Incremental Learning Project

Antonino Geraci
s272089
antonino.geraci@studenti.polito.it

Mauro Musarra
s277914
s277914@studenti.polito.it

Matteo Tarantino
s274156
matteo.tarantino98@studenti.polito.it

Abstract

Incremental learning is an ongoing and open problem, many approaches have been proposed in the last years. In this report we analyze some of these methods, comparing strengths and weaknesses of them, focusing especially on the single components of an incremental learning setting, starting from two common baselines : iCaRL and LwF, all methods are evaluated on the common baseline dataset CIFAR-100.
Finally we propose our variation, composed by three changes in the training pipeline : a custom distillation loss, a constraint relaxation and an advanced scheduling of the regularization term, across different incremental steps.

1. Introduction

The main challenge to face in incremental learning is to find a method to avoid the forgetting of previous learned knowledge. In a normal training setting, if new classes arrive, the network, that is already in a stable state for the classes learnt so far, would change the weights of the previous ones, trying to learn the novelty. Unfortunately, this naïve approach leads to the so-called *catastrophic forgetting*, totally erasing the previous knowledge and learning just the new classes. The most widely explored tools to prevent this problem are :

- Distillation loss
- Retrieval of previous class exemplars
- Classifier selection

and they will be declined in all the experiments.

1.1. Implementation details

All experiments are done on a dataset that is a common baseline for incremental learning algorithms : CIFAR-100. It contains 60k 32x32 RGB images of 100 object classes. Each class has 500 training images and 100 testing images. According to our benchmark protocol the 100 classes are divided in batches of 10 classes, in order to simulate the arrival of new annotations. The CNN used to accomplish this task is the *ResNet-32*.

2. iCaRL

ICaRL[1] is a common baseline compared with all new methods in icremental learning. It is the natural evolution of LwF[2], that imposed the general framework, then consistently improved by iCaRL variations. The idea of LwF is to penalize the forgetting of previous classes, by adding a new term to the loss function : the *distillation loss*. Hence, when new classes arrive, the training phase is driven by a standard classification loss, that should learn properly the new classes and the second term that aims to keep stable the probabilities on the older ones.
In the original LwF implementation, *cross entropy loss* for both classification and distillation is used. In this report, for a fair comparison, both LwF and iCaRL are implemented with the loss proposed by the latter.
iCaRL, keeping the same approach described before, introduces two important novelties : the NME classifier and the use of exemplars.

2.1. BCE Loss

The first difference introduced by iCaRL is the use of the binary cross entropy loss for both classification and distillation. Unlike a standard cross entropy, this loss performs an one vs. all computation then it sums all contributes, for all classes, taking into account all outputs logits and not only the ones of the ground-truth label. The total loss used is:

$$\mathcal{L}_{iCaRL}(p,y) = -\left(\sum_{c\in C^t} y_c \log p^c_{\theta^t} + (1-y_c)\log(1-p^c_{\theta^t})\right) + \\ -\left(\sum_{c\in C^{t-1}} p^c_{\theta^{t-1}} \log p^c_{\theta^t} + (1-p^c_{\theta^{t-1}})\log(1-p^c_{\theta^t})\right) \quad (1)$$

In the first part of the loss, y_c is binary, because we are just caring about the true class, while in the distillation part, $p^c_{\theta^{t-1}}$ is called *soft-target*, because it represents the logit value of the old model computed on the current training data, used as a target for the distillation loss. In fact the second part of eq.(1) has a convex shape, with a single minimum in $p^c_{\theta^t} = p^c_{\theta^{t-1}}$, meaning that the distillation loss is minimized when the old and the new logit values for the class *c* are the same, therefore when the current model keeps the same knowledge of the previous one about the old classes. The effect of this loss is to find a trade-off between classifying correctly the new classes and keeping stable the knowledge about the older ones.

2.2. Exemplars

Our implementation of LwF, using just the loss mentioned before, performs widely better than a standard finetuned model, that achieves a 0% accuracy on old classes, but performances are still not satisfying. Conceptually, human brain, when dealing with a new task, does not erase his own memory, but, a part of what it has learnt before is maintained and always present. In the same way, a method to avoid forgetting is to maintain a small set of images belonging to the previous classes, in the next training phase. If we had too much images, the problem would be transformed in a joint training, that for its computational cost must be abandoned. In order to define a fixed memory budget, the number of exemplars from the previous classes and consequently the number of exemplars for each old class, must be respectively:

$$K = 2000, K_c = \frac{K}{|C^{t-1}|} \quad (2)$$

where C^{t-1} is the set of classes at the previous step.
In this way, a smart trade-off is reached, and in general the use of exemplars is quite widespread in incremental learning tasks. iCaRL presents its own way to select the exemplars, through a greedy algorithm called *herding*; actually some recent works have demonstrated that picking them randomly avoids any kind of bias introduced by an algorithm, leading to a better distributed set of exemplars. Also in this report, we found that the random selection of exemplars is more performing and of course less resources demanding.

2.3. Classifier

Since the net is trained using the FC layer of the *ResNet-32*, in a standard learning setting the straightforward solution at test time would be the use of the fully connected response as a classifier. However, in an incremental setting is not probably the best choice, in fact, as the iCaRL paper underlines, if the feature map is updated after an incremental step, the weights of the fully connected will not be updated accordingly, but they change decoupled from the feature extraction routine, therefore they could have a bias towards the new classes. On the other hand, if the FC layer is used just in the training phase and the *nearest-mean-of-exemplars (NME)* in the classification task, the overall knowledge is distributed more uniformly across all the incremental steps, because the latter uses only the features to classify, making it strictly coupled with the feature extraction routine. The prediction of the *NME* is taken as follows:

$$\hat{y} = \underset{y\in C^t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\| \quad (3)$$

where $\varphi(x)$ is the feature map retrieved by the last layer of the convolutional network and μ_y is the centroid of the class y. Therefore the feature map influences directly the prediction.
We will name the pipeline using FC layers $iCaRL_{Hybrid}$ and the other one $iCaRL_{NME}$.

2.4. Comparison between baselines

In order to compare the results with a lower and an upper bound, using the same setting and hyperparameters of iCaRL, respectively: a finetuned model (i.e. a model trained on the new classes using only a classification loss) and a model trained on all images incrementally (i.e. joint training) were trained. We use the same hyper parameters of iCaRL: *LR* = 2, reduced by a factor 5 after the 49^{th} and the 63^{th} epoch, $weight_{decay}$ = 10^{-5}. Moreover, the mean of exemplars is computed before the reduction phase in order to have a less biased representation of the centroids. The first evident behavior of the incremental accuracy scores over the ten batches is that exemplars are very crucial for this task, in fact LwF has a huge gap with iCaRL. Secondly, the other important comparison is done on the classifiers, demonstrating that NME performs better than FC layers in all batches. [fig. 1] Besides the graph, that is self-descriptive, the difference between confusion matrices is quite interesting :

- $Finetuning$ reflects the negative effect of *catastrophic forgetting*, in fact the network predicts only on the last batch of classes.

