# Open World Recognition in Image Classification

Delleani Mattia
s288854

Girardi Enrico
s280175

Miceli Davide
s280110

## Abstract

*Image classification is one of the most challenging fields in artificial intelligence. In particular, two of the most studied and difficult tasks are incremental learning and open world recognition: the former consists in incrementally adding new classes to a model and the problem, which is known in literature as catastrophic forgetting, occurs when the model forgets about its known classes while updating its parameters in respect to the new ones. The latter refers to the capability of a model to recognize if an image belongs to a class that hasn't learned yet, so that the image can be classified as unknown. The aim of this project is to explore these two problems, with the help of some of the most famous related papers, and to create a framework that combines the two tasks, which will be able t o learn new classes incrementally and at the same time to reject images from unknown classes. Finally, we propose our own modification in order to improve the results.*

## 1 Introduction

Our project consists, in the first part, in implementing some of the most famous frameworks about incremental learning. We define a sequence of tasks and each of them consists in training the model on a set of classes that is always new. Here the final model will be represented by iCaRL, which will be used as a starting point for the second part, where we have to implement the rejection capability into the model, so where the model will try to classify what it thinks he knows and reject what it thinks it doesn't know.

### 1.1 Open world setting

Here the rejection capability is implemented and we divide the classes into closed world set and open world set. We will incrementally train the model only on the classes belonging to the closed world. The closed world classes are also used for testing, to evaluate the accuracy of the model and to see if it rejects any of the samples, which should ideally never happen. Classes from the open world are used only for testing and the model should ideally reject them all.

### 1.2 Proposed modification

The baseline for the open world setting uses the last Fully-connected layer (FC) of the network to decide if a sample should be rejected or not. Our modification instead tries to exploit geometrical characteristics of the samples: using NME classifier, we base our decision to reject or not a sample using its distance to the centroids of the known classes. The details are shown in section 9.

### 1.3 Dataset and Network

The dataset used for all the experiments is CIFAR100: it consists of 32x32 images from 100 different classes. Each class has 600 images, 500 for the training set and 100 for the test set. Also, the classes can be grouped into 20 superclasses and they comprend a large variety of categories, from the natural world (living and non-living creatures) to artificial constructions and machines. The network used is a 32-layer ResNet, specifically adapted to the dataset and the incremental learning settings.

### 1.4 Code

We made a large use of the numpy library, as well as Pytorch for all the parts related to the model. Pandas and Seaborn are used for visualization. The experiments are divided into several folders, each of them corresponding to a different version of the code.

## 2 Related Works

Some papers related to incremental learning and open world recognition helped go through all the steps required. In [1] is introduced a method to distill the knowledge from a cumbersome model to a lighter one, and it's used for incremental learning in [2], where the temperature distillation is described. In [3] the authors describe the concept of exemplars and NME to further improve the results. Some modifications to [3] are implemented in [4] with the bias correction layer and in [5] where the classifier is a cosine linear layer and the less forget constraint is introduced for the loss. The work in [6] helped understanding the problem of open world recognition. It also makes use of advanced methods to boost the results in an open world setting.

## 3 Handling the dataset

The whole dataset is divided into training set and test set following the standard division that we found downloading CIFAR100. In the preprocessing step each training image follows the same transformation pipeline, which consists in random crop, random flip and normalization, while the only transformation done in the test set is the normalization. Then the dataset is divided in 10 splits with 10 random classes for each (a seed is used for replicability).

## 4 Fine Tuning

This section describes the behavior of the fine tuning baseline, together with some details that will be common for all the incremental learning models.

### 4.1 Model

The model trains a number of times equal to the number of tasks. At the end of each training, the accuracies in the training set and the test set are computed. For the accuracy of the training set we only use the current 10 classes, while for the test set we use all the classes seen until that point. The loss used for training is Binary Cross Entropy (BCE), which is a classification loss that requires the one-hot encoding of the labels. What causes catastrophic forgetting is the combination of two factors: the fact that there's only classification loss and the incremental learning settings. The loss only penalizes the difference from the predicted labels to the ground truth, so the model doesn't have any constraint in updating the parameters to adapt to the new classes, there's nothing that prevent the forgetting of features that were able to recognize old classes. The result is that during the test phase, the only labels predicted are the ones relative to the current task, and this is translated to a huge drop in accuracy through the tasks.

