Machine Learning and Deep Learning Final Project Report

Tommaso Calò
S267682

Valerio Di Eugenio
S278972

Paola Privitera
S277760

1. Introduction

One of the major issues in object classification problems is that additional new data is often times either late or simply not available at the time a neural network has to be trained. Whenever new data is available, re-training the network becomes tricky. Indeed, by feeding it with additional images belonging to new classes (i.e. new with respect to those that are used for training the network), it causes the model to overwrite the old parameters with new ones that are no longer compatible to classify older data. This phenomenon is better known as catastrophic forgetting because the network actually "forgets" how to correctly classify former data. One of the greatest forward steps in the field of Artificial Intelligence is the development of incrementally learning systems that learn about new concepts over time from a stream of data. On the other hand, most artificial object recognition models known today, can only be trained in batches, where all object classes are known in advance and the training data of all classes can be accessed simultaneously. It becomes apparent that more flexible strategies are necessary to handle real-world object categorization situations. Indeed, an object classification system, such as an image classification algorithm, should be able to incrementally learn about new classes, when new training data is available. This circumstance is well known as class-incremental learning. In order to prevent, or at the least, mitigate the catastrophic forgetting phenomenon, it is possible to construct neural networks that entail a sort of memory of what has been learned so far. A suitable example is the iCaRL (incremental classifier and representation learning), which identifies a representative set of objects (exemplars) of older classes, which constitutes the network "memory" regarding past data. The main reason for iCaRL's better classification results are indeed its use of exemplar images. While it is intuitive that being able to rely on stored exemplars in addition to the network parameters could be beneficial, nevertheless this effect is much more pronounced in the class-incremental setting. For these reasons, three different scenarios have been proposed. The first one, called fine tuning baseline, which consists in training the network on a smaller set of classes and testing on a larger one (including previous, but unknown classes). The second, called learning without forgetting, which attempts to prevent the catastrophic forgetting by using the distillation loss. Finally, the last one consists in the implementation of an iCaRL whose main components are the classification by a nearest-mean-of-exemplars rule, the creation of exemplars, and the representation learning using knowledge distillation and prototype rehearsal (continuous stimulation of the network not only with new classes but also with the exemplars of older classes). For each of the proposed approaches the corresponding results and comparisons are summarized in a dedicated section. Furthermore, in order to compare models' results with each other and have an understanding of how good they are with respect to an ideal situation where the model is trained with all available data (including all the classes), a joint training strategy has been implemented. Indeed, joint training consists in training with all available classes, which means that the model is not subjected to catastrophic forgetting. For this reason, it represents the ideal training condition which theoretically should deliver the most satisfactorily results (it represents the upper bound for models' results). In other words, joint training comparison with the other models provides a clearer insight on the impacts of catastrophic forgetting. Finally, in order to study the sensitivity of the iCaRL method to structural changes, additional variations of classifiers and loss determination are proposed, plus a brand-new custom modification on both the structure of the network and the algorithm used for the exemplars' choice.

1.1. Dataset

The dataset used in the present work is the CIFAR-100, which consists in 60,000 images (size 32x32) divided equally in 100 mutually exclusive classes. There are 50,000 training images and 10,000 test images (500 training images and 100 testing images per class). Each image comes with a label which represents the class to which it belongs. The experiments run on CIFAR-100 have been carried out using 10 classes per learning step. In particular, the dataset has been divided in 10 sets of classes, which are then learned on different training steps. On the other hand, the test set includes all the classes seen in current and previous training steps. As the subdivision of classes in batches and the initial seed setting from which the model training starts might have significant impacts on the results, three initial seeds have been set. The imposition of an initial seed refers to a specific subdivision of classes into 10 batches. Thus, final model results have been computed as the mean of results coming from training the model under each different initial seed. From a programming standpoint, a new class – iCIFAR100 – has been created in order to load the dataset from TorchVision and allow for the abovementioned splits for training and test data. A set of ad-hoc functions has been put in place to access the dataset and feed the network with the images belonging to the selected group of classes. As the original images are not immediately suitable to be processed by the network, data augmentation during the training has been performed. More specifically, each image of the training set has been subjected to random crop (with the addition of a padding equal to 4), horizontal flip, and further color/brightness/contrast adjustments. Finally, both training and test images have been normalized using the mean and standard deviation of the CIFAR-100 dataset, namely the vectors (0.5071, 0.4867, 0.4408) and (0.2675, 0.2565, 0.2761).

1.2. ResNet Implentation

The model used is made by two components: a feature extractor and a classifier. Commonly, the feature extractor is a Convolutional Neural Network (in this project Resnet-18 has been selected). For the classifier, instead, the focus is on two choices: a fully connected layer or a non-parametric classifier. The ResNet structure deployed in the present study consists in 18 layers ordered as an initial convolution component followed by four blocks (4 layers each), and a final global average pooling layer. Each of the blocks follows the same pattern. They perform 3x3 convolution with a fixed feature map dimension [64, 128, 256, 512] respectively, bypassing the input every 2 convolutions. Furthermore, the width and height dimensions remain constant during the entire block. The final part of the network accounts for a pooling layer that averages over everything spatially, and then be input into the last 100 way classification.

ResNets solve one of the biggest challenges in deep learning, known as vanishing gradient. This phenomenon often occurs when the network depth is too wide, and the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This implies that the weights do not update their values and thus, no learning is achieved. On the other hand, with ResNets, the gradients can flow directly through the skip connections backwards from later layers to initial filters; indeed, ResNet incorporates identity shortcut connections which essentially skip the training of one or more layers - creating a residual block, allowing the network to "memorize" from previous layers. At this stage, an explanation regarding the network dynamics, structure and how the components interact with each other is due. The first layer is the convolutional layer which use a kernel size of 3, a feature map size of 64 with stride and padding set equal to 1. The total number of filters is equal to 64. As each filter provides a channel in the output volume, the overall depth is given by the number of channels, in this case is 64. Furthermore, each filter is applied to the same spatial location of the image but looks for a specific concept in the input volume, producing a separate activation map. Therefore, each filter extrapolates a different information from the image, leading to enriched results. The first convolutional step is followed by a batch normalization, which is an element-wise operation and therefore, it does not change the size of the volume. Finally, the last operation is a ReLU activation. The second part of the network is composed by four blocks which present the same structure (basic block). The sequence of operations inside each block consists in a first convolutional operation, followed by a batch normalization and a ReLU activation, and then a further convolutional step and a batch normalization. The main difference when passing from one block to the next is that the number of filters doubles, and as the stride is set to 2, the result is a decrease in the input volume size (down-sampling due to the convolution), not in terms of depth but in height and width. A new class Network is created with the aim to combine the feature extractor (Resnet) with a fully connected layer, to be used for the traning phase.

