View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Institucional Universidad de Málaga

End-to-End Incremental Learning

Francisco M. Castro[1], Manuel J. Marín-Jiménez[2], Nicolás Guil[1], Cordelia Schmid[3], and Karteek Alahari[3]

[1] Department of Computer Architecture, University of Málaga, Spain
[2] Department of Computing and Numerical Analysis, University of Córdoba, Spain
[3] INRIA - Grenoble, France

Abstract. Although deep learning approaches have stood out in recent years due to their state-of-the-art results, they continue to suffer from *catastrophic forgetting*, a dramatic decrease in overall performance when training with new classes added incrementally. This is due to current neural network architectures requiring the entire dataset, consisting of all the samples from the old as well as the new classes, to update the model—a requirement that becomes easily unsustainable as the number of classes grows. We address this issue with our approach to learn deep neural networks incrementally, using new data and only a small exemplar set corresponding to samples from the old classes. This is based on a loss composed of a distillation measure to retain the knowledge acquired from the old classes, and a cross-entropy loss to learn the new classes. Our incremental training is achieved while keeping the entire framework end-to-end, i.e., learning the data representation and the classifier jointly, unlike recent methods with no such guarantees [1]. We evaluate our method extensively on the CIFAR-100 and ImageNet (ILSVRC 2012) image classification datasets, and show state-of-the-art performance.

Keywords: Incremental learning; CNN; Distillation loss; Image classification

1 Introduction

One of the main challenges in developing a visual recognition system targeted at real-world applications is learning classifiers incrementally, where new classes need to be learned continually. For example, a face recognition system needs to handle new faces to identify new people. This task needs to be accomplished without having to re-learn faces learned previously. While this is trivial to accomplish for most people (we learn to recognize faces of new people we meet every day), it is not the case for a machine learning system. Traditional models require all the data (corresponding to the old and the new classes) to be available at training time, and are not equipped to consider only the new data with a small selection of the old data. In an ideal system, the new classes should be integrated into the existing model, sharing the previously learned parameters. Although some attempts have been made to address this, most of the previous models still suffer from a dramatic decrease in performance on the old classes when new information is added, in particular, in the case of deep learning approaches [1–9]. We address this challenging task in this paper using the problem of image classification to illustrate our results.

A truly incremental deep learning approach for classification is characterized by its: *(i)* ability to being trained from a flow of data, with classes appearing in any order, and at any time; *(ii)* good performance on classifying old and new classes; *(iii)* reasonable number of parameters and memory requirements for the model; and *(iv)* end-to-end learning mechanism to update the classifier and the feature representation jointly. Therefore, an ideal approach would be able to train on an infinitely-large number of classes in an incremental way, without losing accuracy, and having exactly the same number of parameters, as if it were trained from scratch.

None of the existing approaches for incremental learning [1, 8–18] satisfy all these constraints. They often decouple the classifier and representation learning tasks [1], or are limited to very specific situations, e.g., learning from new datasets but not new classes related to the old ones [8, 12–14], or particular problems, e.g., object detection [9]. Some of them [10, 11] are tied to traditional classifiers such as SVMs and are unsuitable for deep learning architectures. Others [15–18] lead to a rapid increase in the number of parameters or layers, resulting in a large memory footprint as the number of classes increases. In summary, there are no state-of-the-art methods that satisfy all the characteristics of a truly incremental learner.

The main contribution of this paper is addressing this challenge with our end-to-end approach designed specifically for incremental learning. The model can be realized with any deep learning architecture, together with our representative memory component, which is akin to an exemplar set for maintaining a small set of samples corresponding to the old classes (see Section 3.1). It is learned by minimizing the cross-distilled loss, a combination of two loss functions: cross-entropy to learn the new classes and distillation to retain the previous knowledge corresponding to the old classes (see Section 3.2). As shown in Section 4, any deep learning architecture can be adapted to our incremental learning framework, with the only requirement being the replacement of its original loss function with our new incremental loss. Finally, we illustrate the effectiveness of our image classification approach in obtaining state-of-the-art results for incremental learning on CIFAR-100 [19] and ImageNet [20] datasets (see Sections 5, 6 and 7).

2 Related Work

We now describe methods relevant to our approach by organizing them into traditional ones using a fixed feature set, and others that learn the features, i.e., through deep learning frameworks, in addition to training classifiers.

Traditional approaches. Many initial methods for incremental learning targeted the SVM classifier [21], exploiting its core components: support vectors and Karush-Kuhn-Tucker conditions. Some of the methods [10] retain the support vectors, which encode the classifier learned on old data, to learn the new decision boundary together with new data. Cauwenberghs and Poggio [11] proposed an alternative to this by retaining the Karush-Kuhn-Tucker conditions on all the previously seen data (which corresponds to the old classes), while updating the solution according to the new data. While these early attempts showed some success, they are limited to a specific classifier and also do not extend to the current paradigm of learning representations and classifiers jointly.

Another relevant approach is learning concepts over time, in the form of lifelong [22] or never-ending [23–25] learning. Lifelong learning is akin to transferring knowledge acquired on old tasks to the new ones. Never-ending learning, on the other hand, focuses on continuously acquiring data to improve existing classifiers or to learn new ones. Methods in both these paradigms either require the entire training dataset, e.g., [24], or rely on a fixed representation, e.g., [25]. Methods such as [26–28] partially address these issues by learning classifiers without the complete training set, but are still limited due to a fixed or engineered data representation. This is achieved by: *(i)* restricting the classifier or regression models (e.g., those that are linearly decomposable [28]), or *(ii)* using a nearest mean classifier (NMC) [26], or a random forest variant [27]. Incremental learning is then performed by updating the bases or the per-class prototype, i.e., the average feature vector of the observed data, respectively.

Overall, the main drawback of all these methods is the lack of a task-specific data representation, which results in lower performance. Our proposed method addresses this issue with joint learning of features and classifiers.