## 5 Learning without forgetting

This approach is described in [2] and is an attempt to preserve some knowledge from the previous model to the current one, this is done with the use of distillation loss.

### 5.1 Distillation Loss

At the beginning of a new task, a copy of the model is created, we will call it *old model*, as it refers to the model from the end of the previous task. We will use the fact that this model is the best in recognizing the old classes. During the training phase we forward the images to both the models, and although we train on a new set of images, the old model, which doesn't know about new classes, will try to extract from them features belonging to old classes and its prediction is compared to the output of the current model. The aim of the distillation loss is to reduce as much as possible the differences between these two outputs, that can be translated into forcing the current model to extract features that can be used both for old classes and for the new ones, applying the knowledge that comes from the old tasks and preventing catastrophic forgetting. So the total loss is the combination of classification loss and distillation loss and it's defined as follows:

$$\ell(\Theta) = -\sum_{(x_i,y_i)\in\mathcal{D}} \Big[\sum_{y=s}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y\neq y_i} \log(1-g_y(x_i))$$

$$+\sum_{y=1}^{s-1} q_i^y \log g_y(x_i) + (1-q_i^y)\log(1-g_y(x_i))\Big]$$

where $q$ is the output of the previous model, $g$ is the output of the new model, the known classes are numbered from 1 to $s-1$ and the new classes are numbered from $s$ to $t$.
To implement this we used *BCEWithLogitsLoss*, where the target matrix refers to the ground truth labels for the new classes and to the output of the old model for the previous classes. Now the model is able to correctly classify some images from old classes even in the current task. However, the performances can be further improved with the use of the techniques described in the next section.

## 6 iCaRL

This model is based on [3] and it's one of the main pillars of incremental learning. The main idea is based on the use of a small portion of the dataset called memory, which contains a maximum of K samples called *exemplars*. These exemplars are the most representative images of the old classes, and using them in training, together with the new training set, helps the model to better remember the old features. The management of the exemplars requires the implementation of the following methods (implemented in model.py), also the new classifier makes use of the exemplars to improve results.

### 6.1 Update Representation

This part of the code refers to the training phase, which is similar to the training in LwF. The only difference is that the training set is now the combination of $X$, which is the set of images from the new classes and the whole exemplar set $P$.

$$\mathcal{D} \leftarrow \bigcup_{y=s,..,t} \{(x,y)x \in X^y\} \cup \bigcup_{y=s,..,s-1} \{(x,y)x \in P^y\}$$

Then the model is trained using *BCEWithLogitsLoss* for both classification and distillation loss. Although in the evaluation phase the classifier is NME, during the training we still use the output of the FC layer of the ResNet.

### 6.2 Construct Exemplar Set

At the end of the training phase, the exemplars of the new classes must be computed. In order to do that, for each new class we use the feature extractor version of the ResNet, which is the whole network except for the FC layer, to extract the features $\varphi$ from the images, compute the mean $\mu$ of the current class, and select the most representative images for each class. An image has a certain distance from the mean of its class, and we use this distance to rank the images from the most representative to the less representative.

$$p_k \leftarrow \underset{x\in X}{\operatorname{argmin}} \left\|\mu - \frac{1}{k}[\varphi(x) + \sum_{j=1}^{k-1}\varphi(p_j)]\right\|$$

When the closest image is chosen, it is removed from the pool of images and the operation is repeated for the number of times required, which is correlated to the size of the memory. In particular, the maximum number $m$ of exemplars for each class depends on the size of the memory $K$.

$$m = K/n$$

where n is the number of known classes. The new set of exemplars is then added to the total exemplar set.

$$P \leftarrow (p_1, ..., p_m)$$

## 6.3 Reduce Exemplar Set

When the exemplars of a new set of classes are introduced we need to delete some exemplars from the old classes due to the max memory constraint. When we need to remove some exemplars we delete the less representative ones (that are the most distant from the mean of the respective class). The exemplars are ordered with increasing distance so the method simply deletes the latest items of the list that contains the exemplars.

