Incremental Learning in Image Classification

Francisco Javier Sanguineti
s279265@studenti.polito.it

Cristina Ojer
s279350@studenti.polito.it

Machine Learning and Deep Learning 01TXFSM - A.A 2019/20 -
Data Science and Engineering Master's Degree - Politecnico di Torino

Abstract

Frequently, it is thought that the data for training different tasks is all available at the beginning. When working in real-world applications where data arrive over time, and previous data may no longer be available, it is not a good solution to use normal deep learning architectures because it exhibits a critical drop of performance due to catastrophic forgetting when they are required to incrementally learn new tasks, therefore, different learning techniques are required.
In this report are explained and discussed the experiments performed while dealing with an image classification problem on CIFAR-100 data set. The goal of this project was to implement a robust image classifier facing an incremental learning process, that is, to build a system capable of increasing the knowledge of different classes over time.

1. Introduction

Incremental Learning systems in image classification provide an opportunity to add knowledge to an already trained classifier so that this system can recognize many other image classes on its classification task.
Given an image classification system that is capable of recognizing up to a given number of different image classes, we are interested in re-training this network with more new data regarding to other image classes, and making this system able to recognize these new classes over old ones and vice versa.
In this project, we implemented several models and our source code and further data are available at `https://github.com/Frannx1/IncrementalLearningProject`.

1.1. The data

In this work, we make use of the CIFAR-100 dataset [4]. It consists of 60000 colour images belonging to 100 classes, with a balanced distribution of 600 images per class. Each image is 32x32 pixels and is labeled with a single class. A prior split of the data is given for training and testing, there are 500 training images and 100 testing images per class.
Recently, it is used as an incremental learning benchmark, where the classes are split in many batches of classes, the learning procedure consists on train in one batch of classes at a time, and evaluate on all classes seen up to moment. This benchmark was introduced by Rebuffi *et al.* [8] with batches classes of 2, 5, 10, 20 and 50 classes, but in our case, we will particularly use the batch size of 10 classes. The groups of classes are randomly selected in each training.

2. Related Work

Recently, incremental learning becomes an active topic due to the success of deep learning. Deep learning achieves great results with respect to learning a specific task, but the same does not happen when seeking to expand its use in incremental learning. The majority of works are trying to improve the performance of the models answering to a simple question: how to not forget the previous knowledge. In order to overcome this limitation that the current convolutional neural network models possess, different mechanisms were proposed, we will mention the ones that we consider most significant for this work below.

2.1. Knowledge distillation

This method is an effective way to transfer knowledge from one network to another. It is first introduced to incremental learning in *Learning without Forgetting (LwF)* [5]. The basis of this method is very simple, it considers the network before being trained in another new task as a teacher of the network that is currently being trained.
In practice, it can be done by saving the outputs on the new training data from the previous network (the network before training in a new task) and then use them to compute a loss with the outputs from the current network. This loss is known as *distillation loss*, provides a great improvement and is widely used in many works.

2.2. Representation learning

Representation learning has become a field in itself in the machine learning community. In the classical literature on neural networks, for example [1, 2], one can already find the first attempts to learn data representations in the incremental learning paradigm. Even more, the problem of catastrophic forgetting, that is, the phenomenon of a training neural network with new data that caused it to overwrite (and therefore forget) what it learned in previous data, was particularly described in the late 1980s by McCloskey $et\ al.$ [6].
What was most recently introduced, a big improve by Rebuffi $et\ al.$ [8], is to adopt the principle of rehearsal: to update the model parameters for learning a representation, using not only the training data for the currently available classes, but also a subset of images belonging to previous known classes. These subsets are known as *exemplars*. In that work, this technique was combined with the aforementioned distillation to prevent information on the network from deteriorating too much over time.

2.3. Learning with a fixed data representation

In a incremental learning context, a key challenge is to design a classifier that is capable of categorize new classes at any time during the training phase without requiring access to all training data seen so far. This is called fixed data representation, the capability of extend the known classes could be carry out by a *k-nearest neighbor*, but it would need to access to all the training data.
Other approaches where presented as the *nearest class mean* by Mensink $et\ al.$ [7]. It is an alternative to the common procedure of classification, proposed to strength the concept of representation learning. For the evaluation task of the incremental system, accuracy is no longer computed by taking new data and outputting it directly to the network for later comparing it to the real labels. Instead, new images are compared to the representative centers of the previous learned class exemplars (used during the training task for the last mentioned representation learning strategy). The outputted label of this classification method will be the same as the mean-exemplar from which the new image is closer to (is more similar to).