Deep learning approaches. This class of methods provides a natural way to learn task-specific features and classifiers jointly [29–31]. However, learning models incrementally in this paradigm results in *catastrophic forgetting*, a phenomenon where the performance on the original (old) set of classes degrades dramatically [1–9]. Initial attempts to overcome this issue were aimed at connectionist networks [2, 4, 5], and are thus inapplicable in the context of today's deep architectures for computer vision problems.

A more recent attempt to preserve the performance on the old tasks was presented in [8] using distillation loss in combination with the standard cross-entropy loss. Distillation loss, which was originally proposed to transfer knowledge between different neural networks [32], was adapted to maintain the responses of the network on the old tasks whilst updating it with new training samples [8]. Although this approach reduced forgetting to some extent, in particular, in simplistic scenarios where the old and the new samples come from different datasets with little confusion between them, its performance is far from ideal. This is likely due to a weak knowledge representation of the old classes, and not augmenting it with an exemplar set, as done in our method. Works such as [1, 14] demonstrated this weakness of [8] showing significant errors in a sequential learning scenario, where samples from new classes are continuously added, and in particular when the new and the old samples are from related distributions—the challenging problem we consider in this paper.

Other approaches using distillation loss, such as Jung *et al.* [12] propose to freeze some of the layers corresponding to the original model, thereby limiting its adaptability to new data. Triki *et al.* [14] build on the method in [8] using an autoencoder to retain the knowledge from old tasks, instead of the distillation loss. This method was also evaluated in a restrictive scenario, where the old and the new networks are trained on different datasets, similar to [8]. Distillation loss was also adopted for learning object detectors incrementally [9]. Despite its success for object detection, the utility of this specific architecture for more general incremental learning scenarios we target here is unclear.

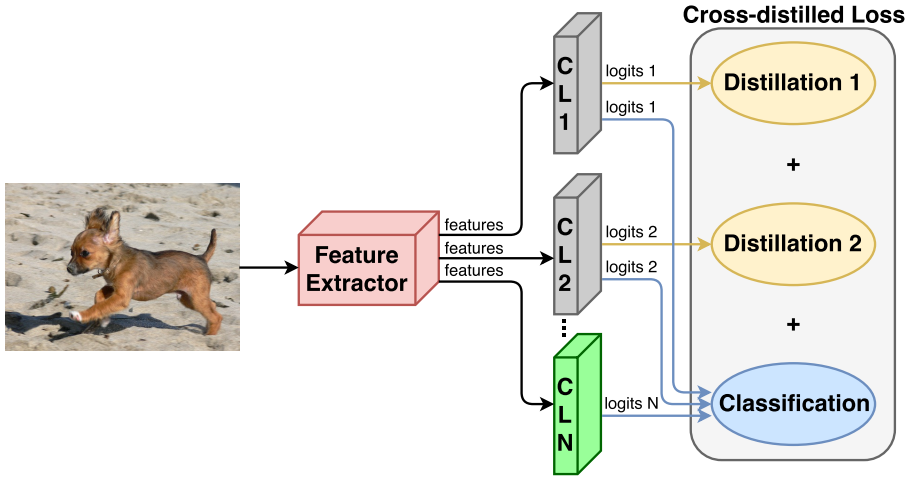

Fig. 1: **Incremental model**. Given an input image, the feature extractor produces a set of features which are used by the *classification layers* (CL-boxes) to generate a set of *logits*. Those *logits* are fed into the Cross-distilled loss function. Grey *classification layers* contain old classes and their *logits* are used for distillation and classification. The green *classification layer* (CLN) contains new classes and its *logits* are only involved in classification. (Best viewed in color.)

Alternative strategies to mitigate catastrophic forgetting include, increasing the number of layers in the network to learn features for the new classes [15, 16], or slowing down the learning rate selectively through per-parameter regularization [17]. Xiao *et al.* [18] also follow a related scheme and grow their tree-structured model incrementally as new classes are observed. The main drawback of all these approaches is the rapid increase in the number of parameters, which grows with the total number of weights, tasks, and the new layers. In contrast, our proposed model results in minimal changes to the size of the original network, as explained in Section 3.

Rebuffi *et al.* [1] present iCaRL, an incremental learning approach where the tasks of learning the classifier and the data representation are decoupled. iCaRL uses a traditional NMC to classify test samples, i.e., it maintains an auxiliary set containing old and new data samples. The data representation model, which is a standard neural network, is updated as and when new samples are available, using a combination of distillation and classification losses [8, 32]. While our approach also uses a few samples from the old classes as exemplars in the representative memory component (cf. Section 3.1), it overcomes the limitations of previous work by learning the classifier and the features jointly, in an end-to-end fashion. Furthermore, as shown in Sections 6 and 7, our new model outperforms [1].

3 Our Model

The two core components of our model are the *representative memory* and the *deep network*. The first component (Sec. 3.1) stores and manages a set of representative samples

from each class to keep its behaviour and, at the same time, to learn how to build a new representation and a classifier to separate old and new classes. The second component can be built employing any deep model designed for classification, since our approach does not require any specific characteristic. A typical architecture for classification can be seen in Fig. 1 considering only one *classification layer* and the classification loss. This *classification layer* uses the features from the feature extractor to produce a set of *logits* which are transformed into probabilities by a softmax layer. Note that, for brevity, this layer is not included in the figure. The only necessary modification is a specific loss function described in Sec. 3.2.

3.1 Representative memory

When a new class or set of classes has been added to the current model, a subset with the most representative samples from those classes is selected and stored in the representative memory. In order to test the behaviour of our incremental learning approach in different scenarios, we propose two different memory setups. The first setup considers that the representative memory has a limited capacity of K samples. As the capacity of the memory is independent of the number of classes, the more classes are stored, the fewer samples per class are retained. Therefore, the number of samples per class n is defined by $n = \lfloor K/c \rfloor$, where c is the number of classes stored in the memory and K is the capacity of the memory. The second setup stores a constant number of exemplars per class. Thus, the size of the memory grows proportionally to the number of classes.

Two operations are performed by the memory unit: selection of new samples to store, and removal of leftover samples.