1.3. Hyperparameters

All the experiments have been executed by training the network (ResNet-18) using the Stochastic Gradient Descent as optimization technique to update the weights based on loss, with momentum equal to 0.9, weight decay of 0.00001 and minibatches of size 128. Each training step consists of 70 epochs. The learning rate starts at 2.0 and is divided by 5 after 49 and 63 epochs (7/10 and 9/10 of all epochs). The above mentioned configuration of hyperparameters has been used in all the experiments conducted.

1.4. Fine Tuning Baseline

The first experiment consisted in training the neural network with sequential batches of classes and testing on the total classes seen so far. In other words, at the first round, the network has been trained on the first batch of 10 classes and tested on the same classes; at the second round the network has been trained on the second batch of 10 classes and tested on the first 20 classes in the dataset, and so forth. At round 2, it is clear that the first 10 classes are not accounted during the training phase, but they are included in the test set. The catastrophic forgetting phenomenon occurs because training a neural network with new data causes it to overwrite (and thereby forget) what it has learned on previous data. Therefore, one might expect that this training method always delivers the worst results; indeed, it can be seen as a lower bound for all models' results.

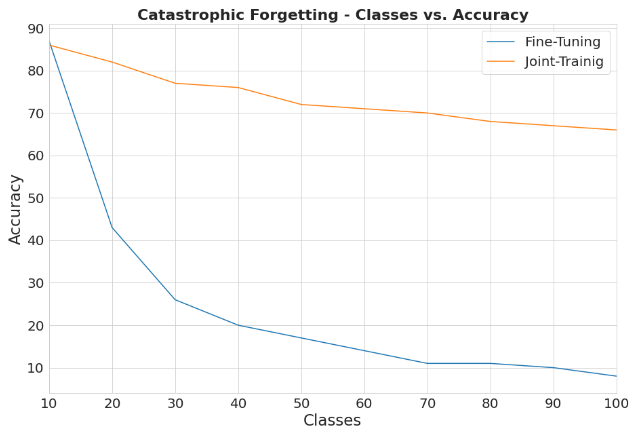

Figure 1. Average accuracy (line) across the classes.

Classes	10	20	30	40	50	60	70	80	90	100
Acc. FT	87	43	26	20	17	14	11	11	10	8
Acc. JT	86	82	77	76	72	71	70	68	67	66

2. Incremental Learning

2.1. Learning without forgetting

As already seen, as the number of tasks grows, storing and retraining past data becomes infeasible, and Finetuning has shown that the net doesn't manage to maintain information of past data without any techniques. In this way Learning Without Forgetting comes in handy: its aim is to optimize both for high new task accuracy and for preservation of responses on existing tasks from the original network. In particular, given the CNN with shared parameters and task-specific parameters, the purpose is to learn new parameters everytime new classes samples occur, which work well both on new and old tasks, by using only images of new ones (without using already seen data) and their predictions on the net trained for the old task. So, unlike joint training, this method doesn't need to store old datasets, but requires only to save the net at the precedent step than the current one. In details, when a new batch with ten new classes occur, the network is trained with it to minimize the loss for all the tasks: in general it is the result of the sum of two different loss functions, a classification loss encouraging the predictions of the new classes to be consistent with the ground truth, and a distillation loss which encourages the ouputs of the new network to approximate the outputs of the old one. For both the reference loss is the Binary Cross Entropy with logits loss: this loss measures how much two distributions diverges from each other (the more they do, the higher it is) and it combines a Sigmoid layer and the BCELoss in one single class. The sigmoid layer ouptus the probabilities of the image to belong to given classes, and it is applied just before computing the loss. it's important to notice that BCE loss averages the contribution of each class, so the more the training phase keeps on (i.e, the more classes the network sees), the more the distillation loss preveals on the classification one: in fact, the former is computed by considering even ten more classes at each step, while the latter always considers only the ten new classes occurring (so its contribution is divided always by 10). To try to overcome this unbalancing, a different way of loss calculation is adopted: instead of computing two losses with different inputs, it is done only one with these inputs:

1. A matrix representing the outputs of the current net, whose size is the number of images in the mini batch * number of all the classes seen so far (after the sigmoid layer each value will correspond to the label probability of the image of the row to belong to the category indicated by the column)..

2. A matrix of the same size, but composed differently: the first old task columns corresponds to the responses of each new task image by the precedent-step network, while the rest are the one-hot ground-truth label vector (a vector of zeros unless in the position corresponding to the ground truth, where it values one).

In this way, all the classes have the same weight in affecting the loss during the same step. In addition, the output of the last fully connected layer, which performs the classification, would have been set to have 100 ouput neurons right away from the beginning. It would have guaranteed a perfect balance in the classes distribution while while computing the loss not only in the same step, but even across the steps, by scaling it always by a factor of 100. Despite this, it has been decided to adopt an incremental approach, updating the net every time new classes occured by adding, in this case, ten more neurons at a time, because it seemed more straight-forward and coherent with the general problem investigating. Even though this choice leads the loss value to be scaled by a different factor at each iterations, it is still sufficient to justify the so high learning rate of 2.0, which would be too much aggressive if the loss wasn't divided by any factor: in fact, for instance, by using the Cross Entropy Loss, which computes the mean only according to the size of the batch, and maintaining the same learning rate, the model diverges after few epochs.

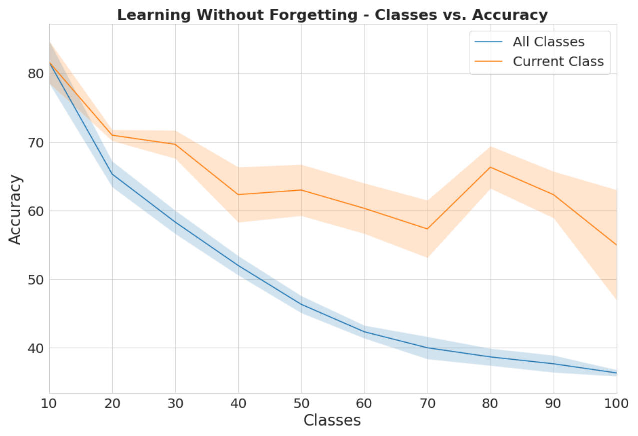

Figure 2. Average accuracy (line) and its variance (light band) across the classes.

