final loss can be easily calculated by averaging on the losses of all the images in the batch.

Classification. The class is assigned according to the output of the last fully connected layer of the network: the highest score will indicate the predicted label.

3.4. iCaRL

iCaRL, Incremental Classifier and Representation Learning, is an innovative strategy for incremental learning problems. It is based on three main components: [6]

- classification using the *nearest mean of exemplars* approach, instead of the scores of the last fully connected layer of the network;
- computation of *prioritized exemplar selection* based on herding;
- employment of *knowledge distillation* and *prototype rehearsal*.

Training. The main structure is the same as the one described for *Learning without forgetting* implementation: the same loss (Binary Cross Entropy Loss with Logits) is used to update the network. The main difference is the presence of exemplars, a subset of the old images dynamically constructed, which are combined with the new images. Exemplars are very useful to prevent forgetting old tasks, since the training phase takes into account both new images and exemplars.

Exemplars management. The procedure followed to handle with exemplars consists in updating the exemplar set: removing some old exemplars and adding new ones since in this set can be stored at most a fixed number of elements. When new classes are encountered, the maximum number of exemplars that can be saved for each class, m, is updated as $m = \frac{K}{t}$, where K is the maximum size of the exemplar set and t is the number of classes seen so far. As a consequences only the first m exemplars for each class previously seen will be kept and m new exemplars will be added for each new class. In this way there will be the same number of exemplars for each class examined until now. Exemplars are chosen according to the feature importance: the network extracts the feature representation for each image, $\phi(x)$, and, after normalization, the latter is used to identify for each class the most representative observations that will be added to the exemplars set. The following procedure is used to select the most meaningful image:

$$p_k = \arg\min_{x \in X} \|\mu - \frac{1}{k}[\phi(x) + \sum_{j=1}^{k-1} \phi(p_j)]\|, \quad \forall k \in [1, m]$$

where $\phi(p_i)$ is the normalized feature representation of the exemplar p_i. Given the image set X of class y, the mean of feature representation is computed as $\mu = \frac{1}{n} \sum_{x \in X} \phi(x)$.

Then the exemplar set is iteratively updated: are added those images which ensure that the euclidean distance between the class mean and the average feature vector of exemplars is as small as possible. In order to not waste memory with duplicates, since for each class can be stored only m exemplars, every time an image is selected as an exemplar p_k, the corresponding feature vector, $\phi(p_k)$, is removed from the set of all features. This approach, the so called *prioritized exemplar selection* based on herding [6], enables to have as final result a list of exemplars sorted according to their importance, i.e. the proximity to the class mean, as shown in the previous formula. In this way the previous step, the removing part, is facilitated because it will be enough to remove the last elements: this is important since the mean vector of old classes is no longer available. This exemplar set created by the explained procedure will be combined with the training set of future tasks in order to implement the principle of *rehearsal*, that consists in using both the new data and exemplars of old classes.

Classification. The classification part is no more based on the output of the last fully connected layer, but exploits the network as feature extractor in order to infer the feature representation. The latter is a fundamental component of the classification method used: NME, *nearest-mean-of-exemplars*. Firstly the average feature vector for each class seen is computed: for the old ones the corresponding exemplars are used, instead for the new classes the images in the training set are taken.

$\mu_y = \frac{1}{|P_y|} \sum_{p \in P_y} \phi(p)$

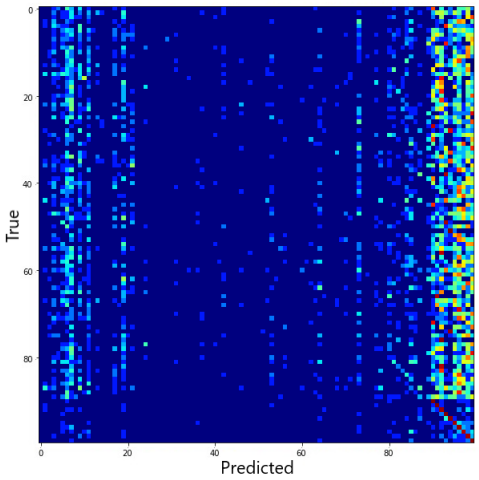
μ_y is also called prototype vector since it summarizes all the feature vectors of the observations of class y.

Note that each feature representation is normalized after being computed.

