University of Milano-Bicocca

Department of Informatics, Systems and Communication

Master's Degree in Computer Science

LARGE SCALE INCREMENTAL LEARNING LOGO DETECTION AND RECOGNITION

Supervisor: Prof. Simone Bianco

Co-Supervisor: Dr. Marco Buzzelli

Master's Degree Thesis by:

Gianluca Giudice

830694

Academic Year 2021-2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Logo detection and recognition

Logo detection and recognition is a Computer Vision (CV) task that can be formulated as a two-stage process: locating the logos in the image (i.e. finding the coordinates of logo instances inside the image) and identifying the detected logos (i.e. producing in output the brand name associated with that logo).

The first studies in this field date back to 1993 [17], yet this task is becoming increasingly important in a variety of applications. Some examples of studies relative to this problem aimed to: monitor the brand visibility on social media [22]; protect the intellectual property on e-commerce platforms, known as intellectual property protection (IPP) [32]; develop online video advertising systems [13]; and help the development of autonomous checkout systems in retail environments [53].

There are several challenges in logo detection and recognition, starting from the fact that a symbol composed by text and images can be considered a logo, but there is no formal definition of what a logo is. In fact, logos can be created from text using a lot of different typographic styles, any particular graphic consisting of many colors, or even a combination of the two. There is a huge variety of logos and this problem has both high intra-class and inter-class variations, since the same brand can have very different logos (e.g. only a stylized text version and a graphic version) and logos which belong to different brands might look very similar.

Since many new brands are constantly being created and each brand has its own logo, it is necessary to develop systems that keep up with the creation of new logos. A method for logo detection and recognition should take into account this particular aspect of the problem and should properly recognize each new logo.

For this reason, there is the need to develop a system which is able to adapt to these

changes where standard closed-set classification techniques would fail. One possible approach could be to train a new classifier each time a new logo is created. However, this method is very inefficient and unfeasible for large scale datasets of logos. Moreover, retraining a model in such a way would require to store a large quantity of examples, since both the data from the previous logos and the new ones would be needed.

Open set logo recognition allows models to detect logos that are not available during the training phase. In this way, it is possible to overcome the problems discussed above. As a consequence, systems in which logo detection and recognition is performed in an open environment have been proposed in the literature [19, 40].

1.2 Class incremental learning

Incremental learning aims to develop artificially intelligent systems that can continuously learn to address new tasks from new data while preserving knowledge learned from previously learned tasks [52]. This way of learning is inspired from natural systems which are intrinsically incremental [84].

The main problem in incremental learning is known as the stability-plasticity dilemma and it holds for both artificial and biological neural systems. The idea is that a learning system requires plasticity for the integration of new knowledge, but also stability to avoid forgetting previous knowledge [55]. Excessive plasticity would lead to forget all the previous knowledge (referred to as catastrophic forgetting [25]), whereas too much stability would prevent the ability to learn novel concepts. The main point is to achieve a trade-off between stability and plasticity.

Many real-world applications require incremental learning capabilities: intelligent robots during their lifetime [78], face recognition systems [41] and autonomous driving [58]. Logo detection and recognition is another example where Class Incremental Learning (CIL) is well suited to address the problem, considering the constant creation of new logos discussed in the previous section. Using this technique it is possible to create a system which detects and recognizes an initial set of logos and, when necessary, enriches the acquired knowledge.

1.3 Proposed approach

Figure 1.1: Simplified system pipeline.

The goal of this thesis is to develop a system that can detect and recognize a large number of different logos present in an image. An important factor that is considered in this work is the incremental learning aspect. This means that the system is trained on an initial set of logos and, when necessary, it is possible to enrich its knowledge to recognize new logos. This is done by taking advantage of state of the art incremental learning techniques.

The proposed system is conceptually simple, it consists of two deep learning models and follows a pipeline composed of two main stages, shown in Figure 1.1:

1. **Object Proposal** for logo detection, performed by the class-agnostic logo detector.
2. **Classification** for logo recognition, performed by the CIL classifier.

