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, Málaga, Spain
[2] Department of Computing and Numerical Analysis, University of Córdoba, Córdoba, Spain
[3] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 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. 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 are learned continually. For example, a face recognition system must handle new faces to identify new people. This task needs to be accomplished without having to re-learn faces already learned. 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 samples (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 [2, 8, 10, 16–18, 22, 23, 30]. 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 [4, 9, 13, 14, 16, 23, 26, 28, 30, 32, 34, 37] satisfy all these constraints. They often decouple the classifier and representation learning tasks [23], or are limited to very specific situations, e.g., learning from new datasets but not new classes related to the old ones [9, 13, 16, 34], or particular problems, e.g., object detection [30]. Some of them [4, 26] are tied to traditional classifiers such as SVMs and are unsuitable for deep learning architectures. Others [14, 28, 32, 37] 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 Sec. 3.1). The model 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 Sec. 3.2). As detailed in Sec. 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 [15] and ImageNet [27] (see Sec. 6 and Sec. 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 through deep learning frameworks, in addition to training classifiers.

Traditional approaches. Initial methods for incremental learning targeted the SVM classifier [6], exploiting its core components: support vectors and Karush-Kuhn-Tucker conditions. Some of these [26] 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 [4] 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 [33] or never-ending [5, 7, 20] 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., [5], or rely on a fixed representation, e.g., [7]. Methods such as [19, 25, 29] 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 [29], or *(ii)* using a nearest mean classifier (NMC) [19], or a random forest variant [25]. 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 [3, 24, 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 [2, 8, 10, 16–18, 22, 23, 30]. Initial attempts to overcome this issue were aimed at connectionist networks [2, 8, 18], 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 [16] using distillation loss in combination with the standard cross-entropy loss. Distillation loss, which was originally proposed to transfer knowledge between different neural networks [12], was adapted to maintain the responses of the network on the old tasks whilst updating it with new training samples [16]. 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 [23, 34] demonstrated this weakness of [16] 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 [13], propose to freeze some of the layers corresponding to the original model, thereby limiting its adaptability to new data. Triki *et al.* [34] build on the method in [16] 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 [16]. Distillation loss was also adopted for learning object detectors incrementally [30]. Despite its success for object detection, the utility of this specific architecture for more general incremental learning scenarios we target here is unclear.

Alternative strategies to mitigate catastrophic forgetting include, increasing the number of layers in the network to learn features for the new classes [28, 32], or slowing

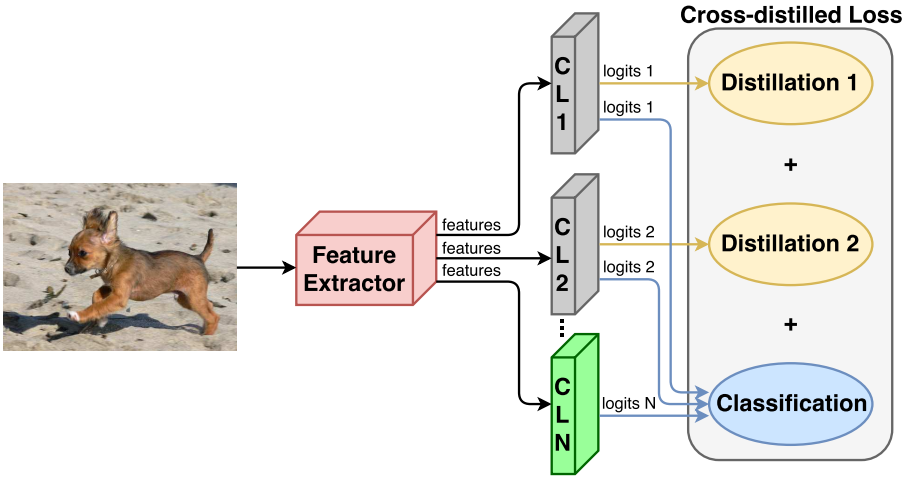

Fig. 1: **Our incremental model**. Given an input image, the feature extractor produces a set of features which are used by the *classification layers* (CLi blocks) to generate a set of *logits*. Grey *classification layers* contain old classes and their *logits* are used for distillation and classification. The green *classification layer* (CLN block) contains new classes and its *logits* are involved only in classification. (Best viewed in color.)

down the learning rate selectively through per-parameter regularization [14]. Xiao *et al.* [37] 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 Sec. 3.