Classes	10	20	30	40	50	60	70	80	90	100
Accuracy(%)	84	65	58	52	46	43	40	39	37	35

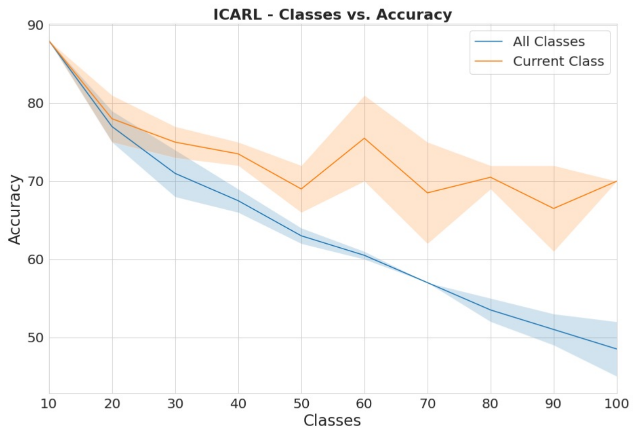

Figure 3. Average accuracy (line) and its variance (light band) across the classes.

Classes	10	20	30	40	50	60	70	80	90	100
Accuracy(%)	86	75	68	66	64	61	57	55	53	51

2.2. Icarl

It is observed that LwF tends to classify the test samples into new classes. In order to overcome this issue, ICaRL model adopts a new strategy, whose key is the exemplars. Exemplars are the most representative images for all the categories : every time a new class occurs, its average features vector is computed over all its training examples, and iteratively one of its examples themselves, the one that leads the average exemplars feature vector to best approximate the one computed over all the examples, is selected and added to the exemplar set, until the number of exemplars stored for the class fits the target number reserved for all the classes. This number, m, is obtained by dividing the maximum number of exemplars possible to be stored, K, which in this case is equal to 2000, by the classes seen so far, t. Being K small in size, the purpose is to use it full extent at each step, so whenever iCaRl encounters new classes it adjusts its exemplar sets according to the new m, which is recomputed according to the new number of classes: so, the number of exemplars available for each class has to be reduced at each step, and in order to do this, the last exemplars for each class are discarded until the new memory size is fit. In this way, the prioritized exemplar sets construction described above guarantees that the average feature vector over any subset of exemplars is a good approximation of the general mean vector of the classes, making it that robust against the removal procedure iteratively called. The training phase is handled by using both all the examples of the new classes occurring and the exemplars of the old classes already seen in a unique training set, and the network parameters are updated by minimizing a loss function with a similar procedure of learning without forgetting method: a unique BCE Loss with Logits is used, whose inputs are one the matrix output from the current network, the other the concatenation of the outputs of the current training set evaluated on the network of the previous step (with the parameters not updated for the new task), properly stored, with the one-hot encoded vector indicating the right samples label, in order to both lead the network,for the new images, to output the right class indicator and for the old to reproduce the scores of the previous step.

The classification step is performed by using Nearest-Mean-of-Exemplars method: to predict the class label of a new sample, iCaRL first computes a prototype vector for each class seen so far by averaging the feature vector of its all exemplars , then compares the extracted feature vector of the samples with all the prototypes and finally assigns the class label with most similar prototype.

2.3. Ablation Study

2.3.1 Losses

In this section are described the different kind of losses adopted and the subsequent results achieved. It's important to stress that for this purpose, a distinction between classification loss, performed to optimize classification task on new classes, and distillation loss, to make the model try to not forget previous knowledges already learnt, and the total one is computed by summing them. According to this, three different combination are tested:

CrossEntropy Loss with Mean Squared Error loss: The first one is a common choice when performing a classification problem with N classes, and in this case it is use for the new task classification: it means that it receives as input the unnormalized probability scores for each of the new classes occurring (a tensor of size Minibatch*number of new classes), and as target the correct labels associated for each minibatch sample. Internally, a Softmax activation is performed on the input to obtain a probability over the classes taken into account for each image, while the targets are one-hot encoded. The loss are computed in the following way:

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

Where Y is the binary indicator (0 or 1) if class label C is the correct classification for sample o and P indicates the probability of the observation o to belong to the class C. In this way it measures how a distribution differs from the truth vector: in fact for each sample the softmax outputs a vector whose values are in the (0,1) interval and, by minimizing this function, it encourages the input to be as similar as possible to the ground truth one-hot encoded which is the ideal distribution representation.

For the distillation knowledge MSE loss is used: it computes the mean of the squared distances between the targets and predicted values. Through this, the aim is to minimize the differences between the output scores of the minibatch training images given by the current net and the output of the same images performed by the old task net. In particular, being the purpose of the distillation to maintain the past knowledges, it makes sense for the output of the current model to consider only the scores corresponding to the old classes. Before computing the loss, both the output are passed through a sigmoid function to squash their values in a range (0,1). Taking into account that the old net is no more updated, the same minibatch obtain the same output across different epochs when evaluated by this, while this changes when using the current one, so the purpose of the distillation function is to make this change as much as possible towards the old scores. At the end, the two losses are summed and averaged over each loss element in the minibatch (i.e, for both the losses the reduction parameter is set to 'sum' and at the end manually divided by the minibatch size). By averaging only over the number of samples and not also over the output neurons (as in the first loss implementation), the learning rate is scaled by a magnitude and set to 0.2. Leaving the other hyperparameters and the classifier unchanged, the following results is obtained:

Classes	10	20	30	40	50	60	70	80	90	100
Accuracy(%)	84	76	71	69	62	58	53	50	46	44

Mse + Mse: Differently from the case above, the Mean Square error here is used to perform the classification task too: It receives as input the scores of the images given by the current model, as target the corresponding labels one-hot encoded and it computes the loss value as described above. The distillation phase works as before: the old model and new model ouputs are taken into account (even in this case only the scores corresponding to the old classes are considered) and a sigmoid function is performed on them before computing the loss. The two resulting losses are first summed and then divided by the batch size to average the results. Even in this case, the learning rate is scaled to 0.2. With this combination of losses, the following results are obtained:

Classes	10	20	30	40	50	60	70	80	90	100
Accuracy(%)	83	75	66	59	57	56	54	49	48	47

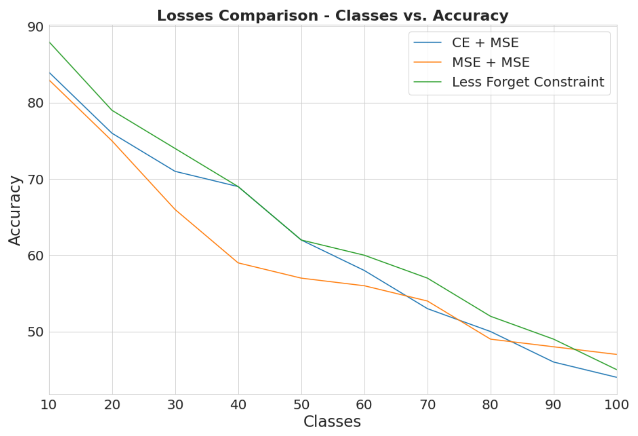

Figure 4. Average accuracy (line) and its variance (light band) across the classes.