3. Baselines

In order to have some baselines to compare all the different learning models, we will establish common parameters and a convolutional network architecture configuration. Regarding the network architecture, and considering that the results may differ due to the learning capabilities of different convolutional neural networks *(CNN)* architectures, it was decided that they all share the same base model, which is the ResNet [3] architecture. Specifically, an adaption for input image size of 32x32 with a deep of 32 layers denoted as $resnet32$. Layers before the last fully connected classifier *(FCC)* layer are known as *feature extractor*. In order for the network to classify, in an incremental learning context, more and more classes the last fully connected layer is expanded at the beginning of each training in a new group of classes.
When talking about training algorithm hyperparameters, we followed the same configuration proposed on iCaRL's work [8]: initial learning rate of 2.0 reduced by a factor of 0.2 on epochs 49 and 63 (over a total number of 70 epochs), weight decay of 0.00001 and an image batch size of 128 for every epoch.
As training loss we used the *Binary Cross Entropy Loss* provided by $PyTorch$, as classification and distillation losses. The decision of why we consider this loss in all our baseline models was due to the fact that, knowing that the different types of loss such as *Cross Entropy Loss* for the classification, also the proposed loss in LwF [5], are similar to the one we used, by always using the same allows us to achieve similar results with each model, without having to perform a grid search of the hyperparameters that performs best for each particular model. Then, other combinations of losses were used, and their effect will be analysed on forward points of the report. Finally, results of the carried out experiments, which are displayed on graphs throughout the report, were created taking the mean and standard deviation over 3 training executions on the model in each moment we're discussing about, in an attempt to obtain more generalized results.

3.1. Starting point: no strategy

We started on a basic fine-tuning baseline implementation of the neural network classifier. This method consists of a sequential training of the model in each group of classes. Every time that the model is trained in a group it is evaluated following the benchmark protocol trying to minimize *Binary Cross Entropy Loss* for the gradient descent. Then, the last fully connected layer is expanded to support more output classes, and another training or fine tuning starts with the next group, so until finish.
As appreciated in Figure 1, a drastic drop in accuracy is evident as the number of classes learned increases. This effect is the so-called "Catastrophic Forgetting" first recognized by McCloskey and Cohen [6]. The main problem we can think about here is that, in every training for new batch of classes, the algorithm does not care about previously learned features and "overwrites" all that knowledge while it learns new classes.

3.2. Learning without forgetting

The next step was to try to overcome the downside effect of Catastrophic Forgetting. Trying not to forget previously

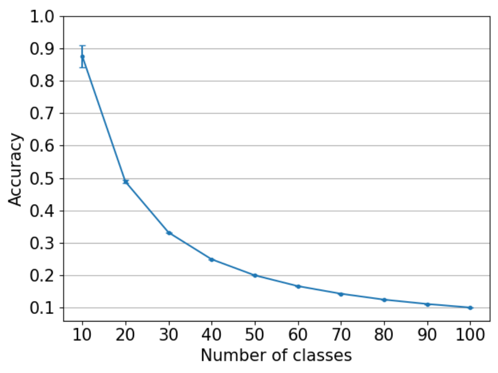

Figure 1. Accuracy (averages and standard deviations) of the no strategy model.

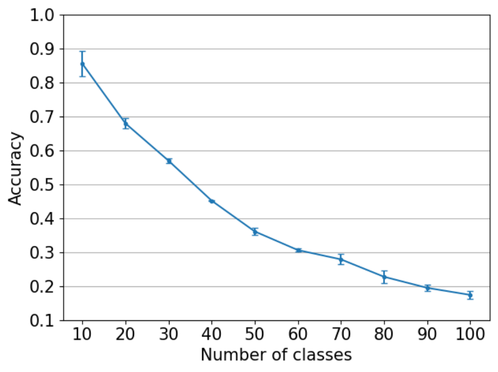

Figure 2. Accuracy (averages and standard deviations) of the LwF model.

learned tasks while the system is able to learn new ones. To achieve it, we use the Learning Without Forgetting strategy *(LwF)* [5], which introduces the concept of knowledge distillation presented on Section 3.3. Here, a different loss in the training algorithm is introduced. This loss tries to ”keep previous task” knowledge in the network at the same time new data arrives. This is possible by preserving the network trained in the last task, then, during the next training in a new group, both outputs are computed, those of the current network and those of the preserved network. This loss optimizes the new task and preserves responses on existing tasks from the original network.

