iCaRL: Incremental Classifier and Representation Learning

Abstract

A major open problem on the road to artificial intelligence is the development of incrementally learning systems that learn about more and more concepts over time from a stream of data. In this work, we introduce a new training strategy, iCaRL, that allows learning in such a class-incremental way: only the training data for a small number of classes has to be present at the same time and new classes can be added progressively.
iCaRL learns strong classifiers and a data representation simultaneously. This distinguishes it from earlier works that were fundamentally limited to fixed data representations and therefore incompatible with deep learning architectures. We show by experiments on CIFAR-100 and ImageNet ILSVRC 2012 data that iCaRL can learn many classes incrementally over a long period of time where other strategies quickly fail.

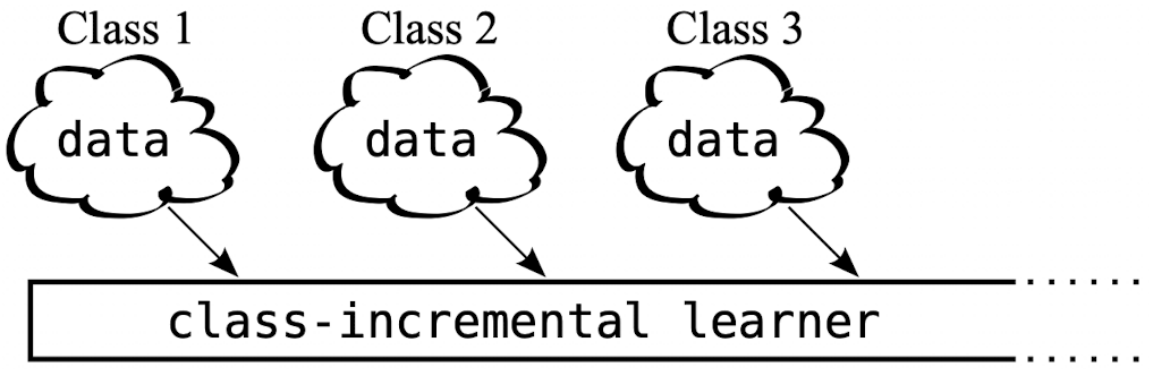

Figure 1: Class-incremental learning: an algorithm learns continuously from a sequential data stream in which new classes occur. At any time, the learner is able to perform multi-class classification for all classes observed so far.

1. Introduction

Natural vision systems are inherently incremental: new visual information is continuously incorporated while existing knowledge is preserved. For example, a child visiting the zoo will learn about many new animals without forgetting the pet it has at home. In contrast, most artificial object recognition systems can only be trained in a batch setting, where all object classes are known in advance and they the training data of all classes can be accessed at the same time and in arbitrary order.

As the field of computer vision moves closer towards artificial intelligence it becomes apparent that more flexible strategies are required to handle the large-scale and dynamic properties of real-world object categorization situations. At the very least, a visual object classification system should be able to incrementally learn about new classes, when training data for them becomes available. We call this scenario class-incremental learning.

Formally, we demand the following three properties of an algorithm to qualify as class-incremental:

i) it should be trainable from a stream of data in which examples of different classes occur at different times,

ii) it should at any time provide a competitive multi-class classifier for the classes observed so far,

iii) its computational requirements and memory footprint should remain bounded, or at least grow very slowly, with respect to the number of classes seen so far.

i) 다른 클래스의 예가 다른 시간에 발생하는 데이터 스트림에서 훈련할 수 있어야 합니다,

ii) 지금까지 관찰한 수업에 대해 언제든지 경쟁력 있는 다계층 분류기를 제공해야 하고,

iii) 계산 요구사항과 메모리 풋프린트는 지금까지 본 클래스 수와 관련하여 제한된 상태로 유지되거나 최소한 매우 느리게 증가해야 합니다.

The first two criteria express the essence of class-incremental learning. The third criterion prevents trivial(자명한) algorithms, such as storing all training examples and retraining an ordinary multi-class classifier whenever new data becomes available. Interestingly, despite the vast progress that image classification has made over the last decades, there is not a single satisfactory class-incremental learning algorithm these days. / Most existing multi-class techniques simply violate i) or ii) as they can only handle a fixed number of classes and/or need all training data to be available at the same time. Naively, one could try to overcome this by training classifiers from class-incremental data streams, e.g. using stochastic gradient descent optimization. This, however, will cause the classification accuracy to quickly deteriorate, an effect known in the literature as catastrophic forgetting or catastrophic interference [22]. / The few existing techniques that do fulfill the above properties are principally limited to situations with a fixed data representation. They cannot be extended to deep architectures that learn classifiers and feature representations at the same time and are therefore not competitive anymore in terms of classification accuracy. More related work is discussed in Section 3.