Rebuffi *et al.* [23] 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 [12, 16]. While our approach also uses a few samples from the old classes as exemplars in the representative memory component (cf. Sec. 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 Sec. 6 and Sec. 7, our new model outperforms [23].

3 Our Model

Our end-to-end approach uses a deep network trained with a cross-distilled loss function, i.e., cross-entropy together with distillation loss. The network can be based on the architecture of most deep models designed for classification, since our approach does not require any specific properties. A typical architecture for classification can be

seen in Fig. 1, with one *classification layer* and a classification loss. This *classification layer* uses features from the feature extractor to produce a set of *logits* which are transformed into class scores by a softmax layer (not shown in the figure). The only necessary modification is the loss function, described in Sec. 3.2. To help our model retain the knowledge acquired from the old classes, we use a representative memory (Sec. 3.1) that stores and manages the most representative samples from the old classes. In addition to this we perform data augmentation and a balanced fine-tuning (Sec. 4). All these components put together allow us to get state-of-the-art results.

3.1 Representative memory

When a new class or set of classes is added to the current model, a subset with the most representative samples from them is selected and stored in the representative memory. We investigate two memory setups in this work. The first setup considers a memory with a limited capacity of K samples. As the capacity of the memory is independent of the number of classes, the more classes stored, the fewer samples retained per class. The number of samples per class, n, is thus given by $n = \lfloor K/c \rfloor$, where c is the number of classes stored in memory, and K is the memory capacity. The second setup stores a constant number of exemplars per class. Thus, the size of the memory grows with the number of classes.

The representative memory unit performs two operations: selection of new samples to store, and removal of leftover samples.

Selection of new samples. This is based on *herding selection* [36], which 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. These samples are most representative of the class according to the mean. This selection method was chosen based on our experiments testing different approaches, such as random selection, histogram of the distances from each sample to the class mean, as shown in Sec. 6.3. The selection is performed 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 sample set of each class. Note that after this operation, the removed samples are never used again.

3.2 Deep network

Architecture. The network is composed of several components, as illustrated in Fig. 1. The first component is a *feature extractor*, which is a set of layers to transform the input image into a feature vector. The next component is a *classification layer* which is the last fully-connected layer of the model, with as many outputs as the number of classes. This component takes the features and produces a set of *logits*. During the training phase, gradients to update the weights of the network are computed with these *logits* through our cross-distilled loss function. At test time, the loss function is replaced by a softmax layer (not shown in the figure).

To build our incremental learning framework, we start with 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* corresponding to these classes, and connect it to the *feature extractor* and the component for computing the cross-distilled loss, as shown in Fig. 1. Note that the architecture of the *feature extractor* does not change during the incremental training process, and 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. This combines a distillation loss [12], which retains the knowledge from old classes, with a multi-class cross-entropy loss, which learns to classify the new 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*. This allows the model to update the decision boundaries of the classes. The loss computation is illustrated in Fig. 1. The cross-distilled loss function $L(\omega)$ is defined as:

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

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

The cross-entropy loss $L_C(\omega)$ is given by:

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

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

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

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

where $pdist_i$ and $qdist_i$ are modified versions of p_i and q_i, respectively. They are obtained by raising p_i and q_i to the exponent $1/T$, as described in [12], where T is the distillation parameter. When $T = 1$, the class with the highest score influences the loss significantly, e.g., more than 0.9 from a maximum of 1.0, and the remaining classes with low scores have minimal impact on the loss. However, with $T > 1$, the remaining classes have a greater influence, and their higher loss values must be minimized. This forces the network to learn a more fine grained separation between them. As a result, the network learns a more discriminative representation of the classes. Based on our empirical results, we set T to 2 for all our experiments.