Comparing the Figures 1 and 2, we can notice that this strategy considerably mitigate the Catastrophic Forgetting by observing a slower rate of decline in the evaluation accuracy.

3.3. iCaRL

In another attempt to further improve network performance during incremental learning, in this section we will analyze iCaRL: Incremental Classifier and Representation

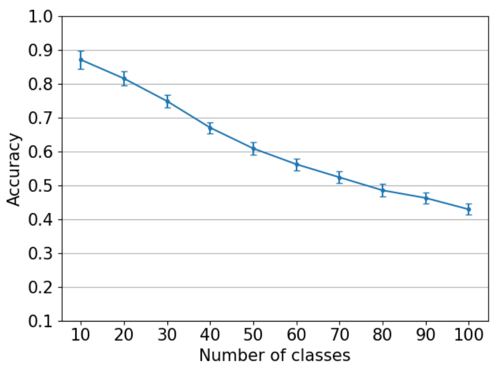

Figure 3. Accuracy (averages and standard deviations) of the iCaRL model.

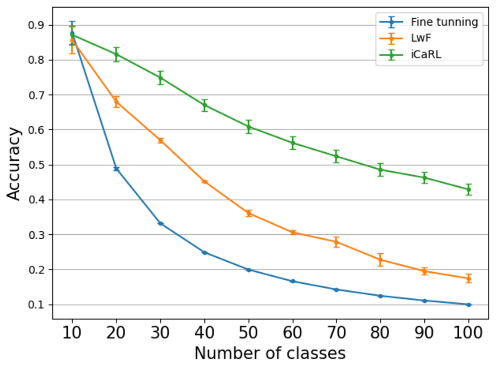

Figure 4. Accuracy (averages and standard deviations) of the 3 different models.

Learning [8], which introduces a new strategy that is considerably superior from the previous LwF approach in Section 3.2.

This strategy introduces a few but very significant computation differences: i) When training with new classes, also a set of images of all the previous known classes is sampled (exemplar set) with which the network is also trained and loss is gonna be computed. How these exemplars (which finally are class representatives) have been chosen is also a discussion topic in the next points. ii) NCM Classification algorithm: the outputted label for a given image will be the one belonging to the exemplar representative mean from which this test image is closer to.

As indicated in the Figure 3, by introducing this new ideas the performance of the classifier is clearly better than in the first mentioned strategies.

On this last graph from Figure 4 we can easily compare all of the three learning approaches performances while increasing the number of known classes. This curves have been obtained computing the mean of 3-running executions for each of the models.

4. Ablation study

In this section we present some variations over the proposed implementation on iCaRL's paper to analyse and study if we can improve this incremental learning classifier performance.

4.1. Different loss functions

Loss functions are a key part of any machine learning model: they define an objective against which the performance of your model is measured, and the setting of weight parameters learned by the model is determined by minimizing a chosen loss function.
To clarify, in each model we consider two types of losses: the *classification loss (L_c)* and the *distillation loss (L_d)*. The first one is in charge of learning the current group of classes, therefore, its inputs are only the outputs and objectives related to these classes. On the other hand, the second is responsible for preserving prior knowledge, and its inputs are the outputs of the networks, the current and the previous, but limiting them only to the old classes.
We experimented combining different losses during the training process to see their effect and if we were able to achieve better results:

- **Cross-Entropy loss (CE):** It is used to quantify the difference or distance over two probability distributions, which in this case are, the output of the network forwarding for the batch of images and their real classification labels.

$$L_{ce} = -\sum_{i=1}^{|\mathcal{C}|} y_i \log(p_i) \tag{1}$$

  where $\mathcal{C}$ is the set of all observed classes so far, y is the ground-truth label and p is the corresponding predicted class probabilities.