Less Forget Constraint The losses used so far have been computed by using the output scores of the models. This third combination, instead, proposes a distillation phase providing a strong constraint on the previous knowledge based on the features of the images, by encouraging the orientation of those extracted by the current network to be similar to those by the original model. The intuition behind this approach is that the spatial configuration of the class embeddings reflects the inherent relationships among classes, so, to preserve the previous knowledge, it is reasonable trying to keep that configuration (for more details check the appendice). According to this, the distillation loss is computed in the following way:

$$L^{G}_{dis} = 1 - \langle \bar{f}^{*}(x), \bar{f}(x) \rangle \tag{2}$$

Where $\langle \bar{f}^{*}(x), \bar{f}(x) \rangle$ denotes the cosine similarity between the normalized features extracted from the old and the new models. In particular, the cosine similarity measures the cosine of the angle between the two inputs, so, the closer they are (the smaller the angle), the more the result tends to one, and, finally, the more the distillation loss tends to zero. The normalization performed on the features is the cosine normalization and it is adoped directly in the last layer of the ResNet feature extractor: in this way, the ReLu in the penultimate layer is removed to allow the features to take both positive and negative values. These changes affect also the classification loss with respect to the other ones used so far: in details, the work flow of the Cross Entropy Loss still remains the same, but the probability score for each class is computed in a different way according to the different ResNet used just described.

Furthermore, the distillation loss is multiplied by a factor lambda which increases according to

$$\lambda = \lambda_{base}\sqrt{|C_n/C_o|} \tag{3}$$

Where Cn and Co are respectively the number of the new and old classes. In this case, the new classes occuring are always ten and, consequently, the old classes size increases always of ten at each step: this leads the second part of the variable to scale always in the same way across different runs, so the weight to the distillation loss is mainly controlled by the base lambda, which there is set to 5. As easily noticeable, the higher it is, the more importance is given to the distillation phase.

Classes	10	20	30	40	50	60	70	80	90	100
Accuracy(%)	88	79	74	69	62	60	57	52	49	45

2.3.2 Classifiers

Different classification approaches are tested instead of Nearest Mean, and their performances are compared not only to each other but also to the performances obtained by the fully connected layer of the network (hybrid 1) to have a general idea of the trend across the classes. The feature extractor task is trained by maintaining the hyperparameters and optimization algorithm of standard iCaRL, in order to make the results directly comparable.

K-Nearest Neighbors The first classifier to be implemented is K-Nearest neighbors. To handle the class unbalance in the training phase (i.e. the training set is composed by 5000 images of new classes occurring and 2000 exemplars of old classes) a random under resampling technique is performed: it consists on randomly selecting examples from the majority classes and deleting them to create a new balanced dataset. This has been done to prevent the majority classes to preveal on the classification performances, leading the classifier to mainly assign the labels of the last categories. Before feeding the classifier, the data are standardized to have a mean around 0 and standard deviation around 1 to avoid anomalies during the learning processing, beacuse features with variance greater in magnitude than others may make the classifier not able to correctly learn other properties as expected. For the same reason, also the test set is standardized before getting evaluated.

The only hyperparameter K is tuned a bit in order to achieve the best performance possible.

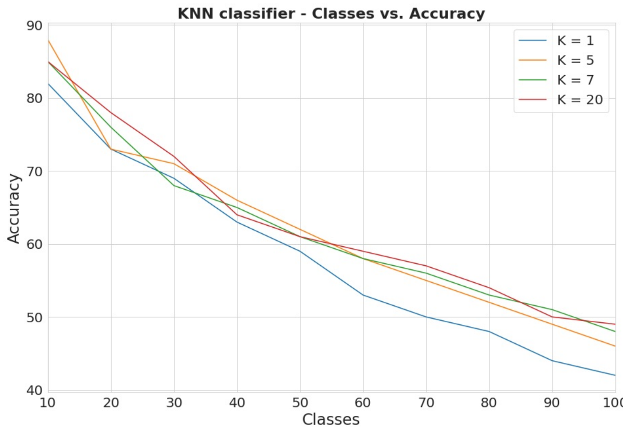

Figure 5. KNN Classifier: accuracy trend for different K values.

K=1										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	82	73	69	63	59	53	50	48	44	42

K=5										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	88	73	71	66	62	58	55	52	49	46

K=7										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	85	76	68	65	61	58	56	53	51	48

K=20										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	85	78	72	64	61	59	57	54	50	49

Supposing the dataset composition is non-homogeneous, it might be difficult to sub-divide each cluster correctly. Indeed, if the dataset is characterized by a high number of outliers, the clusters cannot be linearly differentiated from one another. The KNN algorithm, by exploiting the composition of the neighborhood of the data point to be classified, can overcome such problem and deliver consistent results. Obviously, the size of the neighborhood has a significant impact on the goodness of the classification. If the neighborhood is too small, the probability of miss-classification is higher due to the presence of outliers; on the other hand, if the neighborhood is too large, it may include unwanted datapoints belonging to other classes. The results show that the final accuracy increases with larger values of K. In particular the highest accuracy of 49 has been reached with K equal to 20. As per prior considerations, it should be pointed out that K = 20 cannot necessarily be considered a large neighborhood. Therefore, it cannot be ruled out the hypothesis that with even larger values for K higher accuracies may be obtained, till the critical valued for K that causes the accuracy to decrease (K is too large).

SVM classifier The second classifier tested is SVM classifier with RBF kernel. Even in this case the training phase is handled only by considering the under sampled dataset, for the reason explained above, and the same features preprocessing is maintained. For the purpose, the hyperparameter C is tuned for the classifier, leaving the 'gamma' set to default value calculated authomatically according to the data. The performances are quite good but not as good as those obtained with NMS. (Even with dimensionality reduction) The number of dimension of the space where the features are projected is still too high, and the support vectors to choose as possible solution to are really much: in this way, one solution among these is selected with the high possibility that it separates well some classes but not others.