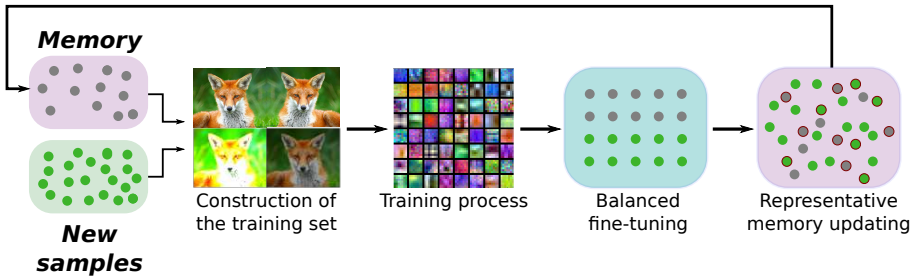

Fig. 2: **Incremental training**. Grey dots correspond to samples stored in the representative memory. Green dots correspond to samples from the new classes. Dots with red border correspond to the selected samples to be stored in the memory. (Best viewed in color.)

4 Incremental Learning

An incremental learning step in our approach consists of four main stages, as illustrated in Fig. 2. 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. We now describe these stages in detail.

Construction of the training set. Our training set is composed of samples from the new classes and exemplars from the old classes stored in the representative memory. As our approach uses two loss functions, i.e., classification and distillation, we need two labels for each sample, associated with the two losses. 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, samples from the new classes are also used for distillation. This way, all images produce gradients for both the losses. Thus, 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 its label. Each image of our training set will have a classification label and F distillation labels; cf. Eq. 1. Note that this label extraction is performed in each incremental step.

Consider an example scenario to better understand this step, where we are performing the third incremental step of our model (Fig. 1). At this point the model has three *classification layers* ($N = 3$), two of them will process old classes (grey boxes), i.e., $F = 2$, and one of them operates on the new classes (green box). When a sample is evaluated, the *logits* produced by the two *classification layers* with the old classes are used for distillation (yellow arrows), and the *logits* produced by the three *classification layers* are used for classification (blue arrows).

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 the weights of the model are updated. Thus, for any sample, features obtained from the feature extractor are likely to change between successive incremental steps, and the *classification layers* should adapt their weights to deal with these new features. This is an important difference with some other incremental approaches like [16], where the the *feature extractor* is frozen and only the *classification layers* are trained.

Balanced fine-tuning. Since we do not store all the samples from the old classes, samples from these classes available for training can be significantly lower than those from the new classes. 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, according to the selection algorithm described in Sec. 3.1. With this removal of samples from the new classes, the model can potentially forget knowledge acquired during the previous training step. We avoid this by adding a temporary distillation loss to the *classification layer* of the new classes.

Representative memory updating. After the balanced fine-tuning step, the representative memory must be updated to include exemplars from the new classes. This is performed with the selection and removing operations described in Sec. 3.1. First, the memory unit removes samples from the stored classes to allocate space for 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.

5 Implementation Details

Our models are implemented on MatConvNet [35]. 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-regularization and random noise [21] (with parameters $\eta = 0.3, \gamma = 0.55$) on the gradients to minimize overfitting.

Following the setting suggested by He *et al.* [11], we use dataset-specific CNN/deep models. This allows the architecture of the network to be adapted to 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 $K = 2000$ distillation samples in the representative memory in both cases. When training the model for CIFAR-100, we normalize 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 normalization, following the implementation of [11].

Since there are no readily-available class-incremental learning benchmarks, we follow the standard setup [23,30] 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 [23], and hybrid1 refers to their variant, which uses a CNN classifier instead of NMC. LwF.MC is the multi-class implementation of LwF [16], as done in [23]. We used the publicly available implementation of iCaRL from GitHub[4]. The results for LwF.MC are also obtained from this code, without the exemplar 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 to incremental learning. This is unlike the evaluation in [23], 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 contrast value.
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 for CIFAR-100 and in Sec. 7 for ImageNet.

6 Evaluation on CIFAR-100

We perform three types of experiments on the CIFAR-100 dataset. In the first one (Sec. 6.1), we set the maximum storage capacity of our representative memory unit, following the experimental protocol in [23]. The second experiment (Sec. 6.2) evaluates the methods without a fixed memory size, and uses a constant number samples for each of the old classes instead. Here, the memory size grows with each incremental step, when new classes are stored in the representative memory unit. Finally, in Sec. 6.3, we perform an ablation study to analyze the influence of different components of our approach on the accuracy.