- **Binary Cross-Entropy loss (BCE):** Applies a sigmoid function over the network outputs before computing the previous mentioned cross-entropy loss. This makes our multi-label classification problem a binary classification problem.

$$L_{bce} = -\sum_{i=1}^{|\mathcal{C}|} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{2}$$

- **Mean Square Errors loss (L2 or MSE):** This loss function minimizes the squared differences between the estimated and existing target values.

$$L_{l2} = -\sum_{i=1}^{|\mathcal{C}|} (y_i - p_i)^2 \tag{3}$$

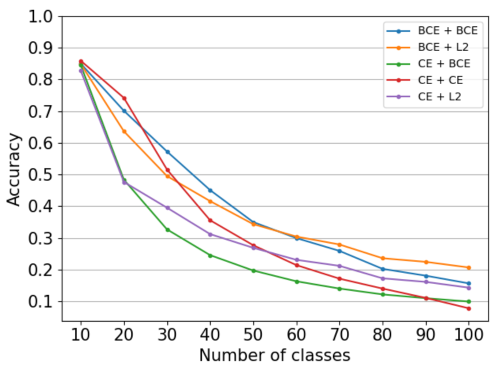

Figure 5. Accuracy (averages) of the LwF model with different losses.

  In our case, we only used the L2 loss as a *distillation on intermediate feature space loss*, which means that it is computed between the output features of the current and the old networks. This choice is due to the fact that the output layer is not anymore a classification layer but instead just an internal stage where the output should be kept close to the previous one in, e.g., L2-norm.

The overall loss we aim to minimize, is computed as the sum of both losses of the proposed combination. However, each of the contributing loss is multiplied by a λ factor in representation of how much each previously known and newly classes contribute to each epoch-time training classification in order to provide a balance of the contributions.

$$L = \lambda L_d + (1 - \lambda) L_c \tag{4}$$

where the scalar λ is computed as $\frac{n}{n+m}$, where n and m are the number of old and new classes respectively. Note that when training in the first batch of classes, the value of λ is 0 and, at the other extreme, λ reaches 1 when $n >> m$, indicating the importance to maintain the old classes.
When facing the experiments we had to fine-tune some hyperparameters such as the learning rate, because the previously proposed initial learning rate of 2.0 didn't fit well in some of the combinations. We worked out with the following configurations, where the first loss represents L_c and the second represents L_d:

- BCE + BCE: initial LR = 2.0
- BCE + L2: initial LR = 2.0
- CE + BCE: initial LR = 0.5
- CE + L2: initial LR = 0.5
- CE + CE: initial LR = 0.1

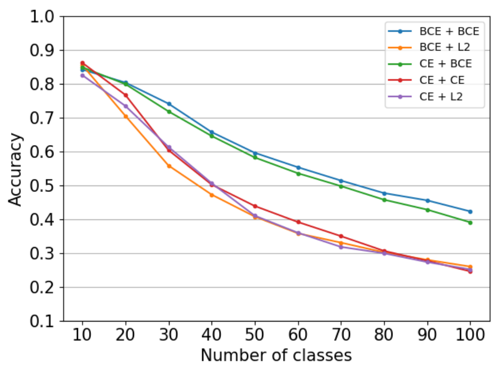

Figure 6. Accuracy (averages) of the iCaRL model with different losses.

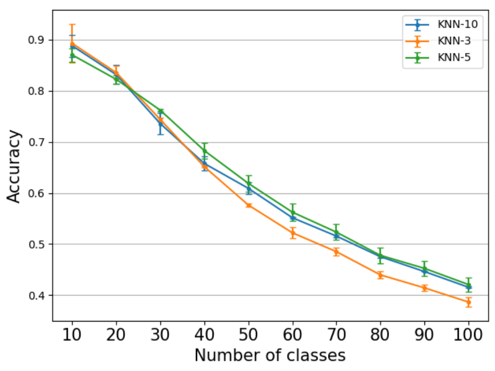

Figure 7. Accuracy of the iCarl model with KNN-3, KNN-5 and KNN-10 classifier approaches.

The graphs in Figures 5 and 6 show how each model performs while the number of known classes is incremented. Figure 5, where Learning Without Forgetting results are displayed, shows that using different losses doesn't avoid accuracy from decaying, even from the first steps, and despite the slight differences that can appear between all performances. All curves draw a similar path which means that previously learned classes are being forgotten step by step.