In the first stage, the model is a class-agnostic logo detector based on YOLOv5 [1]. In this context, class-agnostic means that the logo detector is only responsible for identifying a generic logo, while the actual classification is not performed by the detector, but is delegated to the second model that produces the logo class as output. Given an input image, the purpose of this first stage is to produce as many cropped portions of it as there are logos in the image. These cropped regions are called Regions of Interest (RoIs) and correspond to what the model considers to be logos.

The second stage exploits the RoIs generated by the detector and proceeds with the actual recognition of logos, this step is purely a classification task. Here, the problem is to recognize new logos which are created over time. To do so, the model is initially trained on all the logos that are available up to that time. Then, the knowledge of the model is updated to include a new set of logos. These updates of the knowledge are called incremental learning steps, and a crucial aspect of integrating new knowledge is the ability to do so in such way that the model does not forget its previous knowledge. As a consequence, the developed model uses incremental learning techniques that allow it to recognize new logos (classes).

Unlike the second stage, we can say that there is no need to develop a detector with incremental learning techniques. This is justified by the fact that the idea of what a generic logo is can be learned and well-approximated using only an initial set of logos. The subsequent creation of new logos will not disrupt the general idea of a logo, therefore, there is no need to change the knowledge initially learned.

1.4 Main contributions

This work reformulates the problem of logo recognition in terms of class incremental learning. By doing so, new knowledge can be integrated in a model without the need to retrain it from scratch.

This thesis defines a two-stage logo detector that consists of a first stage concerning the localization of logos in the image and a second stage regarding the classification of logos. To achieve CIL, an algorithm proposed in the literature is here modified using a different optimizer for training the model. In addition, some regularization techniques are used to avoid model overfitting, such as a dropout layer introduced in the neural network architecture and data augmentation to create synthetic samples for model training.

LogoDet-3K [83], a large-scale public dataset for logo recognition is analyzed in detail, highlighting some limitations related to the number of samples for each class and pointing out some flaws related to the labels of some classes.

Particular attention is given to the size of the models in terms of the number of parameters. A pruning technique with learnable channel-level masks proposed in the literature is implemented with the aim of reducing the parameters of the CIL classifier. As an alternative to this method, the Knowledge Distillation (KD) technique is used to train a significantly smaller neural network under the supervision of a teacher model.

Several experiments are conducted to compare the proposed approaches. To this end, three different baselines are defined to assess the drop in performance when using a CIL approach as opposed to a standard one. Then, the system is tested on a large-scale dataset and compared with state-of-the-art methods, highlighting the strengths and weaknesses of the proposed system and focusing on aspects that could be improved.

The work described in this thesis has resulted in a paper submitted to the 12th IEEE International Conference on Consumer Technology (ICCE-Berlin 2022).

This thesis is structured as follows: starting from this introduction, the second chapter describes the state of the art regarding object detection, logo recognition and class incremental learning algorithms; the third chapter describes the dataset used to tackle this problem; the fourth chapter is a detailed description of the developed system and the techniques adopted; the fifth chapter describes the experiments and results obtained using the proposed approach; the last chapter is relative to the conclusions of this work and some considerations about future works.

Chapter 2

State of the art

2.1 Object detection

Object detection is one of the most fundamental and challenging problems in computer vision [97]. This task can be defined as follows: given an image, determine whether or not there are instances of a predefined set of objects, usually referred to as classes, and if present return the location of each instance [45]. The spatial location of an object in an image can be represented using bounding boxes.

Object detection was initially addressed using handcrafted features and shallow trainable architectures. With the rapid development in deep learning, more powerful techniques are used to address the issues existing in traditional solutions [93].

As described in [93], the frameworks of object detection methods can be mainly categorized into two types:

1. **Two-stage Object Detection**: generates region proposals at first and then classifies each proposal into different object categories.

2. **One-stage Object Detection**: adopts a unified framework to achieve final results (categories and locations) directly.