Dataset. CIFAR-100 dataset [15] 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 divide 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 composed of all the trained classes, i.e., old and new ones. 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 report the average incremental accuracy (mean of the accuracy values at every incremental step). As mentioned earlier, we do not con-

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

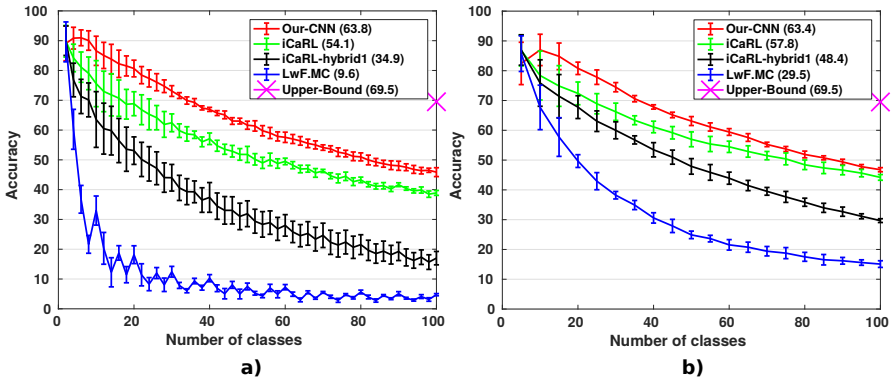

Fig. 3: **Accuracy on CIFAR-100.** Average and standard deviation of 5 executions with (a) 2 and (b) 5 classes per incremental step. Average of the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)

sider 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 normalization, one contrast normalization, three random crops (applied to the original, brightness and contrast images) and six mirrors (applied to the previously generated images and the original one).

6.1 Fixed memory size

We evaluate five 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 appendix [1]. The 'Upper-Bound' result, shown in Fig. 3 with 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.

We 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 behaviour 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 highlight the statistical significance of our method's performance compared to iCaRL, we performed a paired t-test on the results obtained for CIFAR-100. The corresponding p-values are 0.00005, 0.0005, 0.003, 0.0077, 0.9886 for 2, 5, 10, 20, and 50 classes respectively, which shows that the improvement of our method over iCaRL is statistically significant ($p < 0.01$) in all cases, except for 50 classes where both methods show similar performance.

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

# 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
LwF.MC	79.1	43.8

(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.

the other methods, which are 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 behaviour 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 (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.

6.2 Fixed number of samples

In this experiment, we 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. Thus, we allow the memory to grow in proportion to the number of classes, in contrast to the fixed memory experiment, where the memory size remains constant. Additionally, to measure the impact of the number of samples in the accuracy, we evaluate different number of samples per class: 50, 75 and 100. We focus on experiments with incremental step values of 5, 10 and 20 classes. We consider the same class order for both iCaRL and our approach to ensure that the results are comparable. We focus the comparison on iCaRL and Hybrid1 in this experiment, as LwF.MC shows a lower performance than these two methods; see Sec. 6.1.

Tab. 2(a) summarizes the results of these 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 remaining rows show the results of the different approaches evaluated. Comparing the results between Our-CNN and the methods developed in [23], we see that in all scenarios our approach performs better. As in the 'fixed memory size' experiment (Sec. 6.1), our approach achieves a similar average accuracy for incremental step sizes ranging from 5 to 20 classes, e.g., 62.4, 62.7, 63.3 with 50 exemplars per class, showing its stability. To measure the im-

# 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.

pact of the number of exemplars per class on the training, we compare the results in Tab. 1(a) with those in Tab. 2(a). In all cases, the more the exemplars used for training, the better the accuracy obtained. For 50 exemplars, the results are worse than those in Tab. 1 because in the early incremental steps, the number of exemplars available is lower, and these initial models are under trained. This causes a chain effect, and the model obtained in the final stage is worse than expected, even when more exemplars are available.

6.3 Ablation studies

We now analyze the components of our approach and demonstrate their impact on the final performance. All these ablation studies are performed with the fixed memory setup. We first evaluate the sample selection strategy with an experiment using incremental steps of 10 classes and three methods for selecting samples: herding, random and histogram selection. Herding is our selection method, presented in Sec. 3.1. Random selection refers to choosing samples to be stored in memory randomly. In the histogram selection strategy, samples are chosen according to their distance to the mean of their class. We first compute a histogram of distances, with ten bins, and assign each sample to one of these bins. We then select samples from each bin according to the proportion of samples it contains. From the results (herding: 63.6%, random: 63.1%, and histogram: 59.1%), herding and random selection strategies show the best performance.