## 6.4 Nearest Mean of Exemplars

In the classification phase, this classifier makes use of exemplars to achieve better results respect to the FC layer. At this point each class will have a set of exemplars, for which we can compute the mean and assign at each image the label related to the closest centroid. Note that because we use the feature extractor to calculate the mean of exemplars, their position into the feature space changes accordingly to the updated parameters and the number of classes, this prevents the exemplar means from being too close from each other when the number of classes increase.

$$\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$$

$$y^* \leftarrow \underset{y=1,\dots,t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\|$$

where $\mu_y$ is the mean of the features of exemplars of the class $y$ so $y^*$ is the class that minimizes the distance between the features of an image and the mean of the features of the exemplars of that class.

## 6.5 Results

In figure 1 we can see a comparison of the confusion matrices of FineTuning, LwF and iCaRL. The comparison of the accuracies can be seen in figure 2.

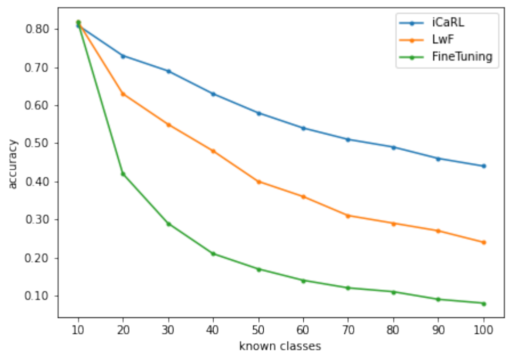


Figure 2: accuracies comparison between the 3 models

# 7 Ablation study

This section describes the functioning of the classifiers and losses tested on the iCaRL framework. The results are shown in Figure 3 and Figure 4. More detailed results are in Section 10.1.

## 7.1 Distillation Losses

We tried 4 different combinations of distillation and classification loss. The first combination uses Cross Entropy (CE) for classification loss and Kullback-Leibler divergence for distillation loss:

$$l(x, y) = L = \{l_1, \dots, l_N\}, l_n = y_n(\log y_n - x_n)$$

The Kullback-Leibler divergence treats the inputs as probability distributions and calculates the distance between them. Among all the combinations tried this was the most successful one and scored very similar to the standard configuration, that uses BCE for both classification and distillation. For the other combinations one uses CE for classification loss and MSE as distillation loss while another uses CE as classification loss and L1 as distillation loss. These two combinations scored slightly below the standard configuration. The last combination has BCE for classification and the distillation loss is softmax with temperature distillation, it is inspired by [1] and by [2], where it was used for the first time in incremental learning. It is a cross entropy applied to probabilities of the old model and the current one, and it's defined as follows:

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) = -\sum_{i=1}^{l} y'^{(i)}_o \log \hat{y}'^{(i)}_o$$

where $y'_o$ and $\hat{y}'_o$ are modified versions of the the probabilities of the two models $y_o$ and $\hat{y}_o$

$$y'^{(i)}_o = \frac{(y^{(i)}_o)^{1/T}}{\sum_j (y^{(j)}_o)^{1/T}}, \quad \hat{y}'^{(i)}_o = \frac{(\hat{y}^{(i)}_o)^{1/T}}{\sum_j (\hat{y}^{(j)}_o)^{1/T}}$$

The value suggested in these and other papers for the temperature $T$ is 2 , which increases the weight of smaller logits into the loss. This loss follows the same trend of standard configuration for the first 5 tasks, then goes below it.