2.1.1 YOLO

YOLO (You Only Look Once) [65] is a model for object detection composed of a single Neural Network (NN) which treats object detection as a regression problem: given an image as input it produces bounding box coordinates and associated class probabilities. Since the predictions are performed directly on the input image without requiring complex pipelines, YOLO is very efficient and can lead to real-time object detection.

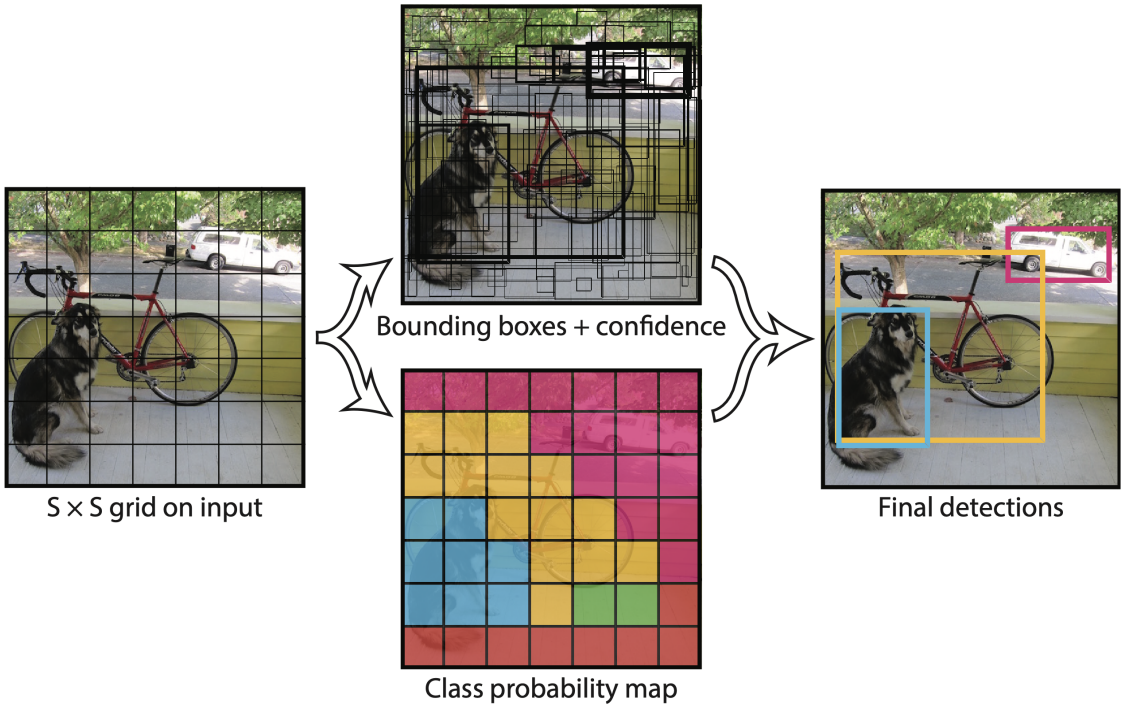


Figure 2.1: Pipeline of YOLO object detector: it divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor [65].

In YOLO, the input image is divided into an $S \times S$ grid and the cell in which the center of the object falls is responsible for the detection of that object. A grid cell can predict more than one bounding box, where each prediction consists of an array composed by 5 elements: center of the bounding box identified by the coordinates x and y, dimensions of the box w and h, and the confidence score of that bounding box representing an object. At the same time, regardless of the number of boxes in each cell, C conditional probabilities $\Pr(Class_i|Object)$ are computed in each grid cell. The final prediction will be encoded as an $S \times S \times (B * 5 + C)$ tensor. The model pipeline is shown in Figure 2.1.

In order to predict and localize many different objects in an image, YOLO uses anchor boxes, which are a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes. Then, the predefined anchor boxes are used as a starting point during detection.