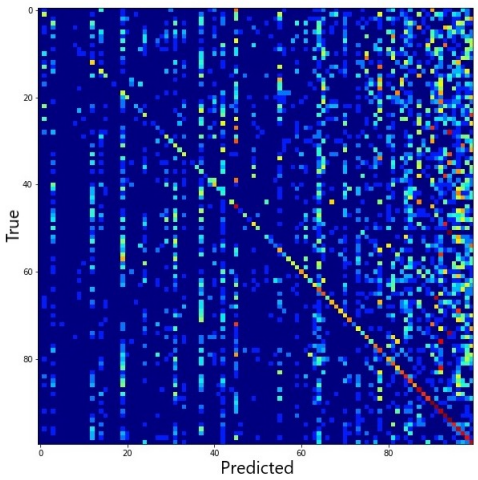
Finally for each image x that has to be classified, the label is assigned taking the smallest distance between the feature vector $\phi(x)$ and each prototype:

$y^* = \arg\min_{y=1,\dots,t} \|\phi(x) - \mu_y\|$

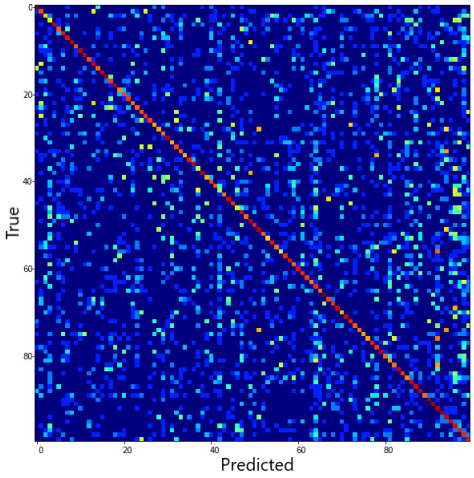
3.5. Results



(a) Finetuning



(b) Learning without forgetting



(c) iCaRL

Figure 2: Confusion matrices of different implementations on CIFAR-100. The entries are transformed by $log(1 + x)$ for better visualization.

Figure 2 provides the confusion matrices of the three implementations done, and interesting patterns can be seen:

Finetuning predicts almost of all the images in the last classes seen, this is the effect of the *catastrophic forgetting*: the network remembers only the most recent classes which are classified in a good manner, but it completely forgets the existence of earlier classes. Only some other classes previously analyzed are remembered and this could due to the fact that they have very particular characteristics that allow to correctly classify them.

LwF presents a more homogeneous pattern with respect to *finetuning*. The diagonal begins to take shape, indicating that some classes starts to be well predicted. However, good classifications are ensured only for the last 50-40 classes and a pattern similar to *finetuning* is present in the last group of classes. This means that the model quite remembers how to recognize old classes thanks to the *distillation loss*, but it is always better in the last tasks.

iCaRL confusion matrix shows that the model is capable of remembering the old classes avoiding the *catastrophic forgetting*. As can be easily seen, almost all the images belonging to previous classes are correctly identified and classified. Thanks to exemplars and *distillation loss*, *iCaRL* implementation performs best: in fact the pattern in this matrix is the most homogeneous. However, at the end, there is a small tendency towards the new classes because there are more images referred to the last 10 classes than the exemplars of the old ones.

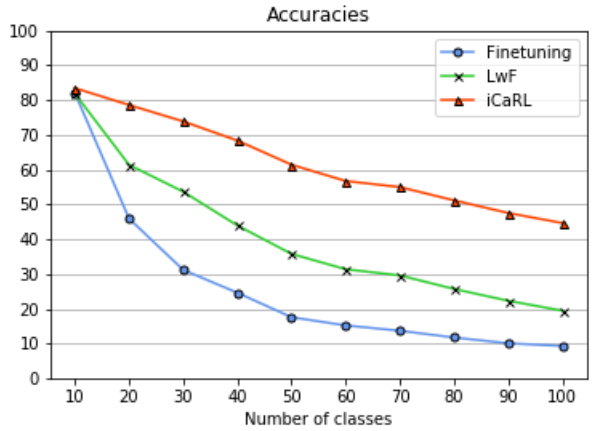


Figure 3: Accuracy of *Finetuning, Learning without Forgetting, iCaRL*

Figure 3 confirms the results presented by the confusion matrices: *finetuning* reaches the worst accuracy scores due to *catastrophic forgetting*, *learning without forgetting* improves a little bit, however *iCaRL* is the best model. In fact it is capable of maintaining good performances also when classes are incremented, and the accuracy decreases in a slower way than the others. At the beginning the performance of all the three models are very similar because there are no old classes and *distillation loss* and *exemplars* are not applied. Then the gap between the methods becomes more evident going on in the process.