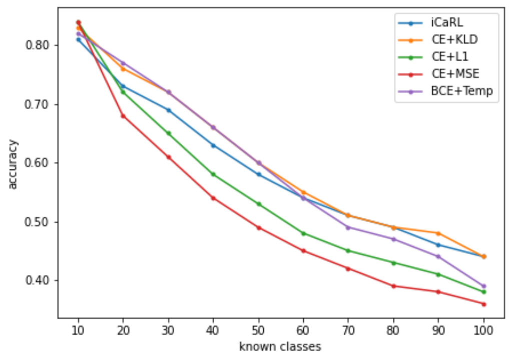
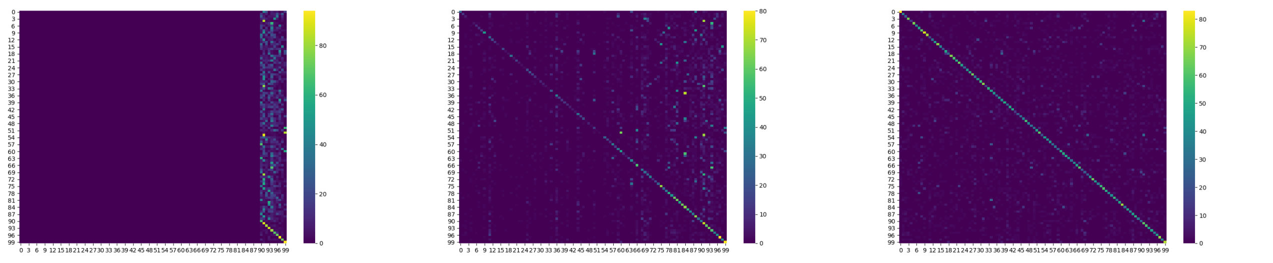


Figure 3: comparison between losses

## 7.2 Classifiers

Different classifiers have been used, from those already implemented to more complicated ones. The former use the features output from the ResNet to train shallow classifiers (SVM and KNN ). For the SVM, both the linear and the



(a) FineTuning (b) LwF (c) iCaRL

Figure 1: confusion matrix of the last task for the three models

rbf kernel have been tried. The first scored very poorly (below 40% accuracy already with the first 40 classes), instead the rbf kernel scored much better but is in any case below NME. KNN has been used with 5 neighbours and scored well in the first split of classes but then went down very quickly and fell behind NME ( below 20% of accuracy in the last 3 splits).

### 7.2.1 Bias Correction Layer

Bias Correction Layer has been implemented, as described in [4]. With this method, for each task, we also train a bias correction layer that tries to correct the bias towards new classes, to do so we use a small dataset with a portion of exemplars of the old classes and samples from the new classes, it is important to notice that this small dataset is balanced. The bias layer consists of only two parameters $\alpha$ and $\beta$ and gives an output as follows:

$$q_k = \begin{cases} o_k & 1 \leq k \leq n \\ \alpha o_k + \beta & n+1 \leq k \leq n+m \end{cases},$$

where $o$ is the input, $n$ is the number of known classes, $m$ is the number of new classes.

Then use simply the output of the Bias Correction Layer for prediction. As predicted from the authors of the paper, this method scored very similarly to iCaRL for small datasets such as CIFAR100 with small splits.

### 7.2.2 Cosine Linear Layer

Cosine Linear Layer, as described in [5], is a unified classifier, which means a classifier that treats both old and new classes uniformly and address the imbalance problem through three main components: cosine normalization, less-forget constraint, and inter-class separation. Since iCaRL predictions are biased towards the new classes, the RELU of the penultimate layer is removed to allow the features to take both positive and negative values and cosine normalization is adopted in the last layer , as:

$$p_i(x) = \frac{exp(\eta\langle\bar{\theta}_i, \bar{f}(x)\rangle)}{\sum_j exp(\eta\langle\bar{\theta}_j, \bar{f}(x)\rangle)}$$

where $f$ is the feature extractor, $\theta$ is the class embedding weight and $\eta$ is a control factor for the the peakiness of softmax distribution. Less-forget constraint aims to enforce a stronger constraint on the previous knowledge fixing the old class embeddings and compute a novel distillation loss on the features as below:

$$L^G_{dis}(x) = 1 - \langle\bar{f}^*(x), \bar{f}(x)\rangle$$

where $\bar{f}^*(x)$ and $\bar{f}(x)$ are respectively the normalized features extracted by the original model and those by the current one. $L^G_{dis}$ encourages the orientation of features extracted by current network to be similar to those by the original model. For what concerns inter-class separation, a margin ranking loss has been introduced as:

$$L_{mr}(x) = \textstyle\sum_{k=1}^{K} max(m - \langle\bar{\theta}(x), \bar{f}(x)\rangle + \langle\bar{\theta}^k, \bar{f}(x)\rangle, 0)$$

where $m$ is the margin threshold, $\bar{\theta}(x)$ is the ground-truth class embedding of x, $\bar{\theta}^k(x)$ is one of the top-K new class embeddings chosen as hard negatives for x. This loss is focused on the boundary and thus is less susceptible to the imbalance among classes. The losses are the combined:

$$L(x) = \frac{1}{|N|}\textstyle\sum_{x\in N}(L_{ce}(x) + \lambda L^G_{dis}(x)) + \frac{1}{|N_o|}\sum_{x\in N_o} L_{mr}(x)$$

where N is a training batch, $N_o \subset N$ are the reserved old samples contained in N. $\lambda = \lambda_{base}\sqrt{\frac{|C_o|}{|C_n|}}$ is a loss weight, where $C_o$ and $C_n$ are the number of old and new classes in each phase, $\lambda_{base}$ is a fixed constant. For the predictions both CNN and NME has been tried and, as noticed by the authors, the CNN performs better than the NME and also better than iCaRL but only for the first tasks.

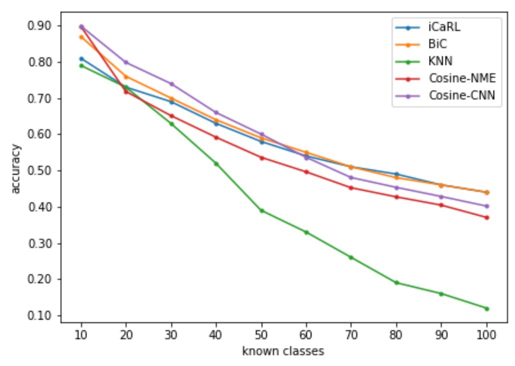


Figure 4: comparison between classifiers

## 8 Open World Recognition

It is now time to test our model in a more difficult situation. In the open world setting the classes are split in two groups made by 50 classes each. The first group is the closed world, is the only set of classes that the model will use to train its parameters, and is also used for evaluation. The open world instead is a set of classes never used for training, so is only used in the test phase. Another change in the model is the implementation of the rejection capability: it means that under certain conditions the model is able to reject a sample (i.e. to classify the sample as unknown). The ideal scenario can be described in two parts:

- The model doesn't reject any sample from the closed world test set, because they belong to known classes.
- The model rejects all the samples from the open world test set, because it has never trained on those classes.

As we will see, this scenario is quite unrealistic due to the difficulties that a model has in dividing known classes from unknown classes, probably because even in unknown classes it is possible to extract features similar to some known classes, and this is enough for the model to think it is facing something he already knows. The metric used is the harmonic mean between the accuracies of the two test sets. The accuracy on the closed world is computed as the number of correct predictions divided by the number of images. A prediction is considered wrong if it's different from the ground truth or if it refers to an image that the model decided to reject. Some images will be rejected for sure, so we expect that the results will be lower with respect to standard iCaRL. The accuracy on the open world is the number of samples rejected divided by the total number of samples. Again, we expect that the accuracy will not be 100% and that some samples will be considered as known.

### 8.1 Baseline

We used the same procedure of iCaRL to split the classes in groups of 10, then the first 5 groups are used for the closed world and the remaining 5 groups for the open world. We have 5 tasks and in each of them we add 10 classes from the closed world. The training is done only with the newest 10 classes, the testing on the closed world is done with all the classes known from the first task to the current, and the test on the open world is always done using all the 50 classes at the same time. The classifier used is the FC layer for both training and testing. The rejection policy, which is used for both closed and open world, works as follows: we take the logits that come from the FC layer, we apply the softmax transformation and we look at the highest probability, if this probability is below a threshold, which in this case is 0.5, the sample is considered unknown. This is a first approach to measure the uncertainty of the prediction made by the model but, as we will see in the results, the threshold is too low to reject the samples from the open world. In fact, even if they belong to unknown classes, the model tries anyway to assign a label from the known classes and in a lot of cases the highest probability is greater than 0.5, maybe because the relative class contains some features encountered in known classes.