In this work, we introduce iCaRL (incremental classifier and representation learning), a practical strategy for simultaneously learning classifiers and a feature representation in the class-incremental setting. Based on a careful analysis of the shortcomings of existing approaches, we introduce three main components that in combination allow iCaRL to fulfill all criteria put forth above. These three components are:

- classification by a nearest-mean-of-exemplars rule,
- prioritized exemplar selection based on herding,
- representation learning using knowledge distillation and prototype rehearsal.

We explain the details of these steps in Section 2, / and subsequently put them into the context of previous work in Section 3. / In Section 4 we report on experiments on the CIFAR and ImageNet datasets that show that iCaRL is able to class-incrementally learn over a long periods of time, where other methods quickly fail. / Finally, we conclude in Section 5 with a discussion of remaining limitations and future work.

2. Method

In this section we describe iCaRL's main components and explain how their combination allows true class-incremental learning. Section 2.1 explains the underlying architecture and gives a high-level overview of the training and classification steps. Sections 2.2 to 2.4 then provides the algorithmic details and explains the design choices.

2.1. Class-Incremental Classifier Learning

the underlying architecture and gives a high-level overview of the training and classification steps.

iCaRL learns classifiers and a feature representation simultaneously from on a data stream in class-incremental form, i.e. sample sets $X1, X2, ...,$ where all examples of a set $X^y = \{x_1^y, ..., x_{n_y}^y\}$ are of class $y \in N$.

Classification. For classification, iCaRL relies on sets, $P1, ..., Pt$, of exemplar images that it selects dynamically out of the data stream. There is one such exemplar set for each observed class so far, and iCaRL ensures that the total number of exemplar images never exceeds a fixed parameter K. `Algorithm 1` describes the mean-of-exemplars classifier that is used to classify images into the set of classes observed so far, see Section 2.2 for a detailed explanation.

Algorithm 1 iCaRL CLASSIFY

input x // image to be classified
require $\mathcal{P} = (P_1, \ldots, P_t)$ // class exemplar sets
require $\varphi : \mathcal{X} \to \mathbb{R}^d$ // feature map
  for $y = 1, \ldots, t$ **do**
    $\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$ // mean-of-exemplars
  end for
  $y^* \leftarrow \underset{y=1,\ldots,t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\|$ // nearest prototype
output class label y^*

Training. For training, iCaRL processes batches of classes at a time using an incremental learning strategy. Every time data for new classes is available iCaRL calls an update routine (`Algorithm 2`, see Sections 2.3 and 2.4). The routine adjusts iCaRL's internal knowledge (the network parameters and exemplars) / based on the additional information available in the new observations (the current training data). This is also how iCaRL learns about the existence of new classes.

Algorithm 2 iCaRL INCREMENTALTRAIN

input $X^s, \ldots, X^t$ // training examples in per-class sets
input K // memory size
require Θ // current model parameters
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // current exemplar sets
  $\Theta \leftarrow \text{UPDATEREPRESENTATION}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$
  $m \leftarrow K/t$ // number of exemplars per class
  for $y = 1, \ldots, s-1$ **do**
    $P_y \leftarrow \text{REDUCEEXEMPLARSET}(P_y, m)$
  end for
  for $y = s, \ldots, t$ **do**
    $P_y \leftarrow \text{CONSTRUCTEXEMPLARSET}(X_y, m, \Theta)$
  end for
  $\mathcal{P} \leftarrow (P_1, \ldots, P_t)$ // new exemplar sets

Architecture. Under the hood, iCaRL makes use of a convolutional neural network (CNN) [19]1. We interpret the network as a trainable feature extractor, $\phi : X \to R^d$, followed by a single classification layer with as many sigmoid output nodes as classes observed so far [3]. All feature vectors are L2-normalized, and the results of any operation on feature vectors, e.g. averages, are also re-normalized, which we do not write explicitly to avoid a cluttered notation.

We denote the parameters of the network by Θ, split into a fixed number of parameters for the feature extraction part and a variable number of weight vectors. We denote the latter by $w_1, ..., w_t \in R^d$, where here and in the following sections we use the convention that t denotes the number of classes that have been observed so far. The resulting network outputs are, for any class $y \in \{1, ..., t\}$,

a single classification layer (sigmoid) * # pre-class sets → CNN: a trainable feature extractor (L2-normalized)

$g_y(x) = \frac{1}{1+exp(-a_y(x))}$ with $a_y(x) = w_y^\top \phi(x)$ → (1)

Note that even though one can interpret these outputs as probabilities, iCaRL uses the network only for representation learning, not for the actual classification step.