The model is trained via the optimization of the following loss function:

$$
\begin{aligned}
\mathcal{L} &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \; + \; \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.1}
$$

where,

$$
\mathbb{1}_{ij} = \begin{cases} 1, & \text{if the } j\text{-th bbox in cell } i \text{ is responsible for that prediction} \\ 0, & \text{otherwise} \end{cases}
$$

$$
\mathbb{1}_{i} = \begin{cases} 1, & \text{if there is an object in cell } i \\ 0, & \text{otherwise} \end{cases}
$$

The loss function in Equation 2.1 considers both detection and classification. A bounding box can be defined by its center in addition to the height and width. The first row of the loss function is responsible for minimizing the difference between the predicted center and the ground truth, while the second row minimizes the difference between width and height. The third row penalizes the neural network if it predicts an object whereas it is not present and vice versa. The last row of the loss function is the mean squared error between the real class and the predicted one, hence it is responsible to match the real class.

Over the years, several versions of YOLO have come out, starting from the first up to fifth version [65, 63, 64, 8, 1]. YOLOv5 represents the state of the art for object detection, and compared with the most recent models it is among the best performing ones [87].

2.2 Logo recognition

A general pipeline for logo detection and recognition consists in logo region proposal followed by a classifier specifically trained for logo classification, as proposed by Bianco et al. in [7] or by Fehérvári et al. in [19].

Another approach presented by Wang et al. [83] involves a model based on YOLOv3 [64] used to produce both bounding boxes and classification for each detected logo. The proposed model is called Logo-YOLO, which is essentially the same version of YOLOv3 with some changes to the loss function (described in Equation 2.1) and the re-computation of the anchors sizes. The modified loss function utilizes the Focal Loss [43] to solve the

problem of logos being small objects compared to the background, and the CIoU loss [94] to obtain more accurate and faster regression of the bounding boxes.

The issue with these approaches is the closed-world assumption which does not apply in the case of logo recognition, as discussed in section 1.1. This is the purpose behind works such as [19]. The authors of the paper propose a method based on Distance Metric Learning (DML) using deep learning techniques called SoftTriple Loss [59]. This work aims to achieve logo recognition via metric learning, where a model learns the similarity and differences of objects in a latent space, thus being able to deal with a large number of previously unseen classes exploiting the distances of objects in the latent space.

Another work based on DML has been presented by Li et al. [40] and can be considered as an extension of [19]. In this work the authors enrich the latent space learned by DML with text features contained in the logos. The authors highlight how a large number of logos have remarkable amount of text or stylized letters, for this reason, in addition to visual features, they consider text features as relevant information for logo classification.

2.3 Class incremental learning

Traditional supervised learning systems are trained in closed-world for a fixed number of classes, that requires all the training data to be available before training. Class Incremental Learning (CIL) aims to design algorithms that can learn new concepts in a sequential way and eventually perform well on all observed classes [86].

To extend a trained model on new classes, a large amount of labeled data for both new and old classes is necessary for network finetuning. Otherwise, if the dataset of old classes is no longer available, finetuning a deployed model with new classes can lead to the catastrophic forgetting problem [71, 90, 54]. Catastrophic forgetting means that a model degrades performance on old classes when retrained on new ones.

2.3.1 Problem setup

During CIL, a stream of class groups $\{\mathcal{Y}_t\}$ and their corresponding training data $\{\mathcal{D}_t\}$ are observed by the model. At step t of incremental learning, the incoming dataset $\{\mathcal{D}_t\}$ is of the form $(\mathbf{x_i^t}, y_i^t)$ where $\mathbf{x_i^t}$ is the input image and $y_i^t \in \mathcal{Y}_t$ is the label set $\mathcal{Y}_t$. The label space of the model consists of all categories seen up to that point $\tilde{\mathcal{Y}}_t = \cup_{i=1}^{t} \mathcal{Y}_i$ and a good model should predict well on all classes $\tilde{\mathcal{Y}}_t$. As described in subsection 2.3.2, some CIL algorithms save a part of data at timestamp t as memory $\mathcal{M}_t$ for future training. The CIL setup is shown in Figure 2.2.