In the following ablation study, we analyze the impact of augmentation and fine-tuning. We first train our model with data augmentation, but without balanced fine-tuning ('Our-CNN-DA'). In the second experiment, we train 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'). Here, 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, to ensure that the results are comparable. Tab. 2(b) and Fig. 4 summarise the results for this study. 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 (59.2). However, due to the unbalanced number of samples between the 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

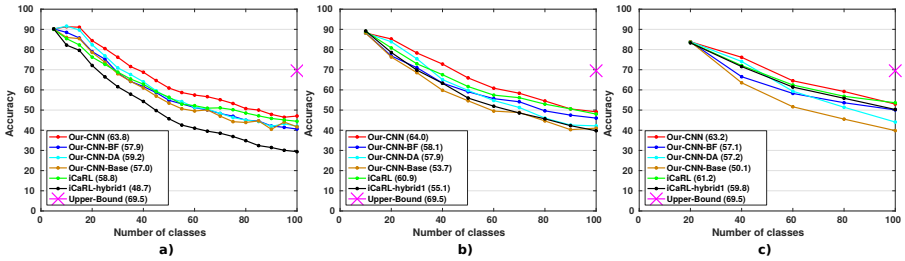

Fig. 4: **Ablation study with CIFAR-100.** Results for (a) 5, (b) 10, and (c) 20 classes. The average over all the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)

steps, which highlights the importance of a balanced training set. Finally, when both the components are added to the baseline, obtaining our full model ('Our-CNN-Full'), we observe the best results and a new state-of-the-art is established on this dataset for incremental learning.

7 Evaluation on ImageNet

Dataset. ImageNet Large-Scale Visual Recognition Challenge 2012 (ILSVRC12) [27] is an annual competition which uses a subset of ImageNet. This subset is composed of 1000 classes with more than 1000 images per class. In total, there are roughly 1.2 million training images, 50k validation images, and 150k testing images. We run two experiments with this dataset. In the first one, we randomly select 100 classes, and divide them into splits of 10 classes selected randomly. In the second one, we divide the 1000 classes into splits of 100 classes randomly selected. Note that the same set of classes are considered for all the approaches to ensure that the results are comparable. After every incremental step, the resulting model is evaluated on test data composed of all the trained classes. We execute the experiments once and report the top-5 accuracy for each incremental step. We also report the average incremental accuracy described in Sec. 6.

We use data augmentation described in Sec. 5, and for each training sample, generate its mirror image, and then randomly apply transformations (cf. Sec. 5) for all the images (original and mirror). Thus, with our data augmentation, we double the number of training samples.

Fixed memory size. We maintain identical class order for all the evaluated methods, to ensure that the results are comparable. We also follow the protocol in [23] for a fair comparison with iCaRL and hybrid1. Tab. 1(b) summarizes the results of this fixed memory size experiment, and Fig. 5 shows the incremental steps with 10 and 100 classes. The 'Upper-Bound' result, shown with a cross in the figure, is obtained by training a non-incremental model using the training samples for all the classes. From the results, we observe that in both cases we establish a new state-of-the-art , improving the previous average results by more than 5%. This suggests that our approach is also suitable for large datasets with many classes. In addition, as the number of samples from new and

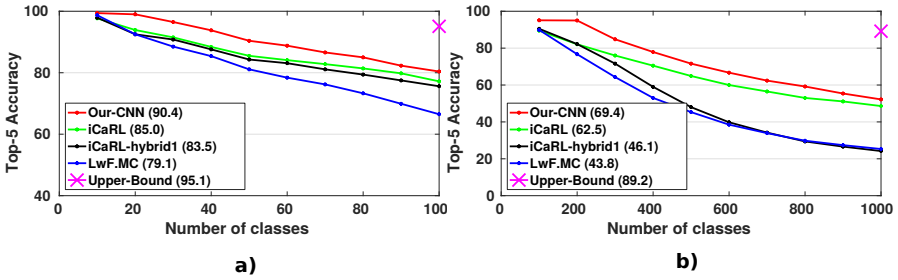

Fig. 5: **Accuracy on ImageNet.** One execution with (a) 10 and (b) 100 classes per incremental step. Average of the incremental steps is shown in parentheses for each method. (Best viewed in pdf.)

old classes is more balanced than in CIFAR-100, our approach achieves good accuracy even with large incremental steps of 100 classes.