- LwF using just a distillation loss has an unbalanced propensity to predict in the new classes, not avoiding consistently catastrophic forgetting.

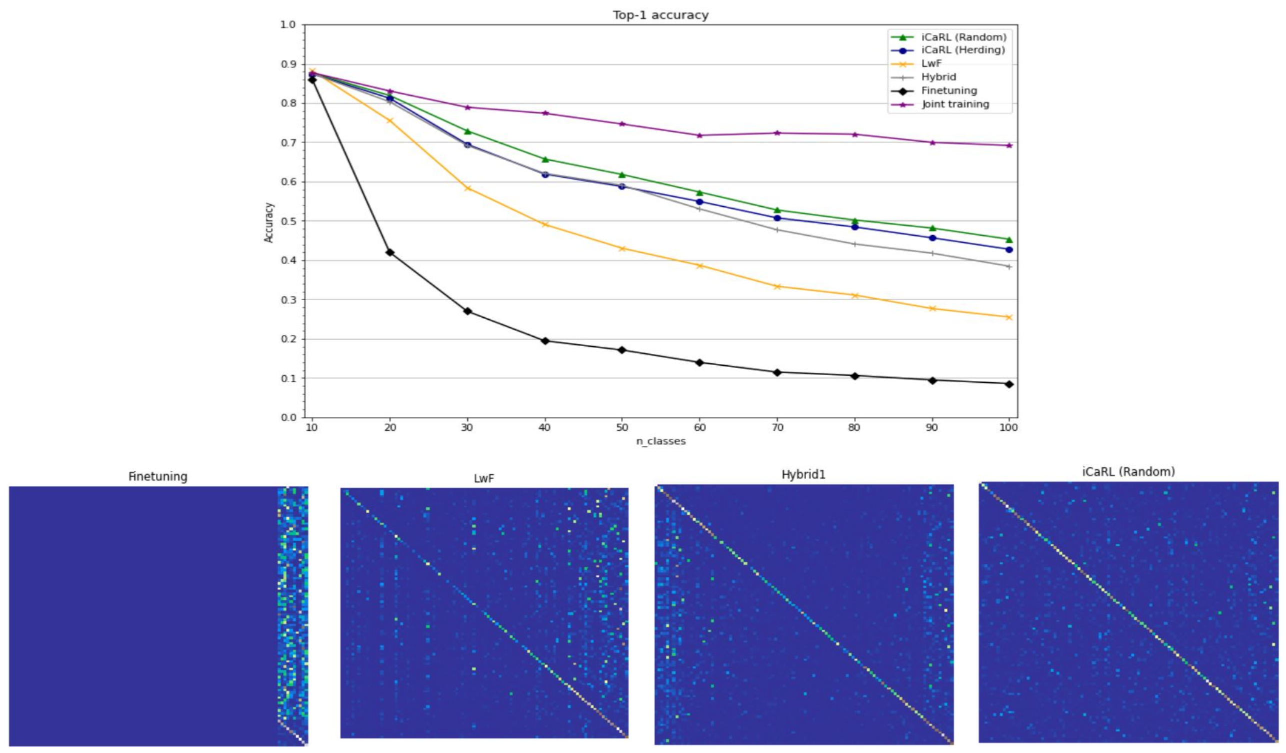

Figure 1. Graph of incremental accuracy (10 batches) - Confusion matrices of the different approaches

- $iCaRL_{Hybrid}$ is basically LwF with the use of exemplars, but in this case the confusion matrix suggests also a strong presence of a bias in favor of the first classes learnt by the model and not only on the last ones.
  In order to understand this behavior is necessary to keep in mind that the distillation term tries to replicate as much as possible the network response on the old classes from one step to another, updating suitably their weights. So, since the distillation term starts to act on the model created after the first step, in the *hybrid* case in which also the exemplars are present, the response of the old model on those images will be ever biased towards the classes learned the first time, whereas in the *LwF* setting, the soft-target is computed only on the new images, therefore the response of the old model cannot favor the first classes as it works with never seen feature maps.

- $iCaRL_{NME}$, the solution proposed by iCaRL is much more balanced than the other ones, showing the effectiveness of this classifier in an incremental learning setting.

3. Ablation study for Classifiers

In this section we will focus on the main aspects of three classifiers: *cosine normalization layer(CNL)*, *KNN*, *incremental dual memory (IL2M)* comparing them with $iCaRL_{NME}$ and $iCaRL_{Hybrid}$.

3.1. IL via rebalancing

In the context of multi-class incremental learning, the main challenge is to deal with the significant imbalance between old and new classes, trying to limit as much as possible the bias in the predictions that favor new classes.
The only addition of a distillation loss is not enough to tackle this problem, since a predicted probability for a class *c* of an image *x* is computed as follows:

$$p_c\left(x\right) = \frac{exp\left(w_c^T\varphi\left(x\right)+b_c\right)}{\sum_{c'\in C} exp\left(w_{c'}^T\varphi\left(x\right)+b_{c'}\right)} \tag{4}$$

where *w* and *b* are respectively the weights and the bias vectors of the fully connected layer, while φ is the feature extractor.
The more new classes are added, the higher are both the weights and biases for new classes compared with the old ones, subsequently the classifier tends to favor new classes during the prediction phase. For this reason, in addition to the loss, iCaRL proposes a *nearest-mean-of-exemplars* classifier described in the paragraph 2.3.
Another approach, proposed by Hou et. al.[3], incorporates three components to deal with the undesired prediction bias due to the imbalance in the training data:

- **Cosine normalization layer**: in order to reduce as much as possible the aforementioned difference in magnitudes of the last layer between old and new classes, a typical fully connected layer is substituted by a *cosine normalization layer*, thus transforming the predicted probability $\hat{p}$ for a class *c* of an image *x* as follows:

$$\hat{p}_c\left(x\right) = \frac{exp\left(\theta\langle\bar{w}_c^T, \bar{\varphi}\left(x\right)\rangle\right)}{\sum_{c' \in C} exp\left(\theta\langle\bar{w}_{c'}^T, \bar{\varphi}\left(x\right)\rangle\right)} \tag{5}$$

  where $\bar{w}$ and $\bar{f}$ are the l_2 normalized vectors, thus the output ranges from -1 to 1, then a learnable scalar θ is introduced to scale properly its value since it will passes through a softmax layer.
  In this way the linear layer would not have a bias towards weights of the new classes because of the normalization: difference in scale would be reduced.