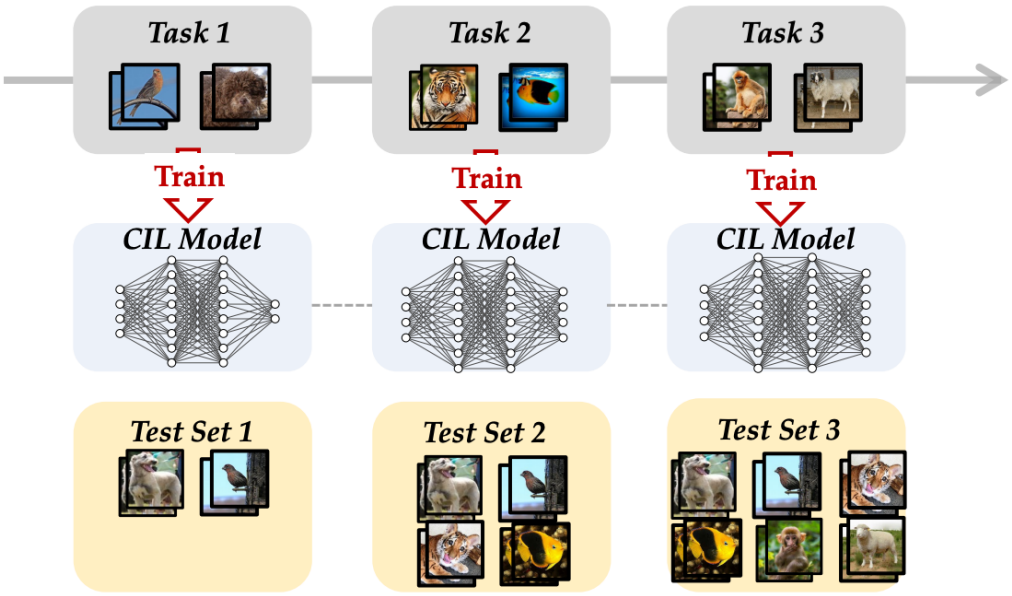


Figure 2.2: General CIL setup. In an incremental learning setup, classes arrive sequentially at each task, so we only have access to the classes of that task and the previously seen classes. To classify new classes using the same model, the classifier is built incrementally. After the learning phase of each task, the model is evaluated among all seen classes. The image shows an example of three incremental learning steps. Image from [96].