Selection of new samples. The selection of the samples is based on *herding selection* [33]. This selection produces a sorted list of samples of one class based on the distance to the mean sample of that class. Given the sorted list of samples, the first n samples of the list are selected. Those selected samples are the most representative samples of the class according to the mean. This selection method has been chosen according to the results obtained in preliminary experiments by testing a set of different approaches such as random selection, hard examples selection, etc.

Note that this selection is executed only once per class, whenever a new class is added to the memory.

Removing samples. This step is performed after the training process to allocate memory for the samples from the new classes. As the samples are stored in a sorted list, this operation is trivial. The memory unit only needs to remove samples from the end of the set of samples of each class. Note that after this operation, the removed samples are never used again in our approach.

3.2 Deep network

Given an input sample, the network produces a set of probabilities according to the trained classes. Therefore, the class of an input samples is the one with the highest probability.

Architecture. The architecture of the deep model consists of the basic components shown in Fig. 1. The first component is the *feature extractor*, which is a set of layers

used to obtain a feature vector from the input. The next component is a *classification layer* which is the last fully-connected layer of the model with as many outputs as classes. This component takes the features from the *feature extractor* and produces a set of *logits*. During training, the Cross-distilled loss function takes the *logits* and produces the gradients to update the weights. At test time, the loss function is substituted by a softmax layer. Note that this softmax layer is not included in the figure for brevity.

To build our incremental model architecture, we start by creating a traditional (i.e. non-incremental) deep architecture for classification for the first set of classes. When new classes are trained, we add a new *classification layer* connected to the *feature extractor* and out Cross-distilled loss function connected to all *classification layers*, as shown in Fig. 1. Note that during the incremental training process, the architecture of the *feature extractor* does not change, only new *classification layers* are connected to it. Therefore, any architecture (or even pre-trained model) can be used with our approach just by adding the incremental classification layers and the Cross-distilled loss function when necessary.

Cross-distilled loss function. Our *cross-distilled loss* function combines a distillation loss [32], which retains the knowledge from old classes, with a multi-class cross-entropy loss, which learns to classify the classes. The distillation loss is applied to the *classification layers* of the old classes while the multi-class cross-entropy is used on all *classification layers*. By this way, our model retains the knowledge while it learns new classes and new decision boundaries between old and new classes. A sketch of the Cross-distilled loss function and its connections can be seen in the right part of the Fig. 1.

Our loss function is defined in Eq. 1:

$$L(\omega) = L_C(\omega) + \sum_{f=1}^{F} L_{D_f}(\omega), \tag{1}$$

where $L_C(\omega)$ is the cross-entropy loss applied to samples from old and new classes, L_{D_f} is the distillation loss of the f *classification layer* and F is the total number of *classification layers* for old classes (grey boxes in Fig. 1).

The cross-entropy loss $L_C(\omega)$ is defined as:

$$L_C(\omega) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} p_{ij} \log q_{ij}, \tag{2}$$

where q_i is a probability obtained by applying a softmax function to the *logits* of a *classification layer* for the sample i, p_i is the ground-truth for the sample i, N is the total number of samples and C the total number of classes.

The distillation loss $L_D(\omega)$ is defined as:

$$L_D(\omega) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{j=1}^{C} pdist_{ij} \log qdist_{ij}, \tag{3}$$

where N is the total number of samples and C the total number of classes, $pdist_i$ and $qdist_i$ are the modified versions of p_i and q_i, respectively. These modified versions

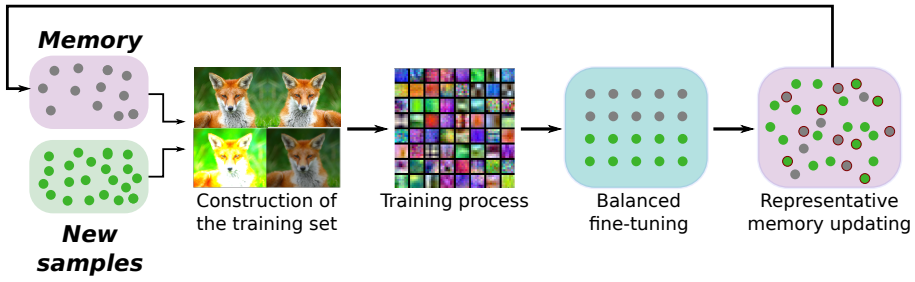

Fig. 2: **Incremental training**. Grey dots correspond to samples stored in the representative memory. Green dots correspond to samples from the new classes. Dots with red border correspond to the selected samples to be stored in the memory. (Best viewed in color.)

are obtained by raising p_i and q_i to the exponent $1/T$ as described in [32]. In this case, T is the distillation parameter. If $T > 1$, it increases the weight of smaller *logit* values and reinforce the learning of similarities between classes. Due to this fact, small logit values (generated by samples belonging to old classes), which were irrelevant to learn new features, are now increased. Therefore, it is easier to retain the knowledge as there is a larger number of logits contributing to the definition of a class. Preliminary experiments showed that the best results were obtained for a value of $T = 2$. Hence, we set T to 2 for all our experiments.

4 Incremental Learning

In this section we describe the stages performed during each incremental training. Thus, for each class or set of classes added to the model in an incremental way, we have to run the steps mentioned below. We consider as incremental step the complete process to include a new class or set of classes in the model.

Briefly, an incremental step in our approach consists of four main stages: the first stage is the construction of the training set, which prepares the training data to be used in the second stage, the training process, which fits a model given the training data. In the third stage, a fine-tuning with a subset of the training data is performed. This subset contains the same number of samples per class. Finally, in the fourth stage, the representative memory is updated to include samples from the new classes. These stages are described in detail below. An illustration of this process is shown in Fig. 2.

Construction of the training set. Our training set is composed of samples from the new classes and the exemplars from the old classes stored in the representative memory. Both sets are joined into a single set. As our approach combines two loss functions (i.e. classification and distillation) into one, for each sample we need two labels, associated to the two different loss functions. Since our distillation loss function uses the *logits* as labels to retain the knowledge as explained in Sec. 3, we perform the usual data augmentation procedure before the training, producing an augmented set by applying typical transformations (e.g. random cropping, mirroring, etc.). If this data transforma-

tion were performed during training at batch level, the distillation labels (*logits*) could be wrong as the input samples are altered.