- **Less forget constraint**: in order to preserve the previous knowledge, a distillation loss on the features, also known as *less-forget constraint*, is computed as below:

$$\mathcal{L}_{dist}\left(x\right) = 1 - \langle\bar{\varphi}_{old}\left(x\right), \bar{\varphi}_{new}\left(x\right)\rangle \tag{6}$$

  where $\bar{f}_{old}$ and $\bar{f}_{new}$ are respectively the l_2 normalized features extracted by the old model and those by the current one, thus encouraging the current network to preserve the orientation of the features extracted by the old model. A feature-based distillation loss combined with a cosine normalization layer tries to overcome the decoupled behavior between features and weights presented in $iCaRL_{Hybrid}$, rebalancing as much as possible the current network response between old and new classes.
  As classification loss, a standard *cross entropy loss* is used.

- **Inter-class separation** : in order to separate as much as possible the network responses of the old classes from the new ones, another contribution to the loss is introduced, now computed only on the exemplar set *E*. In particular, for each exemplar *x* labeled with a specific old class *c*, it tries to separate the current network response for that class *c* from the K higher network responses among new classes, i.e., the new classes considered more likely from the current network for that exemplar *x*.
  This loss, called *margin ranking loss* is calculated as follows:

$$\mathcal{L}_{mr} = \sum_{k \in K} max\{m + \langle\bar{w}_k\left(x\right), \bar{\varphi}\left(x\right)\rangle - \langle\bar{w}_c\left(x\right), \bar{\varphi}\left(x\right)\rangle, 0\} \tag{7}$$

  where *m* is the margin threshold, K is the set containing the top-K new classes, $\bar{f}$ is the l_2 normalized features extracted by the current model, $\bar{w}_k\left(x\right)$ and $\bar{w}_c\left(x\right)$ are respectively the l_2 normalized weight vectors of the cosine layer of the ground-truth label *c* and the *k-th* new class.
  The main idea behind this loss is the following: if the difference of the cosine layer response between the ground-truth label *c* and one (or more) of the top-K new classes is inside the margin, a penalty is given.

Finally, the total loss is as follows:

$$\mathcal{L} = \frac{1}{|X|}\sum_{x \in X}\left(\mathcal{L}_{ce}\left(x\right) + \lambda\mathcal{L}_{dis}\left(x\right)\right) + \frac{1}{|E|}\sum_{x_e \in E}\mathcal{L}_{mr}\left(x_e\right) \tag{8}$$

where X is the set containing both new training data and the exemplars, E is the set containing only the exemplars and λ is a variable coefficient calculated as follows:

$$\lambda = \lambda_{base} \cdot \sqrt{\frac{|C_{new}|}{|C_{old}|}} \tag{9}$$

where $|C_{new}|$ and $|C_{old}|$ are respectively the number of new classes and old classes. The variable coefficient λ is introduced to scale properly the value of the distillation loss, weighting it accordingly to the imbalance between old and new classes. In our experiments we set $\lambda_{base} = 2$, $m = 0.5$, $K = 2$

3.2. KNN

In order to stress the importance to have a classifier that is able to change accordingly with the feature space, therefore to underline the difference between $iCaRL_{Hybrid}$ and $iCaRL_{NME}$, a KNN classifier is used. KNN, as NME, is based on the distance between the record to classify and the training records of different classes. The KNN set can be defined as :

$$Knn = \underset{I}{\operatorname{argmin}} \sum_{x_i \in I} ||\varphi(x_{test}) - \varphi(x_i)||$$
$$I \subseteq X,\ |I| = K, 1 \leq K \leq |X| \tag{10}$$

where X is a balanced set containing the union of exemplars and a subset of the current batch images.

Then a majority voting is applied to select the nearest class among the images in I. In order to select the hyperparameter K, after each training batch, we perform a cross validation on X. Regardless the step, using the following $K = \{7, 9, 11, 13, 15\}$, the *best* K is close to 11 or 13.

3.3. IL2M

An interesting strategy to build an incremental learning model is proposed by Beloudha et. al.[4], shifting from the use of a distillation loss to a statistic approach to avoid the effect of *catastrophic forgetting*. After each incremental step, some metrics are calculated, in order to adjust the predictions. These statistics are made on the outputs of each class, consequently, in order to make them fairly descriptive of a specific incremental step, they are computed when those classes have been just modeled. Therefore, it is possible to reduce the prediction bias due to the imbalanced dataset across different incremental steps by simply rectifying the prediction of the exemplar class $c \in X^{old}$ under the following condition:

$$p_c^r = \begin{cases} p_c \cdot \frac{\mu_c^P}{\mu_c^N} \cdot \frac{C^N}{C^P} & \text{if } pred \in X^{new} \\ p_c & \text{if } pred \in X^{old} \end{cases}$$

where P is the step in which the class *c* was learned, N is the current incremental step, p_c is the score prediction of the current model for the class *c*, μ_c^P and μ_c^N are the mean classification scores of class *c* in the steps P and N respectively, C^P and C^N are the model *confidences* computed by averaging the scores of all training images $x \in X$ available in the steps N and P respectively.
The first important thing to notice is that the rectification is applied only if the test image is predicted as one of the new classes just introduced in the current incremental step N, otherwise there is no risk in terms of prediction bias and the rectification is not necessary.
The main intuition behind the terms appearing in the rectification formula are the following:

- Since the exemplar class *c* was learned in the step P when all its training data are available, its mean prediction μ_c^P is very likely to be higher than μ_c^N, increasing the probability p_c^r compared to p_c.
- Since the model *confidence* overall the training data is different among the different incremental steps, the last term is introduced to re-scale properly the prediction score taking into account the influence of combining the prediction of models learned in different incremental steps.

3.4. Comparison between classifiers

In this subsection we compare the results between the above mentioned classifiers and the methods proposed by *iCaRL*:

- *Cosine normalization layer*: In order to evaluate this pipeline, it is interesting the comparison with $iCaRL_{Hybrid}$, because this approach tries to overcome the problems related with the imbalance of weights of the fully connected layer. The gap between new and old classes is strongly reduced, and also the bias on the first classes introduced by the aforementioned method is eliminated. Also in this case the NME strategy slightly over-performs.

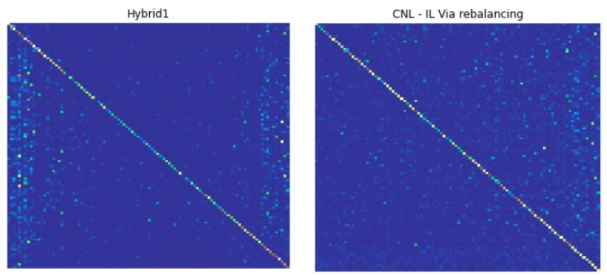

Figure 2. Confusion matrix of *Hybrid* and *CNL*

- *KNN*: KNN is less favourable then NME for four main reasons:
  1. The NME, calculating the mean of all training exemplars is much more robust to noise than KNN.
  2. In NME we have not to define an hyperparameter, therefore it would be easier to use and to adapt.
  3. NME is surely faster, since the distances are computed only with the centroids, while in KNN for each point to classify, the distance from all training points must be computed.
  4. NME allows the trick described in par. 2.4, therefore it handles properly unbalanced data, by summarizing them through the average. KNN is not safe with unbalanced data, therefore an undersampling phase must be provided in order to balance the number of data between new and old classes, in this way the classifier works with much less data and it can be less precise, especially on the new classes.