4. Ablation study

Some modifications to the *iCaRL* loss and to the classification method are done in order to better understand pros and cons of *iCaRL* components and to see if there are some improvements.

4.1. Classifier

Instead of using the *Nearest Mean of Exemplars*, three machine learning algorithms are exploited to classify. Because we passed them as input the feature vectors of all the training images, only the exemplars are used in order to have balancing among new and old classes: otherwise if the entire training set is used there will be much more data for new classes and the old ones will be underrepresented, and in this way the classifier would be unbalanced. This could be done because at each step the exemplars of the new classes are created after the network training but before the classification step, so in the exemplar set there are also images related to the current classes when we are classifying.

4.1.1 K-Nearest Neighbours

K-NN classifier seemed to be a valid alternative to the nearest class mean of exemplars, due to the fact that exploits a mechanism based on distances too. However, instead of taking the minimum Euclidean distance between the features of the image that has to be classified and the mean of exemplars among all seen classes, K-NN makes use of Minkowski distance: it assigns the predicted label by majority voting looking at the distances from the nearest K neighbors. In this case, we passed to the classifier all the feature representations of the exemplars and their corresponding true labels. We noticed that, adding new tasks, it is more difficult for the network to learn, in fact the training accuracy (the one on new images) decreases. Using K-NN as classifier, the model performed similarly to *iCaRL*, both in the accuracy variation among the 10 tasks and in the appearance of the confusion matrix, in which we noticed a behavior similar to the *iCaRL*'s one, but with little tendency to predict many images in the last 10 classes.
To choose a good value for the number of neighbours K, some alternatives were tried: 3, 5 and 10. Among these, K=10 gave the best results.

4.1.2 Support Vector Machines

The classifier is trained on 2000 observations, but when the number of classes increases, the number of exemplars for each class decreases gradually. Having few observation for each class can lead to have a classifier that after the training step is not good at correctly classifying new data. For this reason, we thought that a good candidate could be Support Vector Machine since it minimizes the empirical error and the structural error, which is the best approximation of Bayes risk, and in this way it's possible to decrease the error on test data.
The hyperparameters tuning is done by varying the kernel (*linear* and *rbf*) and C (0.1, 1 and 10) in order to allow the model to be more flexible if there is noise in the data (C indicates the measure in which mistakes are allowed), or if data are not linearly separable, by looking at an hyperplane in higher dimensional spaces thanks to *rbf* kernel. The best configuration found uses rbf kernel and C=0.1, but also a linear kernel works pretty similar and this could mean that data are quite linearly separable.

4.1.3 Logistic Regression

The ability of the logistic regression classifier is to discriminate among different classes by assigning higher weights to the most important features.
To take the best configuration for C, the regularization penalty, 3 different values were used: 0.1, 1 and 10. It resulted that the smaller the value of C the worse the accuracy

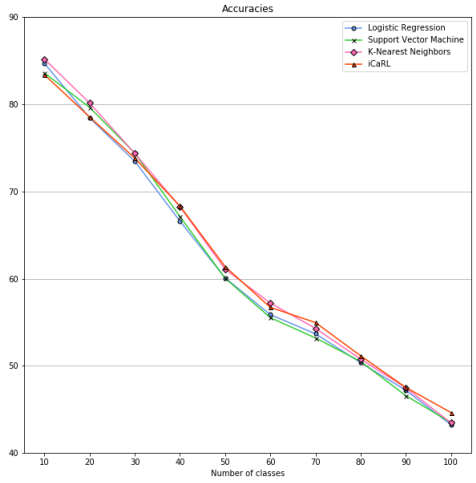
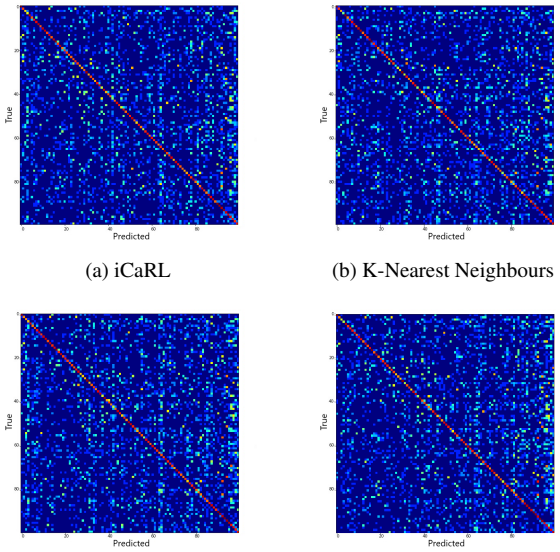