On figure 6 appear more differences when using different combination of losses. Non of the cases do better than the first implemented approach, which uses BCE + BCE loss for both training and distillation losses, and discussed before on Figure 3. Taking notice of the green curve, where CE loss is used as training loss instead of BCE, we want to remark how the accuracy decay occurs in a similar way the first approach do (the curve is similar, even if it remains always below the blue). Red, violet and yellow curves (CE+CE, CE+L2 and BCE+LS) suffer a greater accuracy decay from the very beginning. That means they're not preserving knowledge and the selected distillation loss is not helping to not to forget previous learned classes.

From these experiments on iCaRL network, we can conclude that our classifier works better taking as distillation loss BCE loss than any other of the proposed ones.

4.2. Trying with different classifiers

In this section we attempt to test our network modifying it by taking use of different type classifiers and see their effect on the image classification task.

The network architecture baseline remains the same for each of the cases. However, in each case of the different type-classifiers, the classification function *classify* is modified in such a way the predictions are now made using the *sklearn* provided class for that classifier in the KNN and SVC cases, and the self-implemented *CosineSimilarityClassifier* class for the third experimented case. The applied classifier is re-trained after each training procedure of the convolutional neural network for each batch of 10 classes. The training data for the new-used classifier will be the exemplars.

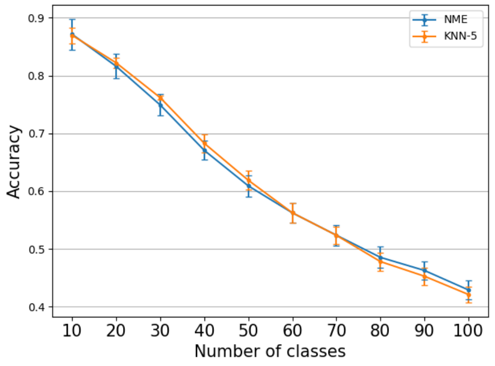

Figure 8. Accuracy of the iCarl model with KNN-5 classifier.

4.2.1 KNN classifier

To begin with, we experimented with a KNN classifier for the evaluation task of the incremental learning network, fine-tuning the model with different values of k. The best approach was obtained with k=5, as shown in Figure 7.

Then, we selected and later compared the best performing model to the last implemented iCaRL learner based on NME classification.

Figure 8 shows this comparison of the performance of the incremental classifier using the KNN algorithm for classification of new test images, against the iCaRL model with NME classification. The curve regarding the KNN classifier remains slightly below the one applying NME for classification, then the NME strategy for classifying new images gives small better results, but these don't seem to be very significant.

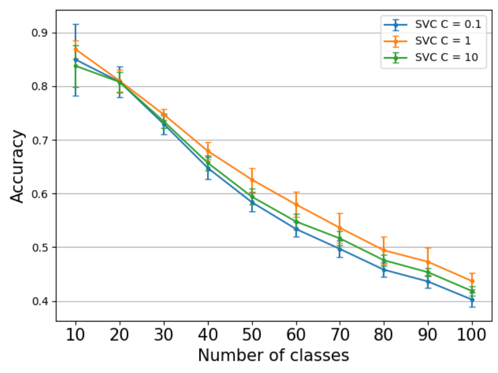

Figure 9. Accuracy of the iCarl model with SVC with RBF kernel for classification task (for different values of C).

4.2.2 SVC with RBF kernel

For our second type-classifier experiment, we tried out with a SVC with RBF kernel for the evaluation task of the incremental learning network. A SVC is a classifier trying to maximize the margin between classes to ensure the most secure classification. But this margin is hard to compute when dealing with non-linear problems, as are most of image classification problems, where we can find many outliers. That's why we decided to use SVC with RBF kernel. This maps the pixels into a higher dimension space and facilitates the computation of the class separation boundary when dealing with non-linear problems, as it is our case.
As with KNN, we first deal to fine-tune C parameter of this particular classifier. Setting C means setting the cost of miss-classification, that is, smaller C is, more mistakes are permitted by the classifier, and the larger C is, the classifier becomes more strict. The obtained results are presented in Figure 9.
Since the hyper-parameter γ defines the sharpness of the RBF distribution, we set it by *scalar* configuration provided by the *sklearn* library, in which it is calculated as $\gamma = \frac{1}{n_{features} * Variance(X)}$.

Following the same procedure, we selected and later compared the best performing model to the last implemented iCaRL learner based on NME classification. This results are displayed on Figure 10. In this case, the performance is almost the same as for iCaRL with NME classification strategy.