  Despite these drawbacks, KNN works much better than the hybrid solution, performing better in all batches, but as expectable NME over-performs.

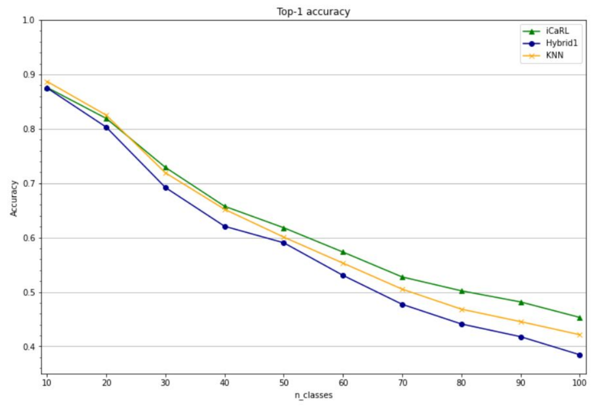

Figure 3. $iCaRL_{NME}$ - $iCaRL_{Hybrid}$ - KNN

- *IL2M*: This approach imposes a totally different view on the problem: if we divide incremental learning procedures in *distillation-based* and *parameter-based*, this method belongs to the second class. The advantage of this setting is that the network is trained using only a classification loss, in fact the only different computations are made after the training phase. On the other hand, in order to define precise metrics, there is the need to have a previous knowledge of the problem, so that the metrics would be well explained and robust. Another advantage is the computational cost, in fact the lack of a distillation loss make the training phase faster.
  As expectable this approach do not play the same role of a distillation loss, in fact the propensity of the network to predict in the new classes is not avoided properly using the proposed metrics.

Classifiers	Avg. Acc.
NME	**62.3%**
Hybrid1	58.3%
KNN	60.8%
IL2M	56.8%
CNL	61.9%

Table 1. Average accuracy over 10 batches

4. Ablation study for Losses

In order to understand deeply how the loss introduced by iCaRL works, we tried to use the same setting with different losses, analyzing the hinton loss, our modified version of MSE and a brand-new loss : Log distance loss.

4.1. Hinton Loss

Interestingly, Hinton et. al[5] proposed a different knowledge distillation, by using a temperate multi-class cross-entropy loss. Essentially, this is a standard softmax cross-entropy loss with *soft-targets* and a temperature parameter T. This loss can be written as follows:

$$\mathcal{L}_{Hinton} = -\sum_{c \in C^t} p'^{c}_{\theta^{t-1}} log(p'^{c}_{\theta^t}) \qquad (11)$$

$$p'^{(c)} = \frac{\left(p^{(c)}\right)^{1/T}}{\sum_{j \in C} \left(p^{(j)}\right)^{1/T}} \qquad (12)$$

Using a softmax introduces correlation between outputs, thus if an output probability is increased, the probability of other outputs must tend to decrease. Actually, in a logaritmic curve, moving probabilities towards 0 or 1 is deeply different, because the slope of the loss is completely unbalanced in the two cases. Therefore, in order to alleviate this effect, the temperature T introduced by Hinton moves the softmax probabilities making them closer in an area with an almost constant slope, thus rebalancing the penalties[fig.4]. This behaviour holds only if $T > 1$, otherwise the effect would be the opposite. Moreover, since we use a cross-entropy loss for classification, by setting $T = 1$ the contribute of the distillation would prevail, because multiplying by a *soft-target* distillation slopes are much higher and the aforementioned imbalance in penalties would be increased. In this experiment we set $T = 2$.

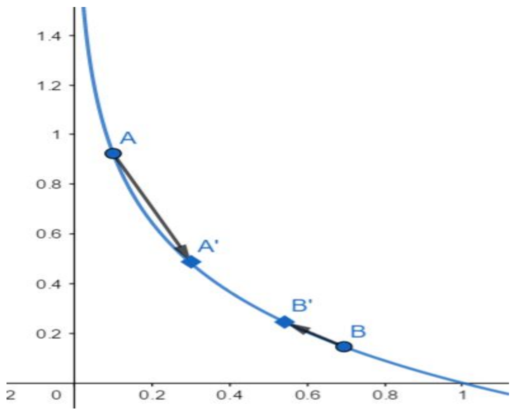

Figure 4. Behavior of Hinton transformation of the softmax, example movement for two points with initial softmax probability p_A = 0.1 e p_B = 0.7. In this graph the distillation loss is represented, with $p^c_{\theta^{t-1}}$ = 0.4.

4.2. LogDistance loss

The distillation loss proposed in iCaRL (eq.1) has its minimum in $p^c_{\theta^t} = p^c_{\theta^{t-1}}$, the function shape is variable with respect to $p^c_{\theta^{t-1}}$ and favors the changes of probabilities close to 0.5 (See par. 5.1). In order to stress the behavior of the distillation and to keep more stable old probabilities, we propose the *log-distance loss*, that has its minimum in $p^c_{\theta^{t-1}}$, but penalizes more heavily also small changes around this old probability and it has the same slopes in both the decreasing and increasing side, therefore it balances the effect of the direction in which probabilities change. Moreover it modifies the behavior around 0.5, because the shape of the curve does not change with respect to $p^c_{\theta^{t-1}}$, but just translate horizontally [fig. 5]. Consequently, this loss does not prefer moves around 0.5 as the BCE does.

The log-distance loss can be expressed as follows:

$$L_{dist} = -\sum_{c \in C^{t-1}} \log(1 - |p^c_{\theta^t} - p^c_{\theta^{t-1}}|) \qquad (13)$$

where $p^c_{\theta^t}$ is the output logit for class c, with network parameters θ at time t. Moreover when the *soft-targets* become hard-targets (0-1), this loss is exactly a BCE Loss, in this way it allows the use of the latter jointly as classification loss. The hypermeters are the same used for iCaRL.