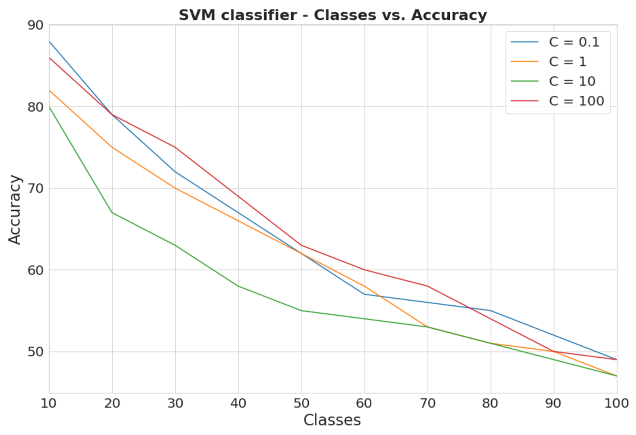

Figure 6. SVM Classifier: accuracy trend for different C values.

Cosine Similarity The last classifier tested is the cosine similarity, which in general computes the cosine of the angles between two vectors. In this way, as mentioned in the previous section concerning the losses, the more two vector are close the more the resulting value is high (with an upper bound of 1, being a cosine function). This classification is performed in two ways: the first one involves the modified Cosine Resnet Network and exploits the output of its last layer as classes prediction, after being trained for the purpose (i.e. the Cosine Resnet Network is already described in the losses section), while the second consists on calculating the similarity between the features vector of the sample

C=0.1										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	88	79	72	67	62	57	56	55	42	49

C=1										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	82	75	70	66	62	58	53	51	50	47

C=10										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	80	67	63	58	55	54	53	51	49	47

C=100										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	86	79	75	69	63	60	58	54	50	49

to be classified and the features mean vector of the examplars, assigning to the sample the label of the exemplar vector which results to have the maximum cosine value with (in this case the usual net is trained with the usual iCaRL optimization alghoritm)

As expected, the second option performs in a better way, reaching pretty similar results to the Nearest mean: indeed, it operates in a pretty similar way by taking into account the mean of the features of the exemplars for each class, only computing the similarity in a different way (one considers the vectors distance element by element, the other evaluates the angle in between).

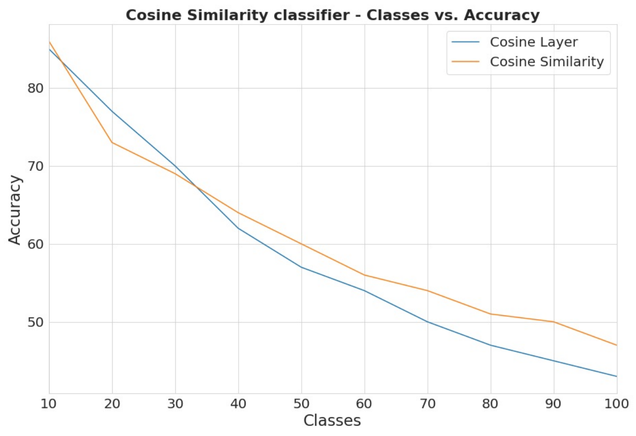

Figure 7. Accuracy Trend for cosine layer and cosine similarity between features.

Cosine Layer										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	85	77	70	62	57	54	50	47	45	43

Cosine Similarity										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	86	73	69	64	60	56	54	51	50	47

3. Improvements

The first improvement that has been consider is on the performance of the network, since one of the major lack of the model is to have few images available per class (500). A review of current papers on improving the performances of networks with few samples led us to consider spatial and channel attention modules implementation. The second improvement is focused on the algorithm. A deep research has been made over recent papers of CVPR 2020 and others [1] [2] [3] [4] in order to find out clues for improvement of the algorithm, unfortunately all of those would imply a very different approach to the problem. Anyway some clues on the importance of choosing exemplars led us to the consideration that instead of maximizing only the distance from the mean, for the exemplars to be a good representation of the original population also the variance should be taken into account. Resuming, our improvement apply on both of the following fronts:

- **Network improvements:** Implementation of Channel and Spatial attention modules.
- **Algorithm improvements:** Optimization of Exemplar choice minimizing also the distance of the exemplar set standard deviation with respect to the variance of the class.

3.1. Network Improvement

The most important problem affecting the network performance is the scarcity of images per class (500), the low-data problem makes the feature of each test class not representative for the true class distribution, as it is obtained from few labeled support samples. The attention idea is inspired by the human few-shot learning behavior. To recognize a sample from unseen class given a few labeled samples, human tends to firstly locate the most relevant regions in the pair of labeled and unlabeled samples. Similarly, given a class feature map, spatial and channel attention modules generates an attention map for each feature to highlight the target object. Since convolution operations extract informative features by blending cross-channel and spatial information together, we adopt the module to emphasize meaningful features along those two principal dimensions: channel and spatial axes. To achieve this, we sequentially apply channel and spatial attention modules, so that each of the branches can learn 'what' and 'where' to attend in the channel and spatial axes respectively. As a result, the module should help the information flow within the network by learning which information to emphasize or suppress.

3.1.1 CBAM: Convolutional Block Attention Module

Given an intermediate feature map $\mathbf{F} \in \mathbb{R}^{C\times H\times W}$ as input, CBAM sequentially infers a 1D channel attention map $\mathbf{M_c} \in \mathbb{R}^{C\times 1\times 1}$ and a 2D spatial attention map $\mathbf{M_s} \in \mathbb{R}^{1\times H\times W}$ as illustrated in Fig. 1. The overall attention process can be summarized as:

$$\begin{aligned} \mathbf{F}' &= \mathbf{M_c} \otimes \mathbf{F} \\ \mathbf{F}'' &= \mathbf{M_s} \otimes \mathbf{F}' \end{aligned} \tag{4}$$

where $\otimes$ denotes element-wise multiplication. During multiplication, the attention values are broadcasted (copied) accordingly: channel attention values are broadcasted along the spatial dimension, and vice versa. $\mathbf{F}''$ is the final refined output.

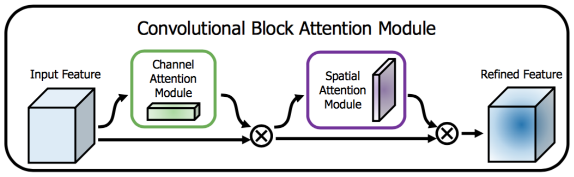

Figure 8. **The overview of CBAM**. The module has two sequential sub-modules: channel and spatial. The intermediate feature map is adaptively refined through our module (CBAM) at every convolutional block of deep networks.