4.2.3 Cosine Similarity classifier

Our third attempt analysing different classification criteria was to implement a cosine similarity classifier. Cosine Similarity is a measure often used to compute the closeness of two vectors, given by the angle between both of them. In our case, we are interested in computing the cosine between two feature vectors, one of them the new image to classify and the other referencing one of the exemplar means. If the cosine result is equal to 1, then the angle between both feature vectors is 0, therefore we are talking about similar features. If the cosine result is equal to 0, then both feature vectors are completely opposite (they point on opposite directions) so they have nothing to do the one with the other. The more the cosine result is closer to 1 (in a range of [-1,1]) the more similar both feature vectors will be.

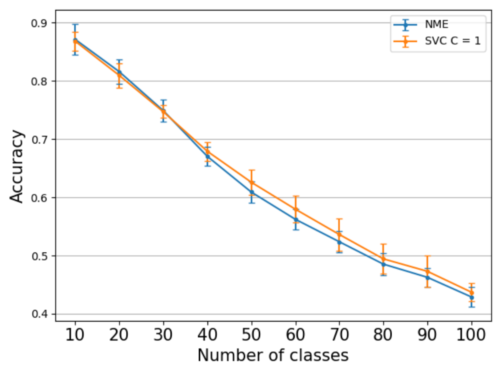

Figure 10. Accuracy of the iCarl model with SVC classifier and iCaRL wiht NME classification.

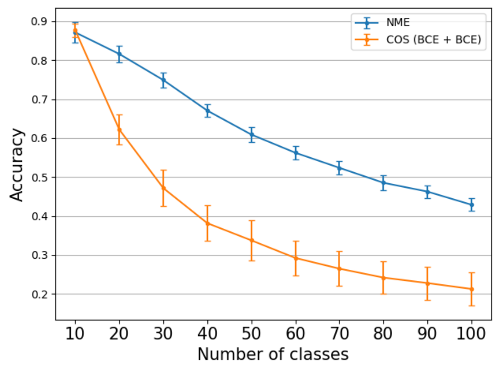

Figure 11. Accuracy of the iCarl model with Cosine similarity classifier.

We implemented this classification strategy over the iCaRL network. At first, results seem to be quite bad, consequently, we tried to modify this model configuration using another distillation loss (substituting the already implemented BCE). As the trainer goal is to maximize the cosine between the outputted feature vector of the current network with the ones from the previous network, we developed a loss class called *MaximizeCosineSimilarityLoss* that satisfies this requirements. It extends the *CosineEmbeddingLoss* class, provided by *PyTorch*, which computes the cosine similarity between two inputs vectors, and depending on a parameter it can manage to minimize or maximize it.

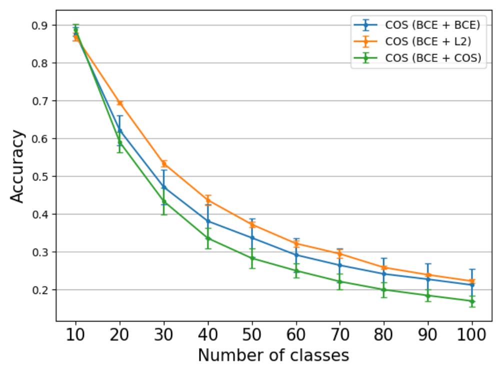

Figure 12. Accuracy of the iCarl model with Cosine similarity classifier using different loss functions.

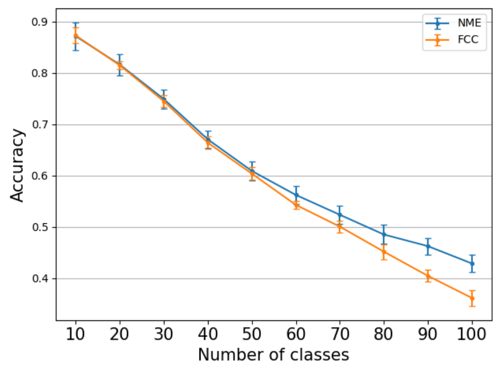

Figure 13. Accuracy (averages and standard deviations) of the Hybrid 1 and iCaRL models.