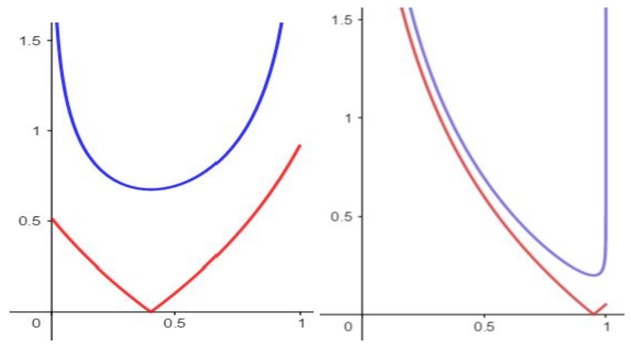

Figure 5. Comparison between BCE Loss(Blue) and log-distance loss(Red). In this graph the distillation losses are represented with $p^c_{\theta t-1} = 0.4$. and $p^c_{\theta t-1} = 0.95$

4.3. Modified MSE

In order to analyze a different behavior, in contrast to the Log Distance Loss, we propose a modified version of the MSE, that embed some practices used in the implementation of our custom loss, described in par. 5.1. The modified MSE can be written as :

$$\mathcal{L}_{dist}(p, y) = -\sum_{c \in C^{t-1}} (p^c_{\theta t} - p^c_{\theta t-1})^2 \cdot \\ \cdot (\xi {p^c_{\theta t-1}}^2 + \xi p^c_{\theta t-1} + 1) \tag{14}$$

where the first factor is a classic square distance distillation, while the second part make the function shape variable, with respect to the output logits of the previous classes. Specifically, ξ defines how flat the curve will be, the more flat is the curve, the more output probabilities are free to move away from the soft-target, since the given penalty would be quite low. With this loss we would revert the effect of the log-distance loss, by allowing more changes in old outputs, thus relaxing slightly the constraints imposed by the distillation loss. Particularly, the more the old score is close to 0.5, the more freedom it has to change, while the curve is much more pendent going towards zero or one [Fig. 6]. In our experiments $\xi = 2$.

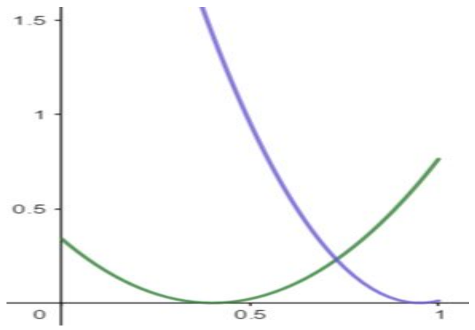

Figure 6. An example of the behavior of our modified MSE, when $p^c_{\theta t-1}$ = 0.4(Red) and when $p^c_{\theta t-1}$ = 0.95(Blue)

4.4. Comparison between losses

In this subsection we compare the results between the aforementioned losses with iCaRL.

- *Hinton Loss*: results related to this loss are very similar to $iCaRL_{NME}$. We experiment this loss with T=1, to understand its limits, as explained in par. 4.1. Actually, if $T = 1$, the accuracy is lower in each incremental step, demonstrating its effectiveness [fig. 7].

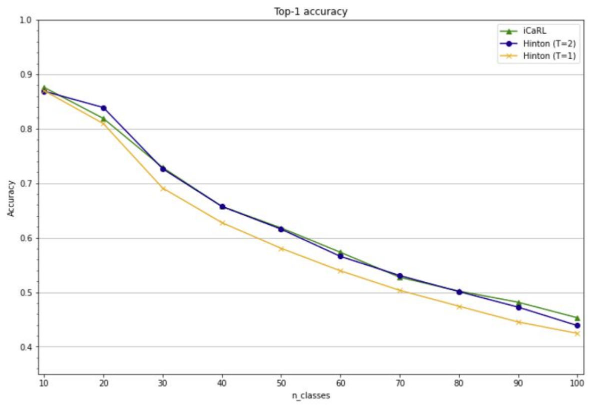

Figure 7. $iCaRL_{NME}$ - $Hinton_{T=2}$ - $Hinton_{T=1}$

- *Log-distance loss* : results are lower than the iCaRL baseline, on the other hand the changes of the accuracy scores on new and old classes is the expected one, in fact the gap between them is reduced by 2%, thus meaning that the network, in this setting, keeps more strictly the old knowledge. On the other hand the drop in overall accuracy is caused by a too strict constraint imposed by the distillation loss, thus not allowing the network to learn properly.
- *Modified MSE Loss* : This loss slightly improves the results of iCaRL. The effect of the relaxation with respect to the log-distance loss allows the network to learn properly the new classes and keep the previous knowledge, increasing slightly the overall accuracy in all batches.

Loss	**Avg. Acc.**
iCaRL$_{NME}$	62.3%
Hinton$_{T=2}$	62.2%
Hinton$_{T=1}$	59.7%
LogDistance	59.7%
MSE	62.3%
Our MSE	**63.0%**

Table 2. Average accuracy using NME classifier

5. Variations

In this section some variations of *iCaRL* are proposed. Such changes derive from a preliminary study of the algorithm and, especially, of the Binary Cross-Entropy Loss on which it is based1. In particular, it is proposed a custom loss function which enhances some characteristics of the BCE, but which presents greater versatility and allows for better results on the test set. Then, it was implemented a custom scheduling of the regularization term and a method that acts as a *constraint relaxation* for the distillation loss.

5.1. Towards our polynomial loss

The process of defining a new loss function starts from the definition of the main criteria that it must respect. Some of them derive from computational or mathematical reasons, others are necessary to have a proper behavior for each task. For example, for most optimization algorithms, it is desirable to have a loss function that is globally continuous, differentiable and that its second derivative is always greater than zero (strictly convex). However, for an incremental learning system, these criteria are necessary but not sufficient. In such case, it is crucial to have a constraint that allows to learn new classes, preventing the so-called *Catastrophic forgetting*. That constraint, in *iCaRL*, is the *Distillation loss*, on which the following is focus.

A distillation loss function works in addition to a standard classification loss and depends heavily on it and on the network to which it is applied. For instance, a function like the Hinton Loss (12) can not be used as a distillation loss with *uncorrelated* outputs (sigmoid for the last layer), because it reaches its minimum when all the outputs are equal to one and not when they are equal to the target. However, it works well when there is correlation between them (softmax for the last layer), because, in that case, the solution with all the *soft-max probabilities* equal to one is infeasible. In this work, a *sigmoid function* was used for the last layer, and subsequently the distillation loss must satisfy the following criteria:

- It must be globally continuous, differentiable and strictly convex. (1)

- It must have its only minimum in $P^c_{\theta t} = P^c_{\theta t-1}$ (the target). Matematically, it means that the first derivative must be equal to zero when $P^c_{\theta t} = P^c_{\theta t-1}$ only. (2)

Furthermore, it is possible to define a loss function without any bias (i.e. $Loss(x) = 0$ when $x = x_{min}$), but it has no effect on the minimization problem, so it is just imposed that the function is always greater than or equal to zero, to avoid negative values of the loss.