Finally, we have to include both kinds of labels in this augmented set. For classification, we use the one-hot vector which indicates the class appearing in the image. For distillation, we use as labels the *logits* produced by every *classification layer* with old classes (grey fully-connected layers in Fig. 1). Thus, we have as many distillation labels per sample as *classification layers* with old classes. To reinforce the old knowledge, the samples from the new classes are also used for distillation. By this way, all images produce gradients for both losses and the training process is improved. Therefore, when an image is evaluated by the network, the output encodes the behaviour of the weights that compose every layer of the deep model, independently of the label, which is only used by the loss function. Then, each image of our training set will have a classification label and F distillation labels, where F is the number of *classification layers* from old classes. Note that this label extraction is performed in each incremental step and before the training process.

Previous paragraph can be better understood using an example that assumes we are running the third incremental step of our incremental model (Fig. 1). At this point the model has three *classification layers* ($N = 3$), two of them will process old classes (grey boxes), which means $F = 2$, and one of them operates with the new classes (green box). When a sample is evaluated, the *logits* produced by the F *classification layers* with old classes are used for distillation (yellow arrows) and the *logits* produced by the N *classification layers* are used for classification (blue arrows).

Training process. During the training process, our cross-distilled loss function (Eq. 1) takes the augmented training set with its corresponding labels and produces a set of gradients to optimise the deep model. Note that, during training, all weights of the model are updated. Therefore, for one sample, the features obtained from the feature extractor could change between successive incremental steps and the *classification layers* should adapt their weights to deal with the new features. This is an important difference with some other incremental approaches like [8], where the the *feature extractor* is frozen and only the *classification layers* are trained.

Balanced fine-tuning. Due to we do not store all samples from the old classes, the number of samples from old classes available for training in the previous stage can be significantly lower than those belonging to new classes. Thus, to deal with this unbalanced training scenario, we add an additional fine-tuning stage with a small learning rate and a balanced subset of samples. The new training subset contains the same number of samples per class, regardless of whether they belong to new or old classes. This subset is built by reducing the number of samples from the new classes keeping only the most representative samples from each class using the selection algorithm described in Sec. 3.1. Since we are removing samples, the model could forget the knowledge obtained during the previous training process. To solve this problem, we add a temporarily distillation loss to the *classification layer* of the new classes to retain the previous behaviour.

Representative memory updating. When the balanced fine-tuning finishes, the representative memory must be updated to include exemplars from the new classes. This operation is performed with the selection and removing functions described in Sec. 3.1.

First, the memory unit removes samples from the stored classes to allocate space for the samples from the new classes. Then, the most representative samples from the new classes are selected and stored in the memory unit according to the selection algorithm described above.

5 Implementation Details

Our models are implemented on MatConvNet [34]. For each incremental step, we perform 40 epochs, and an additional 30 epochs for balanced fine-tuning. Our learning rate for the first 40 epochs starts at 0.1, and is divided by 10 every 10 epochs. The same reduction is used in the case of fine-tuning, except that the starting rate is 0.01. We train the networks using standard stochastic gradient descent with mini-batches of 128 samples, weight decay of 0.0001 and momentum of 0.9. We apply L^2-regularisation and random noise [35] (with parameters $\eta = 0.3, \gamma = 0.55$) on the gradients to minimise overfitting.

We follow the setting suggested by He *et al.* [36] and we use different models depending on the dataset (CIFAR-100 or ImageNet in our case). This allows the architecture of the network to be adapted to the specific characteristics of the dataset. We use a 32-layer ResNet for CIFAR-100, and a 18-layer ResNet for ImageNet as the deep model. We store $K = 2,000$ distillation samples in the representative memory in both the cases. When training the model for CIFAR-100, we normalise the input data by dividing the pixel values by 255, and subtracting the mean image of the training set. In the case of ImageNet, we only perform the subtraction, without the pixel value normalisation, following the implementation of [36].

Since there are no readily-available class-incremental learning benchmarks, we follow the standard setup [1, 9] of splitting the classes of a traditional multi-class dataset into incremental batches. In all the experiments below, iCaRL refers to the final method in [1], and hybrid1 refers to their variant which uses a CNN classifier instead of NMC. LwF.MC is the multi-class implementation of LwF [8], as done in [1]. We used the publicly available implementation of iCaRL downloaded from GitHub[4]. The results for LwF.MC are also obtained from this code but removing the exemplars usage. We report results for each method as the average accuracy over all the incremental batches. Note that we do not consider the accuracy of the first batch in this average, as it does not correspond incremental learning. This is unlike the evaluation in [1], which is the reason for difference between the results we report for their method and the published results.

Data augmentation. The third stage of our approach (cf. Sec. 4) performs data augmentation before the training step. Specifically, the operations performed are:

1. *Brightness*: the intensity of the original image is altered by adding a random intensity value in the range $[-63, 63]$.
2. *Contrast normalization*: the contrast of the original image is altered by a random value in the range $[0.2, 1.8]$. The operation performed is $im_{\text{altered}} = (im - mean) \times contrast + mean$. Where im is the original image, $mean$ is the mean of the pixels per channel and $contrast$ is the random value.

[4] `https://github.com/srebuffi/iCaRL`

3. *Random cropping*: all the images (original, brightness and contrast) are randomly cropped.
4. *Mirroring*: a mirror image is computed for all images (original, brightness, contrast and crops).

Other operations applied on each dataset are specified in Sec. 6.1 for CIFAR-100 and in Sec. 7.1 for ImageNet.

6 Evaluation on CIFAR-100

We perform three types of experiments on the CIFAR-100 dataset [19]. In the first one (Sec. 6.2), we set the maximum storage capacity of our representative memory unit following the experimental protocol in [1]. The second experiment (Sec. 6.3) evaluates the methods without the fixed memory size, and instead uses a constant number samples for each of the old classes. In this experiment, the memory size grows with each incremental step, when new classes are stored in the representative memory unit. Finally, in Sec. 6.4, we perform an ablation study to analyze the influence of different components of our approach on the accuracy.