## 9 Our Modification

For the modification of the project we decide to concentrate on the rejection strategy. Our starting point was the baseline, where with the same setting we only changed the value of the threshold to see how the model behaves. Then we moved to experiments more focused on the distances, changing the classifier and the metric of reference.

### 9.1 Increase Threshold

With the threshold set to 0.5 the model accepts almost all the samples from the closed world, but the number of samples rejected on the open set is too low. The idea behind the increase of the threshold is that for sure more samples from the open set will be rejected, while it's possible that the closed world will remain stable and don't lose so much accuracy, since we expect a large number of predictions with high probability from this set. The results reflected our expectations, increasing the threshold lead to an increase in accuracy on the open set greater than the loss on the closed set, consequently, the harmonic mean also increases. This trend is followed from 0.5 to 0.95, after which the value starts to decrease.

### 9.2 Distance Margin

Since the baseline relies on probabilities it's not possible to use the standard classifier for iCaRL, NME, but we know that NME has higher performances compared to a simple FC. Also, a fixed threshold means different performances based on the dataset and how the network is trained, so it's probably better to work with a metric correlated to the results that comes from data that we know and use that knowledge to reject new samples or not. The union of these two concepts led to the definition of a new metric. Here's the explanation: for each sample we store the *distance margin*, which is the difference between the distance from the closest centroid and the distance from the second closest centroid, then we compute the mean of these differences on the training set and use the result as a reference to reject the samples on the test sets. The reasoning is that the mean of the minimum distances from the closest centroid is a similar value between the different sets, but maybe on the closed world the samples are better defined, so there's more distance from other centroids in respect to the open world, where an image could be more "centered" between the centroids due to more uncertainty. So for both closed test set and open test set:
*A sample is rejected if it's distance margin is less than the mean of the distance margins of the training set.*

#### 9.2.1 Prioritize the open world

We found that the mean of distance margin is similar between the training set and the closed test test, so if we reject if a distance margin is less than the distance margin mean of the training set, a lot of samples from the closed world will be rejected, while in the open world, because the margins are overall lower, we reject the majority of them.

### 9.2.2 Prioritize the closed world

If we instead lighten the constraint the result is the opposite, the majority of closed world samples are accepted but we have a drop in accuracy for the open world. A possible choice for this, without defining a fixed threshold but obtaining it from data, could be the following:
*A sample is rejected if it's distance margin is less than the mean of the distance margins of the training set minus the standard deviation.*

To run this version the code requires the command line argument --*std*

### 9.2.3 Cosine similarity

Another set of experiments is about a change in the definition of distance. Standard NME uses the euclidean distance to compute distances of an image from the centroids. Here instead we use cosine similarity, where the higher the value, the more an image is similar to a given centroid, and it's defined as:

$$similarity = \frac{AB}{||A||||B||}$$

Also this script can be run prioritizing the open set or the closed set using the --*std* argument. This approach has been used also with the Cosine-NME, check Section 7.2.2 and achieves almost the same result as NME Cosine with mean and std.