Once these criteria are defined, it is possible to move on to the study of the *BCE distillation loss* used in *iCaRL*, to understand its strengths and to replicate and stress them on a completely new loss function. A very interesting thing is that the behavior of the BCE loss changes as the target $P^c_{\theta t-1}$ changes. In particular, for any fixed $k > 0$, the farther away the target is from 0.5, the higher the increases of the loss is, when $|P^c_{\theta t-1} - P^c_{\theta t}| = k$. (Link to the mathematical proof : link).

This can also be easily demonstrated by plotting the function for different values of the target $P^c_{\theta t-1}$. Indeed, its slope is greater when the target is closer to zero or one and, in addition, it is more likely that $P^c_{\theta t}$ is closer to 0.5 than $P^c_{\theta t-1}$, because the slope of the function towards 0.5 is always lower than that on the opposite direction [fig. 8]. In informal terms, we can say that when the network is forced to modify something, in order to learn new classes, it prefers to change the outputs closest to 0.5, towards 0.5 (obviously, it modify the weights and not directly the outputs). Hence, it is important to understand if such behavior should be amplified or reduced.
During the training phase, the classification loss (BCE) tries to put all the outputs to 0 or 1. Therefore, the outputs closest to those values represent a condition of maximum certainty and, modifying them, the network could lose the ability to predict the old classes, especially if related to the images of the exemplar set. The outputs with values around 0.5, instead, are those for which the network has greater uncertainty. These differences, of course, are not so evident (all outputs are close to 0 or 1), but analyzing the results obtained with iCaRL after the first training step, it was noticed that the outputs related to the images belonging to a new class are, on average, closer to 0.5 than those related to already known class. Since the outputs closer to 0.5 have less significance, it may be a good idea to stress the behavior of the BCE loss described above (N.B. The LogAbsDistance (13) function does the opposite, to have a counter-test).

5.2. Polynomial loss: demonstration

As illustrated so far, the goal is to obtain a custom function that, at time t, penalizes the changes of the outputs with a target closer to zero or one, more than the others. There are several functions that can be used as a starting point. However, according to the Stone-Weierstrass Theorem: *"every continuous function defined on a closed interval [a, b] can be uniformly approximated as closely as desired by a polynomial function."*. In this case, the input variables of the loss function are the outputs of a sigmoidal activation layer and consequently they are always in the range [0,1]. This means that, probably, a polynomial is the simplest and most versatile choice. For the sake of simplicity, a different notation will be used: the variable s indicate the soft-target $P^c_{\theta t-1}$, while the variable x indicates the new outputs $P^c_{\theta t}$. So, let's start with a function that is flat around its minimum and then impose that it satisfies the previous criteria. Such function could be:

$$L_D(x) = x^d, d = 2, 4, 6, ..., N_{even}$$

The larger the exponent of x is, the flatter the function is around its minimum. For this purpose, $d = 4$ could be a good compromise. But this function does not depends on the target s. So another term $q(s)$ must be added and it must not vanish in the first derivative. We will have:

$$L_D(x, s) = x^4 + q(s)x$$

For the criterion (2) the first derivative of $L_D(x, s)$ must be equal to zero when $x = s$. So, let's compute the first derivative and impose it to be zero when $x = s$:

$$\frac{dL_D(x, s)}{dx} = 4x^3 + q(s)$$

$$4s^3 + q(s) = 0$$

$$q(s) = -4s^3$$

Result:

$$L_D(x, s) = x^4 - 4s^3x$$

This function satisfies the second criterion but it is flat around the minimum when $s = 0$ and not when $s = 0.5$. Moreover, it is not symmetric with respect to $s = 0.5$, hence an horizontal translation is performed and the arguments are multiplied by 2:

$$L_D(x, s) = (2x - 1)^4 - 4(2s - 1)^3(2x - 1)$$

Now let's add a bias to always have a positive function:

$$L_D(x, s) = (2x - 1)^4 - 4(2s - 1)^3(2x - 1) + 3$$

Now, $L_D(x, s)$ satisfies the criterion (2) and it is continuous, differentiable and always greater than or equal to zero. However, performing the second derivative (the Hessian, in multiple dimensions), it is possible to see that it is convex, but not strictly convex (the Hessian is positive semi-definite). But it does not matter, because that constraint can be relaxed using the custom distillation loss in addition with a normal BCE for the classification part. Indeed, the sum of a convex and a strictly convex function is, in turn, strictly convex and so the total loss also satisfies the criterion (1). Therefore, multiplying the distillation part by a factor 1/4 to "level" its absolute value with the BCE function, the total loss is:

$$\sum_{c \in C^t} - \left(y^c \log\left(P^c_{a^t}\right) + (1 - y^c) \log\left(1 - P^c_{\theta^t}\right)\right) +$$

$$+ \sum_{c \in C^{t-1}} \frac{1}{4} \left((2P^c_{\theta^t} - 1)^4 - 4(2P^c_{\theta^{t-1}} - 1)^3(2P^c_{\theta^t} - 1) + 3\right) \tag{15}$$

Where $P^c_{\theta^t}$ is the binary probability (sigmoid) of the class c using the network at time t, y^c is the one-hot target, C^{t-1} is the set of classes known at time $t - 1$ and C^t is the set of new classes.

The figure 8 shows the trend of the function (only the distillation part) for different values of the target $P^c_{\theta^{t-1}}$. The differences between it and the BCE function are evident. They have a similar trend and absolute value, but the custom loss is more flat when the target is around 0.5 (as we wanted).

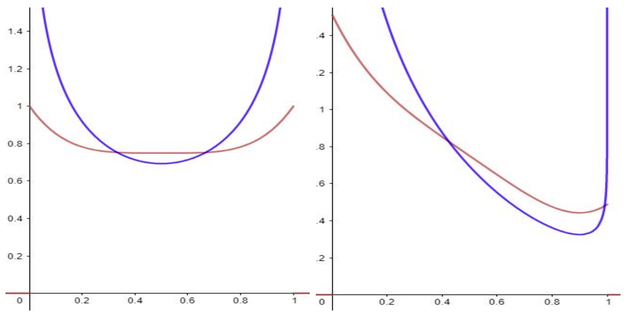

Figure 8. Comparison between BCE Loss(Blue) and our custom loss loss(Red). In this graph the distillation losses are represented with $p^c_{\theta^{t-1}} = 0.5$ (left) and $p^c_{\theta^{t-1}} = 0.95$ (right).