6.1 Dataset

CIFAR-100 dataset [19] is composed of 60k 32×32 RGB images of 100 classes, with 600 images per class. Every class has 500 images for training and 100 images for testing. We split the 100 classes into splits of 2, 5, 10, 20 and 50 classes with a random order. Thus, we will have 50, 20, 10, 5 and 2 incremental training steps respectively. After each incremental step, the resulting model is evaluated on the test data of all trained classes (i.e., old and new ones). Note that test data is never used during the training process, it is used only at test time. Our evaluation metric at each incremental step is the standard multi-class accuracy. We execute the experiments five times with different random class orders, reporting the average accuracy and standard deviation. In addition, we also report the average incremental accuracy (mean of the accuracy values at every incremental step). As mentioned, we do not consider the accuracy of the first step for this average as it does not represent incremental learning.

On CIFAR, we follow the data augmentation steps described in Sec. 5 and, for each training sample, generate 11 new samples: one brightness normalisation, one contrast normalisation, three random crops (applied to the original, brightness and contrast images) and six mirrors (applied to the previously generated images plus the original one).

6.2 Fixed memory size

We evaluate 5 different splits with different class order and incremental steps of 2, 5, 10, 20 and 50 classes. The class order is identical for all the evaluated methods, to ensure that the results are comparable. Tab. 1(a) summarises the results of the experiments and Fig. 3 shows the incremental steps for 2 and 5 classes. The rest of plots are included in the supplemental material. The 'Upper-Bound' result, represented in Fig. 3 as a large

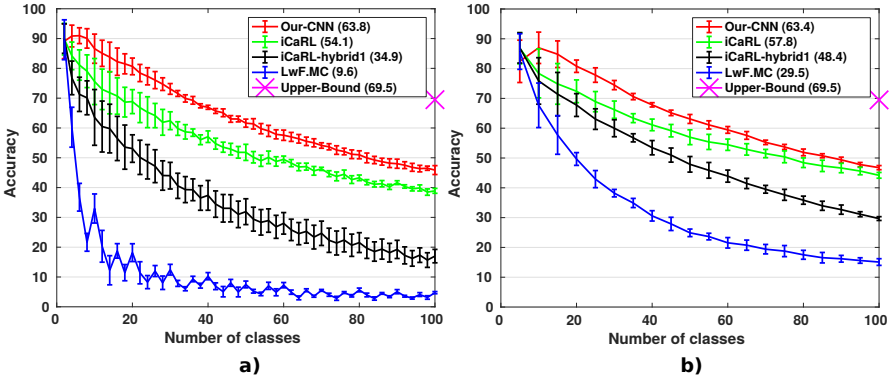

Fig. 3: **Accuracy on CIFAR-100.** Average and standard deviation of 5 executions with (a) 2 and (b) 5 classes per incremental step. Average of the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)

# classes	2	5	10	20	50
Our-CNN	**63.8 ± 1.9**	**63.4 ± 1.6**	**63.6 ± 1.3**	**63.7 ± 1.1**	60.8 ± 0.3
iCaRL	54.1 ± 2.5	57.8 ± 2.6	60.5 ± 1.6	62.0 ± 1.2	**61.8 ± 0.4**
Hybrid1	34.9 ± 4.5	48.4 ± 2.6	55.8 ± 1.8	60.4 ± 1.0	60.8 ± 0.7
LwF.MC	9.6 ± 1.5	29.5 ± 2.2	40.4 ± 2.0	47.6 ± 1.5	52.9 ± 0.6

(a) CIFAR-100

# classes	10	100
Our-CNN	**90.4**	**69.4**
iCaRL	85.0	62.5
Hybrid1	83.5	46.1

(b) ImageNet

Table 1: **Fixed memory size: accuracy on CIFAR-100 and ImageNet.** Each column represents a different number of classes per incremental step. Each row represents a different approach. The best results are marked in bold.

cross (in magenta) in the last step, is obtained by training a non-incremental model using all the classes and all their training samples.

We can observe that our end-to-end approach obtains the best results for 2, 5, 10 and 20 classes. For 50 classes, although we achieve the same score as Hybrid1 (the variant of iCaRL using CNN classifier), we are 1% lower than iCaRL. This behavior is due to the limited memory size, resulting in a heavily unbalanced training set containing 12.5 times more data from the new samples than from the old classes. To remark the statistical significance of our method's performance, we have applied a paired t-test on iCaRL. The corresponding p-values are 0.00005, 0.0005, 0.003, 0.9886 for 2, 5, 10, 20 and 50 classes respectively, what indicates that our approach is significantly better up to 20 classes.

It can be also observed that the performance of our approach remains stable across the incremental step sizes (from 2 to 20 classes per step) in Tab. 1(a), in contrast to all the other methods which are heavily-dependent on the number of classes added in each step. This is because a small number of classes at each incremental step benefits the accuracy in the early stages of the incremental learning process, as only a few classes must be classified. However, as more steps are applied to train all the classes, the accuracy of the final stages decreases.

The behavior is reversed when larger number of classes are added in each incremental step. Lower accuracy values are seen during the early stages, but this is compensated with better values in the final stages. These effects can be seen in Fig. 3, where two different number of classes per incremental step (i.e., 2 and 5) are visualized. Fig. 3 also shows that our approach is significantly better than iCaRL when a small number of classes per incremental step are employed. With larger number of classes in each step, iCaRL approaches our performance, but still remains lower overall. Our approach clearly outperforms LwF.MC in all the cases, thus highlighting the importance of the representative memory in our model.

6.3 Fixed number of samples

In this experiment, we propose to train the models using a constant number of training samples per old class. This limitation is not applied to the samples from the new classes. Therefore, we allow the memory to grow proportionally to the number of classes in contrast to the memory fixed experiment where the memory size remains constant. This experiment is useful to measure the performance of the incremental learning in an isolated scenario, that is, an scenario where the number of classes is augmented but the number of samples per class remains constant. In addition, to measure the impact of the number of samples in the accuracy, we evaluate different number of samples per class: 50, 75 and 100. To minimise the number of executions, we focus on experiments with incremental step values of 5, 10 and 20 classes. The first split for both iCaRL and our approach is run with the same order of classes. Therefore, the results will be comparable and the number of executions is controlled. Note that in this experiment, we do not execute the LwF.MC because previous experiments showed that it always obtains worse results than iCaRL. Thus, we focus the comparison on iCaRL and Hybrid1.