The yellow curve on figure 11 reproduces the performance using as classifier the cosine similarity procedure and using as distillation loss the one according to this strategy (not BCE). It performs considerably worse than iCaRL with NME.

Figure 12 shows the improvement of the iCaRL model with cosine similarity classifier by using as distillation loss a L2 loss approach.

4.2.4 Common sequential classification

We reproduced the iCaRL's network classification performance laying aside the NME classification strategy and outputting the labels for new images forwarding them sequentially through the built network. This approach is proposed on iCaRL's paper [8] as *Hybrid1* experiment.

Results shown on Figure 13 reveal that the *hybrid1* model's capability to not to forget previous knowledge is worse than the first implemented iCaRL approach. Up until 50 known classes the accuracy remains the same, that is, both models are capable of learning new classes similarly, but after that batch, the NME classification leads to a better preservation of the previous learned classes features.

Classifier	Mean accuracy
SVC-1	0.6250
iCaRL	0.6178
KNN-5	0.6192
KNN-10	0.6125
SVC-10	0.6042
iCaRL hybrid1	0.5961
KNN-3	0.5946
SVC-0.1	0.5945
LWF	0.4050
CF	0.2789

Table 1. Implemented classifiers and obtained mean accuracy for each.

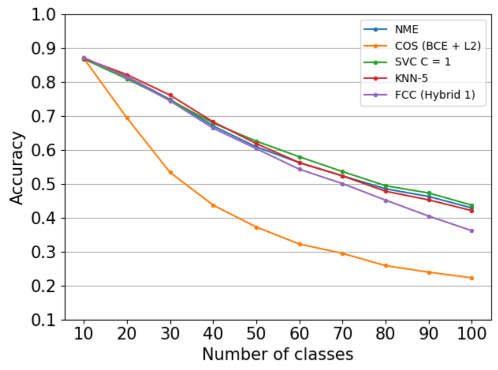

Figure 14. Accuracy (averages) of all presented models.

4.2.5 Best results

Table 4.2.5 displays, from top to bottom, the mean accuracies obtained after processing all classes from CIFAR-100 dataset for each strategy model in our study.

On Figure 14 we can compare the performance between all presented type-classifiers.

5. Our proposal

Our last ablation study consists on analysing the model's performance through a set of variations over the strategy of exemplar selection and classification criteria. After each training task for a batch of 10 classes, the exemplar sets of each of those classes are built considering the most significant training images on each class. The significance of an image is given by the closeness of this to the computed center feature vector of the class. This closeness can be measured in terms of a geometric distance over the feature vector of the image and the class-center feature vector.
The same idea can be applied when we want to classify

a new image in terms of NME classification. A geometric distance rules the definition of how close (how similar) the image is to the mean feature vector of the exemplars of each of the classes. Our proposal is to compare between the use of distinct geometric distances to select the closest features to the class-centers, that is, to set up different 'closeness' function definitions, then applied to both strategies, the most representative exemplars selection and NME classification. Up until this point we have implemented all of our model approaches (building the exemplar sets and classifying with NME) basing the closeness of a feature vector on the L2 distance.

The distinct geometric distances we experimented with and the obtained results are presented on the subsections below.

- **L2 (sum of squared difference) distance** This is the distance we have used until this point on every implementation of our incremental learner.

$$d_{L2} = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{5}$$

- **L1 (sum of absolute difference) distance**

$$d_{L1} = \sum_{i=1}^{n} |x_i - y_i| \tag{6}$$

- **Minkowski distance** The Minkowski distance is the generalized L_p-norm of the difference

$$d_{Minkowski} = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{7}$$

  Also L1 distance correspond to Minkowski with $p = 1$ value. We further tried with $p = 4$ and $p = 5$ to see what happens.

- **Chebyshev distance** The Chebyshev distance is the L_∞-norm of the difference, a special case of the Minkowski distance where p goes to infinity.

$$d_{Chebyshev} = \max_i |x_i - y_i| \tag{8}$$

- **Cosine distance** Cosine distance works on the same idea discussed on Section 4.2.3. It represents the angular distance of two vectors

We attempt to build one classifier for each of the distances, using a loss configuration of BCE + BCE, which worked best for out previous experiments through this project.