With this function it was possible to increase the results obtained with the BCE loss [9], but its advantages are not only related to the results achieved, but also to its versatility. The trend of this custom function can be easily modified by changing some parameters, without violating the previous criteria (this can not be done with a logarithmic function). Indeed, analyzing its coefficients, it can be noticed that they only depend on the polynomial degree, so it is possible to generalize that function in the following way:

$$L_D(P^c_{\theta^t}, P^c_{\theta^{t-1}}, d) =$$

$$\sum_{c \in C^{t-1}} \frac{1}{d} \left((2P^c_{\theta^t} - 1)^d - d(2P^c_{\theta^{t-1}} - 1)^{d-1}(2P^c_{\theta^t} - 1) + b\right)$$

$$b \equiv \textit{bias} = d - 1$$

$$d \equiv \textit{polynomial degree} = 2, 4, 6, ..., N_{even} \tag{16}$$

It represents a set of *polynomial distillation loss functions* and they all satisfy the defined criteria. By increasing the value of d, the behavior described above is more stressed, decreasing d, instead, it is possible to reduce it. With $d = 2$, each term of the distillation function has the same trend of $4(x - t)^2$ (MSE loss), with a bias when $t \neq 0$ and $t \neq 1$. Therefore, it could also be used for the classification part (it is strictly convex). Different polynomials were tested and each of them shows interesting features. However, all the attached results were obtained with the first equation, which has $d = 4$.

5.3. Forget-Constraint Relaxation

The results achieved with the custom loss are certainly encouraging and confirm that it is, probably, the right direction. By directly comparing the BCE and the polynomial loss functions [fig. 8], it can be seen that the latter is actually a relaxation of the former and it leads to an improvement.

The distillation loss is nothing more than a constraint that leads to a compromise between what is learned and what is forgotten, but it does not allow the model to learn without forgetting. It allows only *to forget* when it is strictly necessary. What can be deduced, from the previous results, is that it is probably too binding. The aim is to impose that the network at time t does not change too much compared to the network at time $t-1$, but, perhaps, the *best network* at time t would be very different from what it was at time $t-1$. Since some exemplars are kept in memory (they reduce the risk of catastrophic forgetting), a possible enhancement is to further relax such constraint, increasing the degrees of freedom of the network. This time the goal is not to relax it only when the target is close to 0.5, but also when it is close to zero or one.

To do that, it is common to multiply the distillation part by a fixed weight lower than one. In this way, the network learns new classes more easily, but quickly forgets the old ones. This is appropriate if the aim is to give more importance to the learning part and not to improve the overall accuracy of the classifier. For this reason, the proposed approach is to *randomly set* some of the contributions of the distillation loss (L_D) *to zero*. Mathematically:

$$\sum_{c \in C^{t-1}} \frac{R^c}{p} L_D(P^c_{\theta^t}, P^c_{\theta^{t-1}})$$
$$R^c = \mathit{1,\ with\ probability} = p$$
$$R^c = \mathit{0,\ with\ probability} = 1 - p \tag{17}$$

For this work, only $p = 0.5$ was used.
With this approach, the total value of the distillation loss remains almost unchanged: when R^c is equal to 0, the contribution related to the class c is zero. When R^c is 1, instead, it is multiplied by $1/p$. With $p = 0.5$, therefore, on average half of the contributions are equal to zero, but the remaining half is multiplied by two, keeping the same proportion between the classification and the distillation loss. Unlike multiplying by a fixed weight, it does not give greater importance to the classification part, but it has an impact on gradients and on updating weights.

If R^c is zero, the output $P^c_{\theta^t}$ can take any value: it moves away from the target if this results in a decrease in the *Total Loss*, because its contribution will be zero in any case. But in the next or after a few epochs, R^c will be equal to 1 and its contribution will be doubled. So, if the changes made to the network are harmful, the distillation loss will force the system to return to its previous state (it prevent catastrophic forgetting), otherwise, $P^c_{\theta^t}$ stays away from the target.

The strength of this approach is that allows otherwise impossible changes, because each coefficient (R) could be equal to zero for many consecutive epochs. Hence, a greater number of possible solutions is explored, and this leads to very interesting results:
Not only does the average accuracy further increase [tab. 3], but it often happens that *the accuracy of some old classes* increases over time [fig. 10].

5.4. Incremental Regularization

For the training of a Convolutional Neural Network (CNN), it is important the tuning of its hyperparameters. They can be related, for example, to the network's architecture, to the optimizer or to the regularization method. However, the *best set of hyperparameters* for each task, depends on a lot of factors such as the weight initialization and the data. In an incremental learning task, these factors change over time. At the beginning, the network is randomly initialized with values following a Gaussian distribution and the data set contains only 10 classes. In the following steps, instead, only some weights related to the fully connected layers need a random initialization and the data set is different. So, how can the hyperparameters remain the same?
Obviously, one option is to find a set of values that works well at all training steps, but it is really unlikely that it is always the best one. So, a simple solution would be to tune them every time new classes have to be learned. This approach is certainly effective, but it is slow. On the other hand, finding a method for the scheduling of all hyperparameters is impossible, but some mathematical observation can be made concerning the *regularization term*.

In this work, as regularization, the *L2 penalty* was used. It adds, to the objective function, a penalty equal to the *sum of squared value of the coefficients* and it will force the weights to be relatively small.

$$L = L_0 + \frac{\lambda}{2} \sum_w w^2 \tag{18}$$

Therefore, using the SGD optimizer, the weights will be update in the following way (leaving out the term related to the momentum, since it does not interfere with the following considerations):

$$w_{k+1} \leftarrow w_k - \eta \frac{\partial L_0}{\partial w} - \eta \lambda w \tag{19}$$

$$\eta \equiv \textit{Learning Rate}$$

$$\lambda \equiv \textit{Weight Decay}$$

At the first training step (10 classes), when the network does not contain any previous information, the objective function is equal to the classification loss function and its gradients are high (it is far from its minimum). So, the weight decay must be, in turn, relatively high, otherwise $\partial_w L_0 >> \lambda w$ and the regularization term would be negligible compared with the other.

At the following steps, instead, new classes must be learned and the total loss becomes $L_{tot} = L_{clf} + L_{dist}$. When the number of the old classes is much greater than the number of the new ones, the contribution of the distillation loss is greater than the classification one and so *the global minimum is close to that of the distillation part*. But the training phase, and so the minimization problem, starts exactly from that minimum $(minL_{dist})$. This means that the higher the ratio between the number of old and new classes, the closer the starting coefficients are to the best ones. Intuitively, if we have already learnt M classes, the addition of only N new ones, with $M \gg N$, does not perturb the system and so the new global minimum is close to the previous one. In mathematical terms, the gradients take on smaller values, on average.

Looking at the equation (19), if the gradients become close to zero the contribution of the regularization term becomes dominant and it will force all the network's coefficients to assume lower values. However, when the number of old classes is high, the aim is to change the network as little as possible and not to force the weights to decrease.

To balance these harmful effects, the solution is an advanced scheduling of the Weight Decay (λ):

$$\lambda_{t+1} \leftarrow \lambda_t \frac{\sigma - 1}{\sigma^2 - 2} \tag{20}$$

$$\sigma = \frac{\text{Total number of classes}}{\text{Number of new classes}}$$

where t is the incremental step.