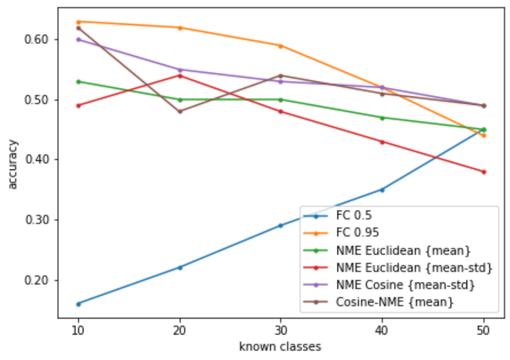


Figure 5: comparison between harmonic mean

## 10 Implementation details

For all the models, except for the Cosine Linear Layer approach, the following settings are in common. We use SGD for the optimizer, and all the hyperparameters are the same of the standard iCaRL implementation (batch size=128, K=2000, learning rate= 2, momentum= 0.9, weight decay= 1e-5, gamma= 0.2 and the learning rate is scaled by gamma on epochs 49 and 63), the only difference is the number of epochs, we use 80 epochs to be sure that the model converges. As said before, the loss used is the BCE with logit loss for both classification and distillation. In some cases a CE loss is used for classification, in these situations the learning rate is set to 0.1. For the Bias Correction Layer we use 250 training epochs for the Resnet and 200 training epochs for the Bias Layer, for the training of the bias layer is used a CE loss with learning rate equal to 0.001 that is constant through the epochs. All the codes were run three times, and each run differs from the others by how the classes are divided into tasks. Each different split corresponds to one of these seeds: 11, 42, 635. For Cosine Linear Layer the number of epochs is set to 160. The network is trained using a SGD with a batch size of 128 and the learning rate starts from 0.1 and is divided by 10 after 80 and 120 epochs. For other hyper-parameters, $\lambda_{base}$ is set to 5, $m$ is set to 0.5.

## Conclusion

We studied two difficult problems in image classification, their strengths, weaknesses and how the experts are trying to overcome them. We think that we will see other works related to these frameworks in the next years, as there is still some margin of improvement. As described in [6], the combination of incremental learning and open world recognition has high potential in real world applications, for example it is possible to recognize an unknown class, search for a dataset of that class and then add it incrementally to the model.

## 10.1 Results

### 10.1.1 Losses

| model | accuracies |
|---|---|
| base iCaRL | [0.81,0.73,0.69,0.63,0.58,0.54,0.51,0.49,0.46,0.44] |
| CE+KLD | [0.83,0.76,0.72,0.66,0.60,0.55,0.51,0.49,0.48,0.44] |
| CE+L1 | [0.84,0.72,0.65,0.58,0.53,0.48,0.45,0.43,0.41,0.38] |
| CE+MSE | [0.84,0.68,0.61,0.54,0.49,0.45,0.42,0.39,0.38,0.36] |
| BCE+Temp | [0.82,0.77,0.72,0.66,0.60,0.54,0.49,0.47,0.44,0.39] |

### 10.1.2 Classifier

| model | accuracies |
|---|---|
| base iCaRL | [0.81,0.73,0.69,0.63,0.58,0.54,0.51,0.49,0.46,0.44] |
| BiC | [0.87,0.76,0.70,0.64,0.59,0.55,0.51,0.48,0.46,0.44] |
| KNN 5 | [0.79,0.73,0.63,0.52,0.39,0.33,0.26,0.19,0.16,0.12] |
| SVM rbf | [0.81,0.73,0.67,0.59,0.51,0.46,0.39,0.34,0.27,0.21] |
| Cosine-NME | [0.90,0.72,0.65,0.59,0.54,0.50,0.45,0.43,0.40,0.37] |
| Cosine-CNN | [0.90,0.80,0.74,0.66,0.60,0.54,0.48,0.45,0.43,0.40] |

### 10.1.3 Harmonic Mean

| model | accuracies |
|---|---|
| FC, threshold = 0.50 | [0.16,0.22,0.29,0.35,0.45] |
| FC, threshold = 0.95 | [0.63,0.62,0.59,0.52,0.44] |
| NME Euclidean, threshold = mean | [0.53,0.50,0.50,0.47,0.45] |
| NME Euclidean, threshold = mean - std | [0.49,0.54,0.48,0.43,0.38] |
| NME Cosine threshold = mean - std | [0.60,0.55,0.53,0.52,0.49] |
| Cosine-NME Cosine, threshold = mean | [0.62,0.48,0.54,0.51,0.49] |

## References

[1] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Workshop*, 2014.

[2] Z. Li and D. Hoiem. Learning without forgetting. in european conference on computer vision. *ECCV*, 2016.

[3] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H Lampert. icarl: Incremental classifier and representation learning. *CVPR*, 2017.

[4] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. *arXiv:1905.13260v1*, 2019.

[5] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. *CVPR*, 2019.

[6] Dario Fontanel, Fabio Cermelli, Massimiliano Mancini, Samuel Rota Bulò, Elisa Ricci, and Barbara Caputo. Boosting deep open world recognition by clustering. *IEEE Robotics and Automation Letters*, 2020.