Channel Attention Module It produces a channel attention map by exploiting the inter-channel relationship of features. As each channel of a feature map is considered as a feature detector, channel attention focuses on 'what' is meaningful given an input image. It first aggregate spatial information of a feature map by using both average-pooling and max-pooling operations, generating two different spatial context descriptors: $\mathbf{F^c_{avg}}$ and $\mathbf{F^c_{max}}$, which denote average-pooled features and max-pooled features respectively. Both descriptors are then forwarded to a shared network to produce our channel attention map $\mathbf{M_c} \in \mathbb{R}^{C\times 1\times 1}$. The shared network is composed of multi-layer perceptron (MLP) with one hidden layer. To reduce parameter overhead, the hidden activation size is set to $\mathbf{R_c}^{r\times 1\times 1}$, where r is the reduction ratio. After the shared network is applied to each descriptor, we merge the output feature vectors using element-wise summation. In short, the channel attention is computed as:

$$\begin{aligned} \mathbf{M_c}(\mathbf{F}) &= \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F}))) \\ &= \sigma(\mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{avg}})) + \mathbf{W_0}(\mathbf{W_1}(\mathbf{F^c_{max}})) \end{aligned} \tag{5}$$

where σ denotes the sigmoid function, $\mathbf{W_0} \in \mathbb{R}^{C/r\times C}$ and $\mathbf{W_1} \in \mathbb{R}^{C/r\times C}$. Note that the MLP weights, $\mathbf{W_0}$ and $\mathbf{W_1}$, are shared for both inputs and the ReLU activation function is followed by $\mathbf{W_0}$.

Spatial Attention Module It generate a spatial attention map by utilizing the inter-spatial relationship of features. Different from the channel attention, the spatial attention focuses on 'where' is an informative part, which is complementary to the channel attention. To compute the spatial attention, we first apply average-pooling and max-pooling operations along the channel axis and concatenate them to generate an efficient feature descriptor. Applying pooling operations along the channel axis is shown to be effective in highlighting informative regions. On the concatenated feature descriptor, we apply a convolution layer to generate a spatial attention map $\mathbf{M_s}(\mathbf{F}) \in \mathbb{R}^{H\times W}$ which encodes where to emphasize or suppress. In detail it aggregates channel information of a feature map by using two pooling operations, generating two 2D maps: $\mathbf{F^s_{avg}} \in \mathbb{R}^{1\times H\times W}$ and $\mathbf{F^s_{max}} \in \mathbb{R}^{1\times H\times W}$. Each denotes average-pooled features and max-pooled features across the channel. Those are then concatenated and convolved by a standard convolution layer, producing our 2D spatial attention map. In short, the spatial attention is computed as:

$$\begin{aligned}\mathbf{M_s}(\mathbf{F}) &= \sigma(f^{7\times7}[AvgPool(\mathbf{F}); MaxPool(\mathbf{F}))\\ &= \sigma(f^{7\times7}[\mathbf{F^s_{avg}}; \mathbf{F^s_{max}}]))\end{aligned} \tag{6}$$

where σ denotes the sigmoid function and $f^{7\times7}$ represents a convolution opera- tion with the filter size of 7×7.

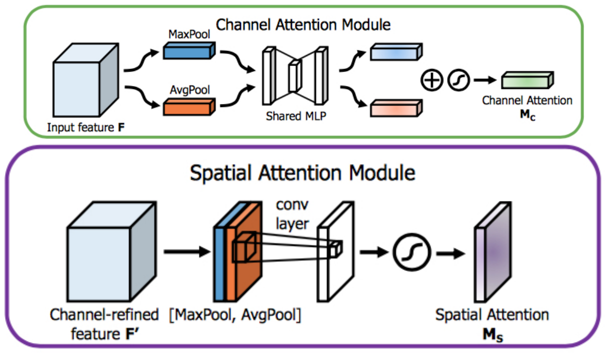

Figure 9. **Diagram of each attention sub-module**. As illustrated, the channel sub-module utilizes both max-pooling outputs and average-pooling outputs with a shared network; the spatial sub-module utilizes similar two outputs that are pooled along the channel axis and forward them to a convolution layer.

Implementation Channel and spatial attention modules are implemented as classes using PyTorch library straight in the ResNet class in the following way:

Where ϕ is the sigmoid function. Reduction factor for the hidden layer has been set to the value of 16.

```
1: **Channel Attention Module**
2: **initialization:**
3: $avgpool \leftarrow AdaptiveAvgPool2d$
4: $maxpool \leftarrow AdaptiveMaxPool2d$
5: $\Gamma \leftarrow Sequential(Flatten, Linear, ReLU, Linear)$
6: **forward(x):**
7: $avg_{out} \leftarrow \Gamma(avgpool(x))$
8: $max_{out} \leftarrow \Gamma(maxpool(x))$
9: **return** $\phi(avg_{out} + max_{out})$
```

```
1: **Spatial Attention Module**
2: **initialization:**
3: $conv \leftarrow Conv2d$
4: **forward(x):**
5: $avg_{out} \leftarrow Mean(x)$
6: $max_{out} \leftarrow Max(x)$
7: $x \leftarrow CAT(max_{out}, avg_{out})$
8: $x \leftarrow conv(x)$
9: **return** $\phi(x)$
```

Spatial and Channel Modules results are the multiplied in the Resnet Class with the result of the convolutional layers as explained above. The two modules can be placed in a parallel or sequential manner. Sequential arrangement has been shown to give a better result than a parallel arrangement. With channel placed above of spatial module.

3.2. Algorithm Improvements

As we already pointed out, one of the most critical points of the iCarl model, if not the most critical, is the way exemplars are chosen. In the original paper, exemplar are chosen minimizing the distance of the means of the exemplars set with respect the original population. This with the aim of representing the original population as good as possible in the exemplars set. Going further into this way of reasoning, has been decided to to minimize non only the distance with respect to the mean but also the distance with respect to the variance.