Resource usage. Due to its incremental nature, iCaRL does not need a priori(선험적인) information about which and how many classes will occur, and it can –in theory– run for an unlimited amount of time. At any time during its runtime its memory requirement will be the size of the feature extraction parameters, the storage of K exemplar images and as many weight vectors as classes that have been observed. This knowledge allows us to assign resources depending on the application scenario. If an upper bound on the number of classes is known, one can simply pre-allocate space for as many weight vectors as required and use all remaining available memory to store exemplars. Without an upper limit, one would actually grow the number of weight vectors over time, and decrease the size of the exemplar set accordingly. Clearly, at least one exemplar image and weight vector is required for each classes to be learned, so ultimately, only a finite number of classes can be learned, unless one allows for the possibility to add more resources over the runtime of the algorithm. Note that iCaRL can handle an increase of resources on-the-fly(즉석에서) without retraining: it will simply not discard any exemplars unless it is forced to do so by memory limitations.

2.2. Nearest-Mean-of-Exemplars Classification

the algorithmic details and explains the design choices.

iCaRL uses a nearest-mean-of-exemplars classification strategy. To predict a label, y^*, for a new image, x, it computes a prototype vector for each class observed so far, $\mu_1, ..., \mu_t$, where $\mu_y = \frac{1}{|P_y|} \sum_{p \in P_y} \phi(p)$ is the average feature vector of all exemplars for a class y. It also computes the feature vector of the image that should be classified and assigns the class label with most similar prototype:

$y^* = \arg\min_{y=1,...,t} \|\phi(x) - \mu\|$ → (2)

Background. The nearest-mean-of-exemplars classification rule overcomes two major problems of the incremental learning setting, as can be seen by contrasting it against other possibilities for multi-class classification.
The usual classification rule for a neural network would be $y^* = argmax_{y=1,...,t} g_y(x)$, where $g_y(x)$ is the network output as defined in (1) or alternatively with a softmax output layer. Because $argmax_{y=1,...,t} g_y(x) = argmax_y w_y^\top \phi(x)$, the network's prediction rule is equivalent to the use of a linear classifier with non-linear feature map ϕ and weight vectors $w_1, ..., w_t$. In the class-incremental setting, it is problematic that the weight vectors w_y are decoupled(탈동조화) from the feature extraction routine ϕ: whenever ϕ changes, all $w_1, ..., w_t$ must be updated as well. Otherwise, the network outputs will change uncontrollably, which is observable as catastrophic forgetting. In contrast, the nearest- mean-of-exemplars rule (2) does not have decoupled weight vectors. The class-prototypes automatically change whenever the feature representation changes, making the classifier robust against changes of the feature representation.

The choice of the average vector as prototype is inspired by the nearest-class-mean classifier [24] for incremental learning with a fixed feature representation. In the class- incremental setting, we cannot make use of the true class mean, since all training data would have to be stored in order to recompute this quantity after a representation change. Instead, we use the average over a flexible number of exemplars that are chosen in a way to provide a good approximation to the class mean.
Note that, because we work with normalized feature vectors, Equation (2) can be written equivalently as $y^* = \arg\max_y \mu_y^\top \phi(x)$. Therefore, we can also interpret the
classification step as classification with a weight vector, but one that is not decoupled from the data representation but changes consistently with it.

2.3. Representation Learning

Algorithm 3 iCaRL UPDATEREPRESENTATION

input $X^s, \ldots, X^t$ // training images of classes $s, \ldots, t$
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // exemplar sets
require Θ // current model parameters
// form combined training set:

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

// store network outputs with pre-update parameters:
for $y = 1, \ldots, s-1$ **do**
$q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}$
end for
run network training (*e.g.* BackProp) with loss function

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

that consists of *classification* and *distillation* terms.

Whenever iCaRL obtains data, $X_s, ..., X_t$, for new classes, $s, ..., t$, it updates its feature extraction routine and the exemplar set. `Algorithm 3` lists the steps for incrementally improving the feature representation. First, iCaRL constructs an augmented training set consisting of the currently available training examples together with the stored exemplars. Next, the current network is evaluated for each example and the resulting network outputs for all previous classes are stored (not for the new classes, since the network has not been trained for these, yet). Finally, the network parameters are updated by minimizing a loss function that for each new image encourages the network to output the correct class indicator for new classes (classification loss), and for old classes, to reproduce the scores stored in the previous step (distillation loss).

Background. The representation learning step resembles ordinary network finetuning: starting from previously learned network weights it minimizes a loss function over a training set. As a consequence, standard end-to-end learning methods can be used, such as backpropagation with mini-batches, but also recent improvements, such as dropout [38], adaptive stepsize selection [14] or batch normalization [13], as well as potential future improvements.
There are two modifications to plain finetuning that aim at preventing or at least mitigating catastrophic forgetting. First, the training set is augmented. It consists not only of the new training examples but also of the stored exemplars. By this it is ensured that at least some information about the data distribution of all previous

classes enters the training process. Note that for this step it is important that the exemplars are stored as images, not in a feature representation that would become outdated over time. Second, the loss function is augmented as well. Besides the standard classification loss, which encourages improvements of the feature representation that allow classifying the newly observed classes well, it also contains the distillation loss, which ensures that the discriminative information learned previously is not lost during the new learning step.