Figure 4: Accuracy of the three different classifiers tried, compared to *iCaRL*.



(a) iCaRL (b) K-Nearest Neighbours

(c) Support Vector Machines (d) Logistic Regression

Figure 5: Confusion matrices of the three different classifiers tried, compared to *iCaRL*. The entries are transformed by $log(1 + x)$ for better visualization.

and so, when C is too low it underfits the data and so it cannot learn enough information; instead the best results are reached by C=10.

According to figure 4 performances of these variations and *iCaRL* are very similar. It could be noticed that *iCaRL* at the first 10 classes has the worst performance, while at the end it reaches the highest accuracy. The similar behaviours of the four implementations is also confirmed by the confusion matrices of figure 5.

4.2. Loss

About the loss used in the model, *iCaRL* proposed a *BCE with Logits* loss for both distillation and classification part. Next some alternatives are tried in order to understand how the choice of the loss can impact on the performances of the model.

4.2.1 BCE - CE

The first modification consists in changing only the classification loss, and instead of using *BCE with Logits* we used *Cross Entropy Loss*. This loss was adopted to try an alternative to the use of one hot encoding. In fact it is suitable for multi-class classification and it does not require any mechanism to transform labels in a binary form. This loss is computed according the following formula:

$$loss(x, class) = -log\left(\frac{exp(x[class])}{\sum_j exp(x[j])}\right)$$

Where x is the corresponding network output of an image belonging to a $class$, so $x[class]$ is the output of the neuron corresponding to that $class$. In other words this formula computes the negative logarithm of $softmax(x[class])$.

4.2.2 BCE - L2

We tried to use the same distillation loss of *iCaRL*: *BCE with Logits*, but *L2* loss for classification.
The latter is computed by the Euclidean distance as follows: $l_n = (x_n - y_n)^2$.
Where x_n is obtained by doing the softmax of the network output, considering only the neurons related to the last task, while y_n is the one hot encoding of the labels. This loss is tried because it allows to measure the distance between the probability predicted and the one hot encoding: namely for each image x_n will contain the probabilities that the image belongs to each new class, while y_n contains the one hot representation of the label, so there will be 1 for the element corresponding to the true class, while the other elements will be 0. In this way the model will be more penalized if higher probabilities will be assigned to the elements in y_n equal to 0, and at the same time if a low probability corresponds to the element equal to 1.

4.2.3 L2 - L2

Another variation consists in using *L2* loss both for distillation and classification. The classification part is the same as described before. The distillation loss takes as input the sigmoid of the network output, considering only the neurons