Mathematical Background Since we did not find a way to implement one expression which results in an image which minimizes both distance with respect to the mean and with respect to the variance of the class, it has been decided to alternate the appending of an image which minimizes the first and one image that minimizes the latter. For our scope, since the features data must be summed up till a certain number, the classical variance formula is not suitable. We use indeed the derived following formulation:

$$\begin{aligned}\sigma^{\mathbf{2}} &= \frac{\sum_1^N (X-\mu)^2}{N} = \frac{\sum_1^N (X^2 - 2\mu X + \mu^2)}{N} \\ &= \frac{\sum_1^N X^2}{N} - \frac{2\mu \sum_1^N X}{N} + \frac{N\mu^2}{N} \\ &= \frac{\sum_1^N X^2}{N} - 2\mu^2 + \mu^2 = \frac{\sum_1^N X^2}{N} - \mu^2\end{aligned} \tag{7}$$

Consider to be in the n_{th} step, with $n \in (0, m)$. The k_{th} image will be chosen as exemplar as follow:

$$\text{k} = \arg\min_{k \in K} \left| \sigma^2 - \left(\frac{\sum_1^n X_i^2}{N} + \frac{X_k^2}{N} - \mu^2 \right) \right| \tag{8}$$

Where $X_i,\ i \in I$ is the tensor of i_{th} images already present in the exemplars. Notice that since we want no exemplars to be repeated $K \cap I = \emptyset$.

Implementation The modification are contained in the same method used before to construct exemplars set.

```
    Improved Construct Exemplars Set
    $indexes \leftarrow [\,]$
 3: $exemplars \leftarrow [\,]$
    $cm \leftarrow classMean$
    $cv \leftarrow classVariance$
 6: SF $\leftarrow summedFeatures^2$
    IMGF $\leftarrow imagesFeatures^2$
    X $\leftarrow cv - \left( \frac{\textbf{SF}+\textbf{IMGF}}{i+1} - cm^2 \right)$
 9: $\vec{\mathbf{x}} \leftarrow \|\mathbf{X}\|^2$
    $index \leftarrow \arg\min \vec{\mathbf{x}}$
    while $index \in indexes$ do
12:     $x_{index} \leftarrow M$
        $index \leftarrow \arg\min \vec{\mathbf{x}}$
    end while
15: $append(exemplars, img_{index})$
    $append(indexes, index)$
```

The procedure is invoked each time $i \mod 2 = 1$

3.3. Results

Unfortunately Channel and Spatial attention do not improve the model performance, this could be due to the fact that our net is not deep as the one for which in the article are obtained the improvements.
The minimization of the distance with respect of the population variance in choosing exemplars doesn't improve the model.

CBAM										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	86	79	73	65	63	58	56	55	50	49

EV										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	82	78	73	68	65	61	55	53	50	47

CBAM+EV										
Classes	10	20	30	40	50	60	70	80	90	100
Accuracy	84	75	71	67	61	55	53	51	49	47

4. Final Results and Conclusions

Model	Avg. Accuracy
Finetuning	24.7
LwF	49.9
iCaRL	63.9
Less Forget	57.3
MSE + MSE	61.3
CE + MSE	61.3
KNN (K=20)	62.9
SVM (C=100)	64.3
Cosine Classifier	59.0
Cosine Similarity	61.1

In this section we will analyze and explain through visualizations the final results obtained through all the methods implemented. From fig.11,12,13 the following observation could be derived:

- **Fine-Tuning** Makes Correct predictions only on last batch of classes, this shows clearly the effects of catastrophic forgetting.
- **Learning without forgetting** Tends to make more prediction to the last classes, this can be seen by the increasing brightness for increasing classes.
- **ICarl** confusion matrix shows no overall pattern, the method rightly balances the prediction between old and new classes although not showing the best accuracy.

As it can be seen in figure 10 iCaRL clearly outperforms the other methods, and the more so the more incremental the setting, implemented improvements don't affect significantly the performance of the model although the logic behind them. Finetuning always achieves the worst results, confirming that catastrophic forgetting is indeed a major problem for in class-incremental learning, the efficacy of iCarl strictly relies on the bag of exemplars, this could be seen not confirming with the learning without forgetting paradigm but in practice is a good trade-off which although leaves the necessity of finding new well performing model more compliant with the mentioned paradigm.