Tab. 2.(a) summarises the results of the experiments. The number of classes per incremental step is indicated in the first row of the table. The second row contains the number of exemplars per old class used during training. The rest of the lines show the results of the different approaches evaluated. Comparing the results between Our-CNN and both approaches developed in [1], we can see that in all scenarios our approach is better. Like in the 'fixed memory size' experiment (Sec. 6.2), our approach achieves similar average accuracy for incremental step sizes ranging from 5 to 20 classes. To measure the effect of the number of exemplars per class during the training process, we compare the results shown in Tab. 1(a) with the results shown in Tab. 2(a). For all approaches in all cases, the more exemplars used for training, the better accuracy is obtained. For 50 exemplars, the results are worse than those shown in Tab. 1 because at the early incremental steps, the number of exemplars available is lower, and the first models are undertrained. This causes a chain effect and the model obtained in the final stage is worse than expected, although there are more exemplars available.

6.4 Ablation study

We analyse here the components of our approach and demonstrate their impact on the final performance. In this case, we use the memory fixed setup for our memory unit.

# classes	5			10			20		
# img / cls	50	75	100	50	75	100	50	75	100
Our-CNN	**62.4**	**66.9**	**68.6**	**62.7**	**65.7**	**68.5**	**63.3**	**65.4**	**67.3**
iCaRL	56.5	59.9	62.2	60.0	62.3	63.7	61.9	63.0	64.0
Hybrid1	45.7	49.2	50.9	55.3	56.5	57.4	60.4	61.5	62.2

(a) Fixed number of samples

# classes	5	10	20
Our-CNN-Base	57.0	53.7	50.1
Our-CNN-DA	59.2	57.9	57.2
Our-CNN-BF	57.9	58.1	57.1
Our-CNN-Full	**63.8**	**64.0**	**63.2**
iCaRL	58.8	60.9	61.2
Hybrid1	48.7	55.1	59.8

(b) Ablation study

Table 2: **Accuracy on CIFAR-100.** Each row represents a different approach. The best results are marked in bold. See the main text for more details.

In the first experiment, we train our approach with data augmentation but without balanced fine-tuning ('Our-CNN-DA'). In the second one, we perform the reverse operation, we train our approach without data augmentation but with balanced fine-tuning ('Our-CNN-BF'). Finally, we train our model without data augmentation and balanced fine-tuning ('Our-CNN-Base'). To minimise the number of executions, we focus on experiments with incremental step values of 5, 10 and 20 classes. As in previous experiments, the first split for iCaRL and our approach is run with the same order of classes. Therefore, the results will be comparable and the number of executions is controlled.

Tab. 2(b) summarises the results for this experiment. The baseline ('Our-CNN-Base') is the worst one for all cases. However, when the data augmentation ('Our-CNN-DA') is added, the results improve in all cases, obtaining the best result for 5 classes. However, due to the unbalanced number of samples between old and new classes, with larger incremental steps it is necessary to add our balanced fine-tuning ('Our-CNN-BF'). When balanced fine-tuning ('Our-CNN-BF') is added, it improves the results in all cases, specially with big incremental steps, what remarks the importance of a balanced training set of old and new classes. Finally, we add both components to the baseline, obtaining our full model ('Our-CNN-Full'). With this final approach, we obtain the best results and a new state-of-the-art is established in this dataset for incremental learning.

7 Evaluation on ImageNet

For a fair comparison with the iCaRL and hybrid1 methods proposed in [1], we follow their protocol also on ImageNet.

7.1 Dataset

ImageNet Large-Scale Visual Recognition Challenge 2012 (ILSVRC12) [20] is an annual competition which uses a subset of ImageNet. This subset is composed of 1,000 classes with more than 1,000 images per class. In total, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images. With this dataset, we run two different experiments. In the first one, we randomly select 100 classes and divide them into splits of 10 classes with a random order at the appearing moment of the classes. In the second one, we divide the $1,000$ classes into splits of

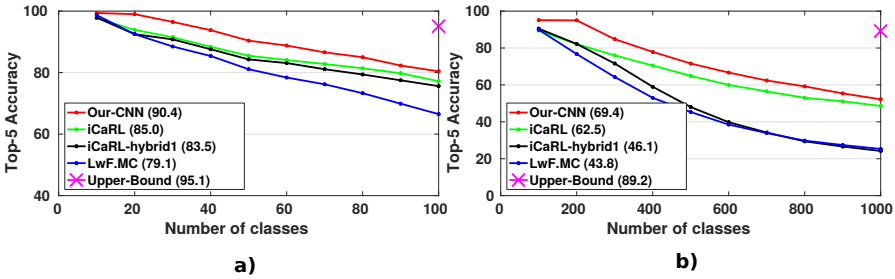

Fig. 4: **Accuracy on ImageNet.** One execution with (a) 10 and (b) 100 classes per incremental step. Average of the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)

100 classes with a random order at the appearing moment of the classes. After every incremental step, the resulting model is evaluated on the test data of all trained classes (i.e. old and new ones). Note that test data is never used during the training process, it is only used at test time. In this case, we execute the experiments once and we report the top-5 accuracy for each incremental step. In addition, we also report the average incremental accuracy described in Sec. 6.1.

In this dataset we also use the data augmentation operations described in Sec. 5 and, for each training sample, we generate its mirror image and, for all the images (originals and mirrors) we randomly apply all the operations described in Sec. 5. Therefore, after our data augmentation process, we have double number of training samples.

7.2 Fixed memory size

In this section, we perform two experiments, one with a subset of 100 classes randomly selected and incremental steps of 10 classes. In the second experiment, we use all classes with incremental steps of 100 classes. The class order is exactly the same for all evaluated methods, so the results are comparable. We compare our results with the ones obtained by iCaRL. Note that we use their code downloaded from GitHub[4] and we encoded the class order to guarantee a fair comparison.