8 Summary

This paper presents a novel approach for training CNNs in an incremental fashion using a combination of cross-entropy and distillation loss functions. Experimental results on CIFAR-100 and ImageNet presented in the paper lead to the following conclusions. *(i)* Our end-to-end approach is more robust than other recent methods, such as iCaRL, relying on a sub-optimal, independently-learned external classifier. *(ii)* Representative memory, its size, and unbalanced training sets play an important role in the final accuracy. As part of future work we plan to explore new sample selection strategies, using a dynamic number of samples per class.

Acknowledgements. This work was supported in part by the projects TIC-1692 (Junta de Andalucía), TIN2016-80920R (Spanish Ministry of Science and Tech.), ERC advanced grant ALLEGRO, and EVEREST (no. 5302-1) funded by CEFIPRA. We gratefully acknowledge the support of NVIDIA Corporation with the donation of a Titan X Pascal GPU used for this research.

References

1. Supplementary material. Also available in the arXiv technical report. `https://github.com/fmcp/EndToEndIncrementalLearning`
2. 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), 71–99 (2004)
3. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. PAMI **35**(8), 1798–1828 (2013)
4. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: NIPS (2000)

5. Chen, X., Shrivastava, A., Gupta, A.: NEIL: Extracting visual knowledge from web data. In: ICCV (2013)
6. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3), 273–297 (1995)
7. Divvala, S., Farhadi, A., Guestrin, C.: Learning everything about anything: Webly-supervised visual concept learning. In: CVPR (2014)
8. French, R.M.: Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. In: Cognitive Science Society Conf. (1994)
9. Furlanello, T., Zhao, J., Saxe, A.M., Itti, L., Tjan, B.S.: Active long term memory networks. ArXiv e-prints, arXiv 1606.02355 (2016)
10. 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)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
12. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS workshop (2014)
13. Jung, H., Ju, J., Jung, M., Kim, J.: Less-forgetting learning in deep neural networks. ArXiv e-prints, arXiv 1607.00122 (2016)
14. 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), 3521–3526 (2017)
15. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
16. Li, Z., Hoiem, D.: Learning without forgetting. PAMI (2018)
17. Lopez-Paz, D., Ranzato, M.A.: Gradient episodic memory for continual learning. In: NIPS (2017)
18. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of Learning and Motivation **24**, 109 – 165 (1989)
19. Mensink, T., Verbeek, J., Perronnin, F., Csurka, G.: Distance-based image classification: Generalizing to new classes at near-zero cost. PAMI **35**(11), 2624–2637 (2013)
20. 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)
21. 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)
22. Ratcliff, R.: Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychological review **97**(2), 285 (1990)
23. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: iCaRL: Incremental classifier and representation learning. In: CVPR (2017)
24. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS (2015)
25. Ristin, M., Guillaumin, M., Gall, J., Gool, L.V.: Incremental learning of ncm forests for large-scale image classification. In: CVPR (2014)
26. Ruping, S.: Incremental learning with support vector machines. In: ICDM (2001)
27. 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. IJCV **115**(3), 211–252 (2015)

28. 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)
29. Ruvolo, P., Eaton, E.: ELLA: An efficient lifelong learning algorithm. In: ICML (2013)
30. Shmelkov, K., Schmid, C., Alahari, K.: Incremental learning of object detectors without catastrophic forgetting. In: ICCV (2017)
31. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: NIPS (2014)
32. Terekhov, A.V., Montone, G., O'Regan, J.K.: Knowledge transfer in deep block-modular neural networks. In: Biomimetic and Biohybrid Systems (2015)
33. Thrun, S.: Lifelong Learning Algorithms, pp. 181–209. Springer US (1998)
34. Triki, A.R., Aljundi, R., Blaschko, M.B., Tuytelaars, T.: Encoder based lifelong learning. In: ICCV (2017)
35. Vedaldi, A., Lenc, K.: MatConvNet – Convolutional Neural Networks for MATLAB. In: ACM Multimedia (2015)
36. Welling, M.: Herding dynamical weights to learn. In: ICML (2009)
37. 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)