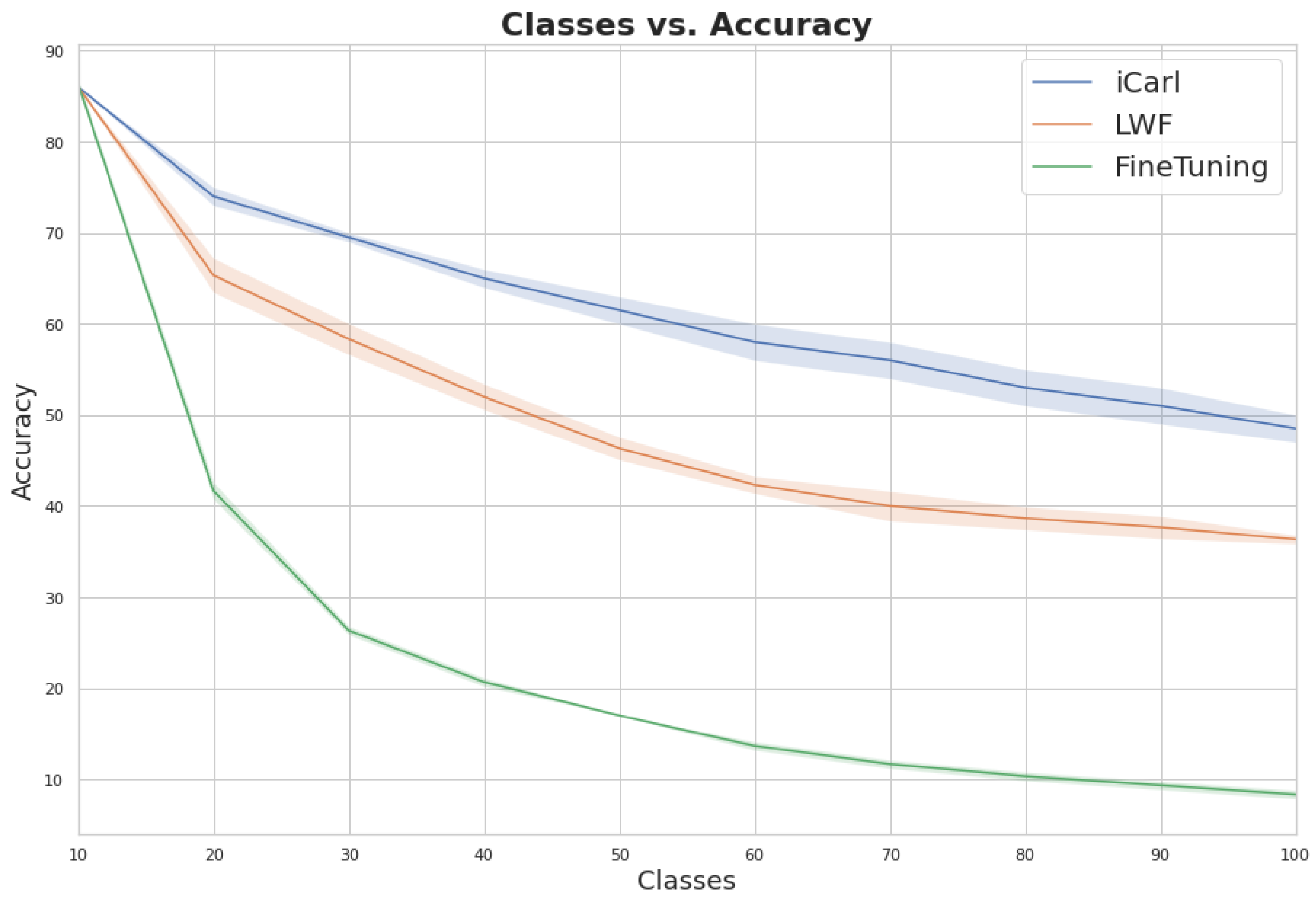

Figure 10. Comparison of iCarl, Learning Without Forgetting and FineTuning models

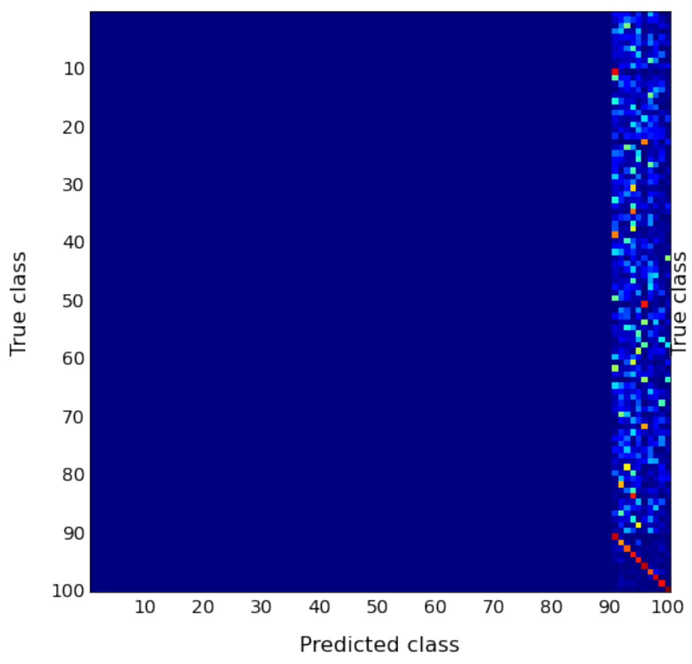

Figure 11. Fine-Tuning

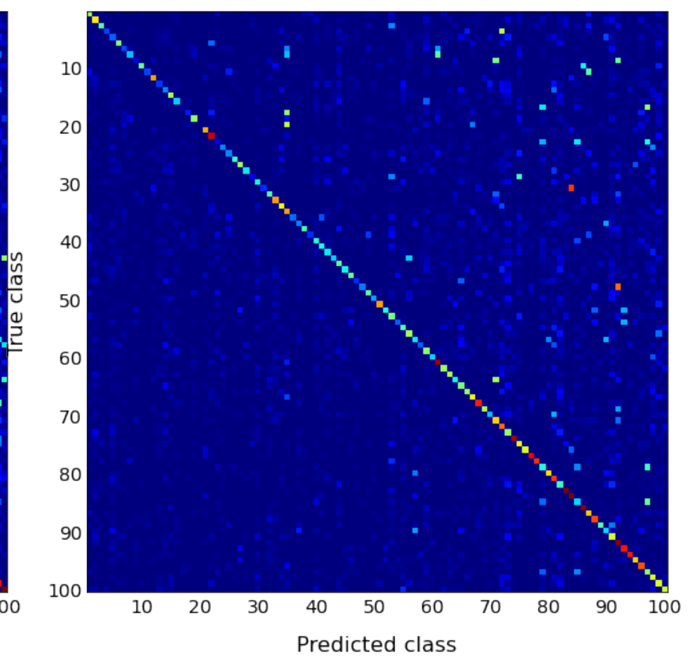

Figure 12. Learning Without Forgetting

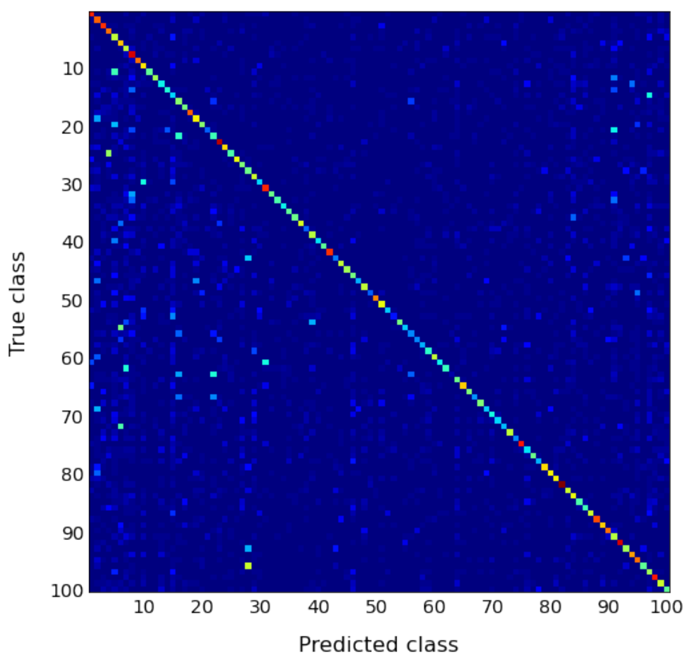

Figure 13. Icarl

Github Link: https://github.com/tommasocalo/Project_MLDL

References

[1] T. L. K. Y. B. F. Fei Mi, Lingjing Kong. Generalized class incremental learning.

[2] B. M. S. S. X. C. Ruibing Hou, Hong Chang. Cross attention network for few-shot classification.

[3] J.-Y. L. Sanghyun Woo, Jongchan Park and I. S. Kweon. Cbam: Convolutional block attention module.

[4] I. T. Wojciech Masarczyk. Reducing catastrophic forgetting with learning on synthetic data.