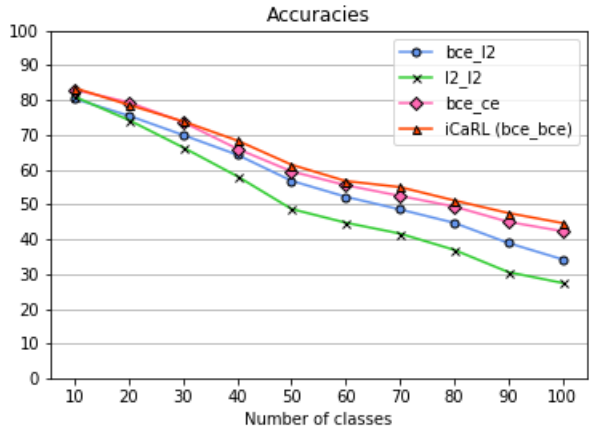
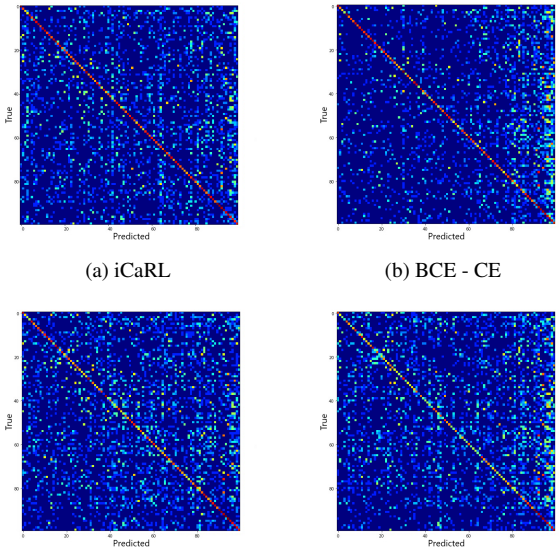


Figure 6: Accuracy of the three different loss combinations tried, compared to the one used in *iCaRL*.



(a) iCaRL (b) BCE - CE

(c) BCE - L2 (d) L2 - L2

Figure 7: Confusion matrices of the three different combinations of losses tried, compared to *iCaRL*. The entries are transformed by $log(1 + x)$ for better visualization.

related to old tasks. In this case the target passed to the loss are the sigmoid of the old network: we did not use the softmax as before in order to have a direct comparison with the sigmoid of the network outputs. We though that this could be an interesting variation since it allows to easily analyze the differences between the logits of the old and current network for the images belonging to the old classes (thanks to distillation part), and the differences between the predicted probabilities and the true one for the images belonging to the new classes (thanks to classification part). But this decision resulted to be slightly inefficient, maybe the choice of hyperparameters is not adequate, even if we tried to adapt the learning rate to this different loss.

Figure 6 and figure 7 show the results obtained.

5. Our Proposal

We have identified as weakest point of iCaRL the fact that even if exemplars of old classes are combined with the train dataset, the network is still unbalanced and tends to predict new classes in a more accurate way. We have addressed this problem with two different proposals: the use of weights in loss calculation and the creation of an additional network used in the classification phase combined with a strong data augmentation on the exemplar set.

5.1. Loss with weights

To implement this first proposal we adopted an alternative to the Binary Cross Entropy loss for the classification term: Cross Entropy loss with weights. During the computation of the classification loss, all the network scores (both new and old) are taken into account and different weights are assigned to the old classes in order to give the same importance to all the classes present at that moment. At the beginning, when there are only 10 classes, all the classes have the same weight. Later, as new classes arrive, we assigned them a smaller weight because they are represented by more images, while we gave the exemplars of old classes a higher weight.

In particular the weights associated to the new classes are all ones, while those for old classes are computed by:

$$weigh\_old\_classes = \frac{n\_images\_per\_new\_class \times n\_old}{K}$$

where $n\_images\_per\_new\_class$ is the number of images for each new class, n_old is the number of old classes and K is the maximum number of exemplars that can be stored. Doing in this way each class gives the same contribution in the calculation of the classification loss. It does not matter anymore that there are more images for the new classes. The final result did not present significant improvements, because the model is focusing too much on remembering old classes and it is not learning in an efficient way new classes.

5.2. Second network for feature representation extraction

The latter proposal focuses on the idea of using a second network to perform the feature extraction at classification time. As previously described, during the classification the model takes the exemplars and the images of new classes from training set, extracts their feature representation, computes the mean feature vector for each class and then classifies images according to the nearest class mean. The problem here is that the feature representation of each image is computed by a network which is trained on a lot of images