Tab. 1.(b) summarises the results of the experiments and Fig. 4 shows the incremental steps for 10 and 100 classes. The 'Upper-Bound' result is obtained training a non-incremental model using all classes and all training samples for each class. Thus, it is represented as a pink cross in the last step in Fig. 4.

Comparing our approach to the other methods, it can be seen that in both cases we establish a new state-of-the-art , improving the previous average results more than 5%. These results suggest that our approach is suitable for large datasets with many classes. In addition, as the number of samples from new and old classes is more balanced than in CIFAR-100, our approach achieves good accuracy even with incremental steps of 100 classes.

8 Summary

This paper presented a novel approach for training CNNs in an incremental fashion by using a combination of cross-entropy and distillation loss functions that have never been used in a multi-class incremental setup. Experimental results on CIFAR-100 and ImageNet allow to settle the following main conclusions. *(i)* Our end-to-end approach is more robust than other methods like iCaRL, which relies on a sub-optimal external classifier. Thus, we have tested that this external classifier suffers from poor accuracy especially when the classes are similar with close means between them (e.g. incremental face recognition problem). And, *(ii)* the memory size and the unbalanced samples play an important role in the final accuracy. The use of an adequate memory size and the proposed balanced fine-tuning approach help to boost the method performance and achieve state-of-the-art results.

As future work, we plan to explore a new selection using a dynamic number of samples per class attending to their classification accuracy. Moreover, we want to study the effect of adding more parameters to the model during the incremental steps.

Acknowledgements

This work has been funded by project TIC-1692 (Junta de Andalucía), TIN2016-80920R (Spanish Ministry of Science and Technology) and University of Málaga (Campus de Excelencia Internacional Andalucía Tech). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

1. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: iCaRL: Incremental classifier and representation learning. In: CVPR. (2017)
2. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of Learning and Motivation **24** (1989) 109 – 165
3. Goodfellow, I., Mirza, M., Xiao, D., Courville, A., Bengio, Y.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. ArXiv e-prints, arXiv 1312.6211 (2013)
4. French, R.M.: Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. In: Cognitive Science Society Conf. (1994)
5. Ans, B., Rousset, S., French, R.M., Musca, S.: Self-refreshing memory in artificial neural networks: Learning temporal sequences without catastrophic forgetting. Connection Science **16**(2) (2004) 71–99
6. Ratcliff, R.: Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychological review **97**(2) (1990) 285
7. Lopez-Paz, D., Ranzato, M.A.: Gradient episodic memory for continual learning. In: NIPS. (2017)
8. Li, Z., Hoiem, D.: Learning without forgetting. PAMI (2018)
9. Shmelkov, K., Schmid, C., Alahari, K.: Incremental learning of object detectors without catastrophic forgetting. In: ICCV. (2017)

10. Ruping, S.: Incremental learning with support vector machines. In: ICDM. (2001)
11. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: NIPS. (2000)
12. Jung, H., Ju, J., Jung, M., Kim, J.: Less-forgetting learning in deep neural networks. ArXiv e-prints, arXiv 1607.00122 (2016)
13. Furlanello, T., Zhao, J., Saxe, A.M., Itti, L., Tjan, B.S.: Active long term memory networks. ArXiv e-prints, arXiv 1606.02355 (2016)
14. Triki, A.R., Aljundi, R., Blaschko, M.B., Tuytelaars, T.: Encoder based lifelong learning. In: ICCV. (2017)
15. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. ArXiv e-prints, arXiv 1606.04671 (2016)
16. Terekhov, A.V., Montone, G., O'Regan, J.K.: Knowledge transfer in deep block-modular neural networks. In: Biomimetic and Biohybrid Systems. (2015) 268–279
17. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. Proc. National Academy of Sciences **114**(13) (2017) 3521–3526
18. Xiao, T., Zhang, J., Yang, K., Peng, Y., Zhang, Z.: Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In: ACM Multimedia. (2014)
19. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report, University of Toronto (2009)
20. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3) (2015) 211–252
21. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3) (1995) 273–297
22. Thrun, S.: Lifelong learning algorithms. In: Learning to Learn. (1998) 181–209
23. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Mishra, B.D., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., Welling, J.: Never-ending learning. In: AAAI. (2015)
24. Chen, X., Shrivastava, A., Gupta, A.: NEIL: Extracting visual knowledge from web data. In: ICCV. (2013)
25. Divvala, S., Farhadi, A., Guestrin, C.: Learning everything about anything: Webly-supervised visual concept learning. In: CVPR. (2014)
26. Mensink, T., Verbeek, J., Perronnin, F., Csurka, G.: Distance-based image classification: Generalizing to new classes at near-zero cost. PAMI **35**(11) (2013) 2624–2637
27. Ristin, M., Guillaumin, M., Gall, J., Gool, L.V.: Incremental learning of ncm forests for large-scale image classification. In: CVPR. (2014)
28. Ruvolo, P., Eaton, E.: ELLA: An efficient lifelong learning algorithm. In: ICML. (2013)
29. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)
30. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: NIPS. (2014)
31. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. PAMI **35**(8) (2013) 1798–1828
32. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS workshop. (2014)
33. Welling, M.: Herding dynamical weights to learn. In: ICML. (2009)

34. Vedaldi, A., Lenc, K.: MatConvNet – Convolutional Neural Networks for MATLAB. In: ACM Multimedia. (2015)
35. Neelakantan, A., Vilnis, L., Le, Q.V., Sutskever, I., Kaiser, L., Kurach, K., Martens, J.: Adding gradient noise improves learning for very deep networks. ArXiv e-prints, arXiv 1511.06807 (2017)
36. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
37. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continuum learning. CoRR **abs/1706.08840** (2017)

Appendix 1: Comparison with Gradient Episodic Memory

Lopez-Paz et. al [37] present a novel approach for continual learning. This approach is based on an episodic memory which stores a subset of samples from each task trained. During the continual or incremental learning, this memory stores new samples from the previously trained tasks. To train new classes and, at the same time, to retain the knowledge from the previously trained classes, the loss functions are used as inequality constraints. Thus, the losses can decrease due to an improvement in the solution but they cannot increase, avoiding worse solutions and a lost of previous knowledge.