This equation derives from an heuristic approach.

5.5. Comparison between our strategy and iCaRL

Finally, we will compare the results between iCaRL and our three components: *Polynomial loss*, *Forget-Constraint Relaxation*, *Incremental Regularization* , underlining also how the last two components suits with the strategy proposed by iCaRL.

- *Polynomial Loss*: The strength of our custom distillation loss is to deal better under uncertainty, allowing more freedom but at the same time keeping strictly the overall knowledge.

  In fact, as the data demonstrate, the more new incremental steps are added, the more the difference in accuracy between our custom loss and iCaRL is evident.

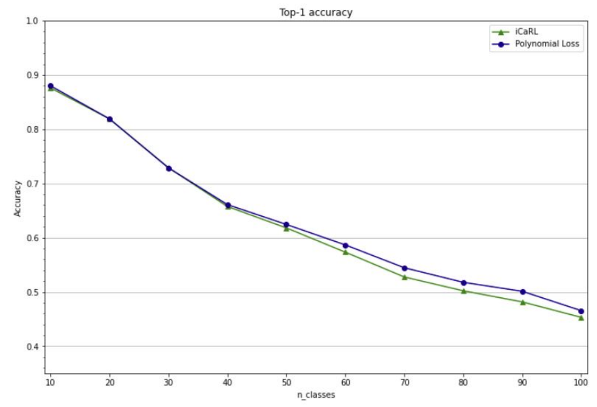

Figure 9. Top 1. accuracy $iCaRL_{NME}$ - $PolynomialLoss$

- *Forget-Constraint Relaxation*: By relaxing some constraints imposed by the distillation loss, we allow freer movements of the weights of the old classes, thus exploring widely the solution space and reducing the risk to be stuck in local minima.

  A very interesting aspect of this approach is that not only the overall accuracy increases, but sometimes also the accuracy of classes learned so far increases, as shown in the following figure:

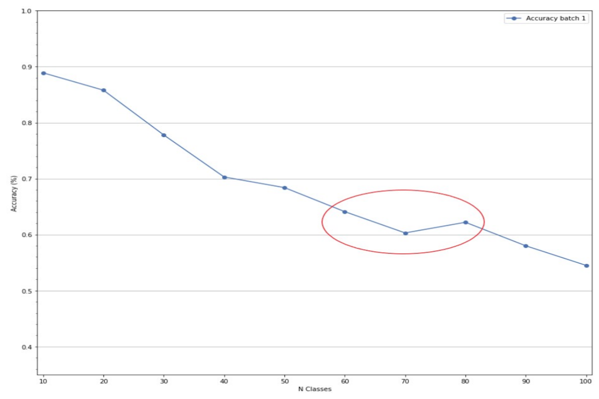

Figure 10. Top 1. accuracy about the first batch of 10 classes over the 10^{th} incremental steps

- *Incremental Regularization*: By introducing an incremental regularization scheduling the performances are still better, being able to overperform each new step compared with iCaRL, even if in the last steps the accuracy converges around the same value due to a lack of a consistent number of exemplars.

  Varying the regularization term is crucial in an incremental setting. We noticed that the proposed value of the regularization term was too low for the first groups of classes and actually, increasing it, improved the accuracy score on the first batches. On the other hand such behavior is not replicated along the steps, therefore maintainig stable $\lambda = 10^{-4}$, will reflects again the effect of *catastrophic forgetting*, as shown in the following plot:

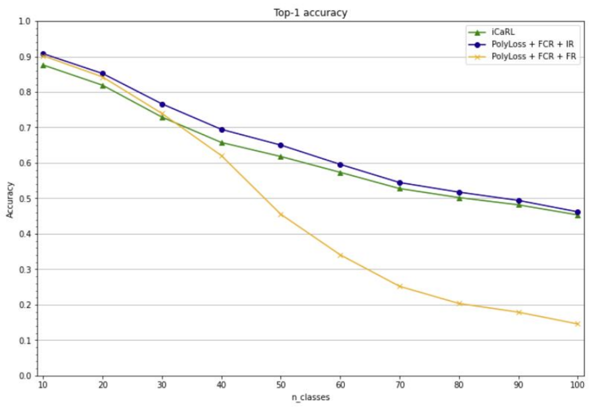

Figure 11. Top 1. accuracy: comparison between $iCaRL_{NME}$, our strategy and our strategy with a fixed regularization term.

The proposed solutions are very flexible and they can be easily added to every strategy.
In this table, we show how *Forget-Constraints Relaxation* and *Incremental regularization* improves consistently both the performance reached by $iCaRL_{NME}$ and $iCaRL_{Hybrid}$, but if combined with our *Polynomial loss* overperforms. A remarkable result is the one obtained with our strategy and predicting with the FC layer, in fact, beside increasing the average accuracy, our strategy reduces consistently the aforementioned issues introduced by $iCaRL_{Hybrid}$.

Strategy	Avg. Acc.
iCaRL$_{NME}$	62.3%
PL$_{NME}$	**63.3%**
iCaRL + FCR$_{NME}$	62.7%
PL + FCR$_{NME}$	**63.8%**
iCaRL + FCR + IR$_{NME}$	64.6%
PL + FCR + IR$_{NME}$	**64.9%**
iCaRL$_{Hybrid}$	58.3%
PL + FCR + IR$_{Hybrid}$	**62.5%**

Table 3. Average accuracy

6. Conclusion and future work

Our work aimed to study and deepen the *distillation-based* incremental learning, trying to understand if this setting was too much constrictive for a DNN. Therefore we create a strategy over the normal knowledge distillation, adding some plug and play modifications, not altering the global architecture. Actually we found that relaxing the constraint imposed by the distillation loss, performances become higher, on the other hand, they approaches to iCaRL going further in the incremental steps. The main reason is the lack of exemplars in the last batches and the consequent imbalance in training data. Hence, one of the interesting future works is to concentrate on these last batches and to understand how to balance the effect of the lack of exemplars. Moreover, we would like to analyze how a probabilistic approach like FCR (par. 5.3) can be more effective, with the adoption of other ways to decide which gradients to prune. Finally, also a more precise weight decay scheduling function could be used. Our code is available to : `https://github.com/wAnto97/Incremental_Learning_MLDL`

References

[1] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “icarl: Incremental classifier and representation learning,” 2016.

[2] Z. Li and D. Hoiem, “Learning without forgetting,” 2016.

[3] S. Hou, X. Pan, C. C. Loy, Z. Wang, and D. Lin, “Learning a unified classifier incrementally via rebalancing,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[4] E. Belouadah and A. Popescu, “Il2m: Class incremental learning with dual memory,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 583–592, 2019.

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