belonging to new classes and less images from old classes. This leads to have an unavoidable unbalancing: the network is more good in working with new classes instead of previous ones.
To solve this problem, we have used a second network to acquire the feature representation: this new network is obtained by coping the principal network and then it is trained only on the exemplars set (which at this step contains the exemplars of all the classes seen so far) on which data augmentation is applied. In particular, besides horizontal flip and random crop, some new transformations are allowed: color jittering (brightness, contrast and saturation are changed of a factor $\gamma \in [0.5, 1.5]$, instead hue is modified by a factor chosen uniformly from $[-0.5, 0.5]$), random vertical flip (but with a small probability, $p = 0.1$) and random rotation (of an angle $\alpha \in [-15, +15]$).
In this way the second network should be less liable to act in an unbalance way since it is re-trained by using the same number of images for each class. This new network is now used to extract the feature representation of each training image (which is needed to compute the classes' means) and to obtain the feature representation of each observation that has to be classified. We have considered this an interesting approach since also the feature representations of the images belonging to old classes, that have to be classified, are computed by a network more balanced than the one previously used.
The final performances did not improve and this could be due to the fact that, even if the network used to extract the feature representation is more balanced, the final classes' means (needed to classify) are computed by using more images for the new classes.

Figure 8 and figure 9 show the comparison between the performances of our proposals and *iCaRL*.

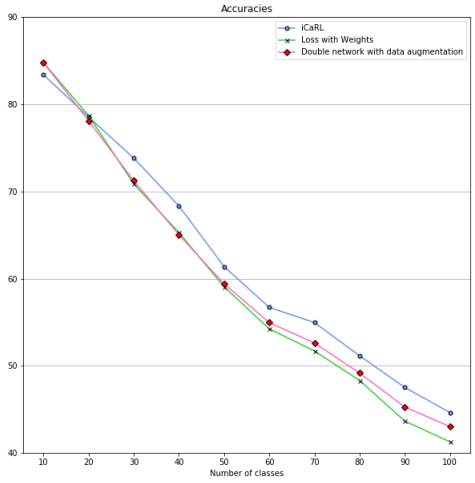
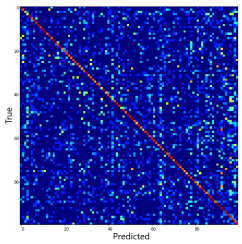
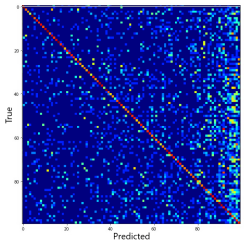


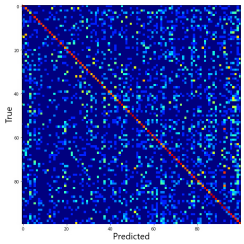
Figure 8: Accuracy of our proposals, compared to *iCaRL*.



(a) iCaRL



(b) Loss with weights



(c) Double network with data augmentation

Figure 9: Confusion matrices of our proposals, compared to *iCaRL*. The entries are transformed by $log(1 + x)$ for better visualization.

6. Conclusion

Our aim is to forget knowledge as less as possible in order to be able to perform a classification as good as possible. Even though is not completely possible, our project tried to minimize the loss of the knowledge, derived by catastrophic forgetting and imbalance of the classes. The experiments done confirm that distillation loss and exemplars are fundamental in order to avoid catastrophic forgetting.

About the different modifications tried, the choice of the loss done by *iCaRL* is undoubtedly the best one since it's the most efficient to train the network in learning new classes and remembering the old ones. Regarding the classifiers used, it is interesting to note that the final results of *nearest mean of exemplars* of *iCaRL*, *K-Nearest Neighbours*, *Support Vector Machines* and *Logistic regression* are very similar, but *iCaRL* needs much more images to compute the means since it uses the exemplars and the images in the train, instead the other three classifiers need only the feature representation of the exemplars.

We tried to overcome the issue of imbalanced data by giving the same importance to all classes in our first proposal and by using an additional network, which is trained only on the K exemplars, to compute the feature representations, in the second one.

Further improvements could be focused on finding a more stable model since the variance of performances is quite high: according to the order in which the model encounters classes the resulting accuracy could be pretty different. This could depend on the fact that if the new classes are completely different from the old ones, the classification task will be harder. Moreover, with the arrival of new classes there is a higher probability that a class will be wrongly classified, if there are similar classes.

References

[1] A.Vedaldi and K.Lenc. Matconvnet– convolutional neural networks for matlab, 2015.

[2] E. Belouadah and A. Popescu. Il2m: Class incremental learningwith dual memory, 2019.

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

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

[5] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin. Learning a unified classifier incrementally via rebalancing, 2019.

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

[7] Y.Wu, Y.Chen, L.Wang, Y.Ye, Z.Liu, Y.Guo, and Y.Fu. Large scale incremental learning, 2019.

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