Using their released code[5], we perform a comparison with our method using two different setups. In the first experiment, which is the one proposed in [37], the task of each test sample is known beforehand and only the probabilities from this task are used at test time. In the second experiment, the task is unknown and the probabilities from all tasks are taken into account during the test process. Therefore, better accuracies are expected in the first experiment as only scores for the classes belonging to the correct task are evaluated.

8.1 A priory known task

For this experiment, the setup proposed in [37] is used. In this experiment, the 100 classes included in CIFAR-100 are split into groups of 5 classes and trained in an incremental way. Tab. 1 shows the final accuracy on CIFAR-100 with different memory sizes. This final accuracy is obtained at the end of each incremental learning step by averaging the individual accuracy of each task. The class order is identical for all the evaluated methods, to ensure that the results are comparable.

According to the results, our approach performs better in all cases, specially when the memory size is small. Therefore, our method is able to retain better the knowledge per task compared to GEM.

memory size	200	1280	2560	5120
GEM	48.0	62.6	64.6	67.8
Ours	**77.9**	**86.1**	**88.4**	**90.7**

Table 1: Accuracy for different memory sizes on CIFAR-100

8.2 Unknown task

In this experiment, we use the setup explained in [1]. Again, the 100 classes included in CIFAR-100 are split into groups of 5 classes and trained in an incremental way. However, during test time, the task index is unknown and the method must differentiate by itself between classes from different tasks. Fig. 1 shows the incremental accuracy per step on CIFAR-100 with a memory size of 2000 samples. This incremental accuracy is obtained at the end of the incremental step on the test samples from the trained classes (previous and new). The class order is identical for all the evaluated methods, to ensure that the results are comparable. The 'Upper-Bound' result, represented in Fig. 1 as

[5] `https://github.com/facebookresearch/GradientEpisodicMemory`

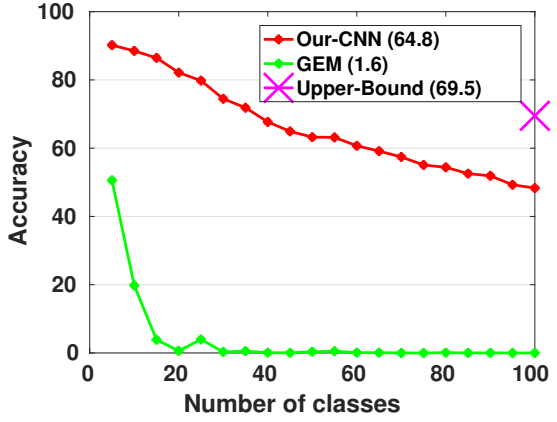

Fig. 1: **Accuracy on CIFAR-100.** Execution with 5 classes per incremental step. Average of the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)

a large cross (in magenta) in the last step, is obtained by training a non-incremental model using all the classes and all their training samples.

According to the results, our method is able to obtain good results without knowing the task of each test sample. The accuracy decreases with the number of classes as it has to retain more knowledge from previous classes and the number of samples from those classes is reduced. Note that the number of samples is fixed to 2000. Thus, if more classes are included in the memory, the number of samples per class is reduced.

Focusing on GEM's results, the accuracy decreases drastically with the number of classes being 0% with 30 classes or more. Thus, in a real scenario where the task is unknown beforehand, GEM performs worse than our approach.

Appendix 2: Comparison with similar classes

Using an external classifier (like NCM in iCaRL [1]), instead of an end-to-end approach, should produce lower results when the dataset contains similar classes. For example, in a face recognition problem trained in an incremental way, all classes have a similar mean as only small details of the face change. Therefore, an end-to-end classifier trained together with the feature extractor should obtain better results, as during the training process it has learned to deal with those small differences between classes.

To validate this hypothesis, we perform an experiment with a subset of 10 vehicles included in CIFAR-100. We use the standard training/test sets with 5 incremental steps of 2 classes each. For ImageNet, we use a subset of 120 dog breeds running 12 incremental steps of 10 classes each. For the cars experiment we use a memory size of 200 samples and for the dog breeds we use 2400 samples.

Fig. 1.a shows the results for the cars experiment, and Fig. 1.b shows the results for the dog breeds experiment. In both cases, our end-to-end approach achieves the best results with big differences on average accuracy. Comparing the average accuracy for the cars experiment, our model obtains an average accuracy of 73.3% while iCaRL obtains 47.5%. In this case, Hybrid1 obtains similar results to iCaRL what also validates our hypothesis about the classifiers. Focusing on the dog breeds experiment, iCaRL obtains lower accuracy (47%) than our approach (57%). In this case, Hybrid1 performs worse than iCaRL but the differences are smaller than in ImageNet with 1000 classes.

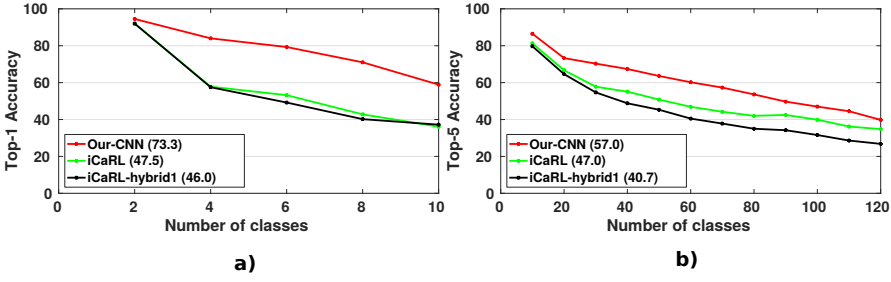

Fig. 1: **Accuracy on subsets with similar classes.** (a) Two classes per incremental step on 10 types of vehicles included in CIFAR-100. (b) Ten classes per incremental step on 120 dog breeds included in ImageNet. Average of the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)