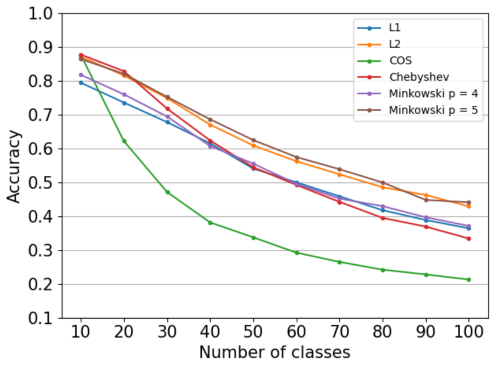

Figure 15. Accuracy (averages) of each of the models using different geometric distances as metrics of how close two feature-vectors are.

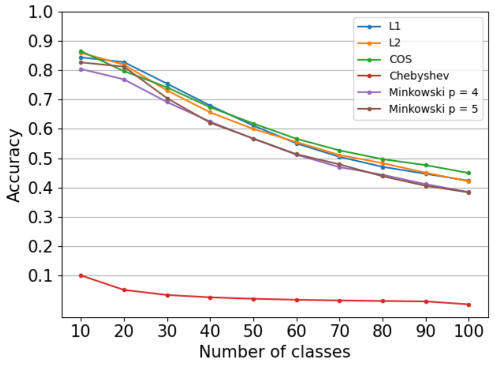

Figure 16. Accuracy (averages) of each of the models using different geometric distances, without normalizing feature vectors outputted by the network.

The results presented on Figure 15 do not get to improve L2 distance performance. As much, the model implementing Minkowski distance with p=4 achieved more or less the same accuracy.

In a further attempt to improve this results, we tested this variety of models that implement different geometric distances, without normalizing the output features obtained from the network, and also without normalizing the exemplar means. We thought that applying in all cases L2 normalization could be wrong.

Results on Figure 16 show up the new behaviours, slightly improved in every case except the one from Chebyshev distance. But the most significant change was made with the cosine distance, where accuracy grew considerably.

We realized we were making an important mistake normalizing feature vectors while using this metric (cosine similarity) for exemplar selection as well as for classification. In fact, this metric compares two vectors based on their direction and forgetting about their scale. The bad results

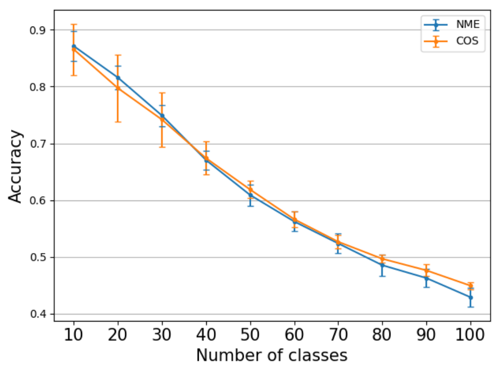

Figure 17. Accuracy (averages and standard deviations) of the CS selection and iCaRL models.

obtained in Section 4.2.3 were now improved. In Figure 17 we can see how the Cosine Similarity approach for exemplar selection and for classification retrieves almost the same results as iCaRL wiht NME does.

6. Conclusions

This work develops different strategies to overcome the incremental learning issues. We start from the Fine-Tuning base to have an lower bound, then, we implemented the well-known methods as LwF and iCaRL, using knowledge distillation and representation learning. We conducted further studies, exploring how different loss functions and classifiers affect its performance.

We proposed different ways to select the examples, and along with it, another way of selecting the class means closer to a feature extracted from an image, without making improvements.

Further studies regarding to the representation of the class means could be done as, for example by using a representation based on a set of class centroids, to obtain more flexible and non-linear classifiers. Another study could be, not only using the class mean, but also using the class standard deviation when calculating proximity.

References

[1] B. Ans and S. Rousset. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'Académie des Sciences - Series III - Sciences de la Vie*, 320:989–997, 12 1997.

[2] R. M. French. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 1176–1177. Morgan-Kaufmann, 1994.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research).

[5] Z. Li and D. Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016.

[6] M. Mccloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104–169, 1989.

[7] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric Learning for Large Scale Image Classification: Generalizing to New Classes at Near-Zero Cost. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *ECCV 2012 - 12th European Conference on Computer Vision*, volume 7573 of *Lecture Notes in Computer Science*, pages 488–501, Florence, Italy, Oct. 2012. Springer.

[8] S. Rebuffi, A. Kolesnikov, and C. H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016.