2.4. Exemplar Management

Whenever iCaRL encounters new classes it adjusts its exemplar set. All classes are treated equally in this, i.e., when t classes have been observed so far and K is the total number of exemplars that can be stored, iCaRL will use $m = K/t$ exemplars (up to rounding) for each class. By this it is ensured that the available memory budget of K exemplars is always used to full extent, but never exceeded.

Algorithm 4 iCaRL CONSTRUCTEXEMPLARSET

input image set $X = \{x_1, \dots, x_n\}$ of class y
input m target number of exemplars
require current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$
  $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ // current class mean
  for $k = 1, \dots, m$ **do**
    $p_k \leftarrow \operatorname{argmin}_{x \in X} \left\| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right\|$
  end for
  $P \leftarrow (p_1, \dots, p_m)$
output exemplar set P

Two routines are responsible for exemplar management: one **to select exemplars for new classes** and one **to reduce the sizes of the exemplar sets of previous classes**. `Algorithm 4` describes the exemplar selection step. Exemplars $p_1, ..., p_m$ are selected and stored iteratively until the target number, m, is met. In each step of the iteration, one more example of the current training set is added to the exemplar set, namely the one that causes the average feature vector over all exemplars to

best approximate the average feature vector over all training examples. Thus, the exemplar ”set” is really a prioritized list. The order of its elements matters, with exemplars earlier in the list being more important. The procedure for removing exemplars is specified in `Algorithm 5`. It is particularly simple: to reduce the number of exemplars from any m' to m, one discards the exemplars$p_{m+1}, ..., p_{m'}$, keeping only the examples $p_1, ..., p_m$.

Background. The exemplar management routines are designed with **two objectives** in mind: the initial exemplar set should approximate the class mean vector well, and it should be possible to remove exemplars at any time during the algorithm’s runtime without violating this property.
The latter property is challenging because the actual class mean vector is not available to the algorithm anymore when the removal procedure is called. Therefore, we adopt a data-independent removal strategy, removing elements in fixed order starting at the end, and we make it the responsibility of the exemplar set construction routine to make sure that the desired approximation properties are fulfilled even after the removal procedure is called at later times. The prioritized construction is the logical consequence of this condition: it ensures that the average feature vector over any subset of exemplars, starting at the first one, is a good approximation of the mean vector. The same prioritized construction is used in herding [39] to create a representative set of samples from a distribution. There it was also shown that the iterative selection requires fewer samples to achieve a high approximation quality than, e.g., random subsampling. In contrast, other potential methods for exemplar selection, such as [7, 26], were designed with other objectives and are not guaranteed to provide a good approximation quality for any number of prototypes. Overall, iCaRL’s steps for **exemplar selection and reduction** fit exactly to the incremental learning setting: the selection step is required for each class only once, when it is first observed and its training data is available. At later times, only the reduction step is called, which does not need access to any earlier training data.

5. Conclusion

We introduced iCaRL, a strategy for class-incremental learning that learns classifiers and a feature representation simultaneously. iCaRL’s three main components are: 1) a nearest-mean-of-exemplars classifier that is robust against changes in the data representation while needing to store only a small number of exemplars per class, 2) a herding- based step for prioritized exemplar selection, and 3) a representation learning step that uses the exemplars in combination with distillation to avoid catastrophic forgetting. Experiments on CIFAR-100 and ImageNet ILSVRC 2012

data show that iCaRL is able to learn incrementally over a long period of time where other methods fail quickly.

The main reason for iCaRL's strong classification results are its use of exemplar images. While it is intuitive that be- ing able to rely on stored exemplars in addition to the net- work parameters could be beneficial, we nevertheless find it an important observation how pronounced this effect is in the class-incremental setting. We therefore hypothesize that also other architectures should be able to benefit from using a combination of network parameters and exemplars, especially given the fact that many thousands of images can be stored (in compressed form) with memory requirements comparable to the sizes of current deep networks.

Despite the promising results, class-incremental classification is far from solved. In particular, iCaRL's performance is still lower than what systems achieve when trained in a batch setting, i.e. with all training examples of all classes available at the same time. In future work we plan to analyze the reasons for this in more detail with the goal of closing the remaining performance gap. We also plan to study related scenarios in which the classifier cannot store any of the training data in raw form, e.g. for privacy reasons.