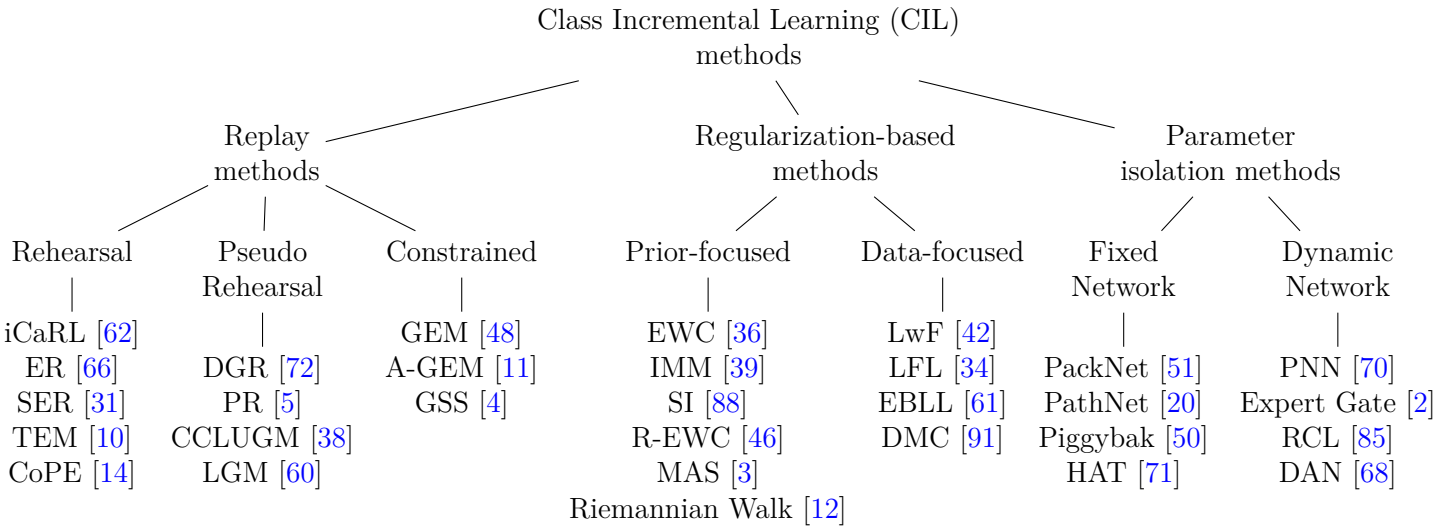


Figure 2.3: CIL algorithms taxonomy presented in [15].

2.3.2 Methods

The problem of CIL has been addressed using different methods, and they can be divided into three main categories and their sub-categories. The taxonomy and the list of algorithms, summarized in the Figure 2.3, is based on these works [47, 15]:

1. **Replay methods**: these works store samples of old classes which are replayed while learning a new task, by doing so it is possible to alleviate forgetting. The samples are either reused as model inputs for rehearsal, or to constrain the optimization of the loss function on new tasks.

   (a) *Rehearsal* [62, 66, 31, 10, 14]: the model is retrained on a limited subset of stored samples.

   (b) *Pseudo Rehearsal* [72, 5, 38, 60]: approximates previous tasks samples using either random input vector or more advanced techniques like generative models. However, the latter one has the drawback of adding complexity to the system pipeline.

   (c) *Constrained optimization* [48, 11, 4]: updates the model weights for the new task so as not to interfere with the weights of previous tasks.

2. **Regularization-based methods**: these works do not store raw data and reduce the memory requirements. An extra regularization term is introduced in the loss function, in this way it is possible to learn new classes while maintaining the previous knowledge.

   (a) *Prior-focused methods* [36, 39, 88, 46, 3, 12]: this approach tries to estimate the importance of the neural network parameters to avoid forgetting as much as possible. Then, when training on new tasks, the optimization process penalizes changes to important weights.

   (b) *Data-focused methods* [42, 34, 91, 61]: this approach uses knowledge distillation [28] by treating the old model trained on the previous task as the *teacher*, and uses the new data to train the *student* model.

3. **Parameter isolation methods**: this class of methods use different model parameters for each task.

   (a) *Fixed Network* [51, 50, 71, 20]: the model architecture is kept fixed and the knowledge is updated involving masks for the model parameters.

   (b) *Dynamic Architectures* [70, 85, 2, 68]: the model architecture dynamically changes after each incremental task, and the old parameters are frozen to prevent forgetting. Another approach is to dedicate a model to each incremental task.

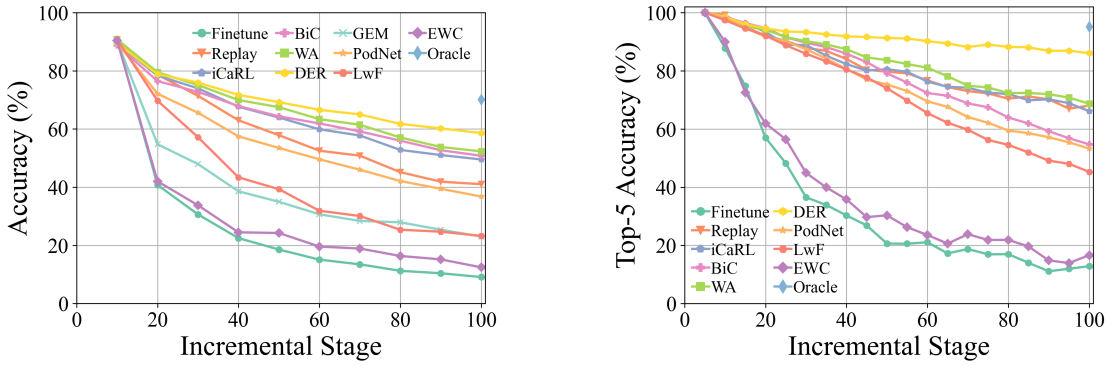
Since there are several algorithms for CIL, the work presented by Zhou et al. in [96] aims to compare a subset of these algorithms considering the same experimental setup. To this purpose, they publish a repository[1] on GitHub where they introduce *PyCIL: A Python Toolbox for Class-Incremental Learning*, which is an implementation of all the following algorithms:

- **Finetune**: the baseline method which updates parameters on new tasks and suffers from severe catastrophic forgetting.
- **Replay**: the baseline method which updates parameters on new task with instances from the new dataset and exemplar set.
- **EWC** [36]: uses Fisher Information Matrix to weight the importance of each parameter and regularizes important parameters to overcome forgetting.
- **LwF** [42]: uses knowledge distillation [28] to align the output probability between old and new model.
- **iCaRL** [62]: uses knowledge distillation [28] to align the output probability between old and new model. It also introduces exemplar set for rehearsal and uses nearest center mean classifier.
- **GEM** [48]: uses exemplars as the regularization of gradient updating.
- **BiC** [84]: trains an extra adaptation layer based on iCaRL, which adjusts the logits on new classes.
- **WA** [92]: normalizes the classifier weight after each learning session based on iCaRL.
- **PODNet** [18]: introduces a novel distillation loss (Pooled Outputs Distillation) constraining the whole convolutional network.
- **DER** [86]: utilizes a dynamically expandable representation for more effective incremental concept modeling.
- **Coil** [95]: builds bi-directional knowledge transfer in the class incremental learning process with optimal transport [82].

The CIL algorithms are tested on two different benchmark datasets, i.e. CIFAR100 [37] and ImageNet100 [69]. These datasets are composed of 100 classes and the experiments performed in PyCIL [96] adopt this CIL setup: 10 classes for the initial task followed by 9

[1]PyCIL GitHub repository: https://github.com/G-U-N/PyCIL

incremental learning steps consisting of 10 new classes. Then, the top-1 accuracy is reported for all the incremental learning steps. The results are shown in Figure 2.4 and the top-1 accuracy at the final step is reported in Table 2.1.



(a) 10 steps on CIFAR100

(b) 10 steps on ImageNet100

Figure 2.4: Performance comparison of the CIL algorithms during 10 incremental learning steps on CIFAR100 and ImageNet100 datasets [96].

Method	Reproduced in PyCIL	Reported
Finetune	26.25	26.25
Replay	59.31	-
GEM	40.18	-
LwF	43.56	-
iCaRL	64.42	64.10
EWC	29.73	-
WA	67.09	64.5
PODNet	55.22	-
BiC	65.08	-
Coil	65.48	65.48
DER	**69.74**	69.41

Table 2.1: Top-1 accuracy of the CIL algorithms at the 10-th incremental learning step using the CIFAR100 dataset [96].

From the final results reported in PyCIL in Table 2.1 it is clear that the best performing algorithm on the datasets considered is DER [86]. Moreover, even if the authors of PyCIL re-implement the DER algorithm, they point out how the performance obtained from the implementation of PyCIL is very similar to that reported in the original DER paper (69.74 of top-1 accuracy in PyCIL vs 69.41 reported by the DER paper).

2.3.3 DER: an algorithm for class incremental learning

DER (Dynamically Expandable Representation) is an algorithm for CIL that utilizes a dynamically expandable representation and uses a limited memory for the old classes introduced in each incremental step. The system proposed in this thesis takes advantage of this algorithm for what concerns the incremental learning aspect of the classifier (see section 4.2).

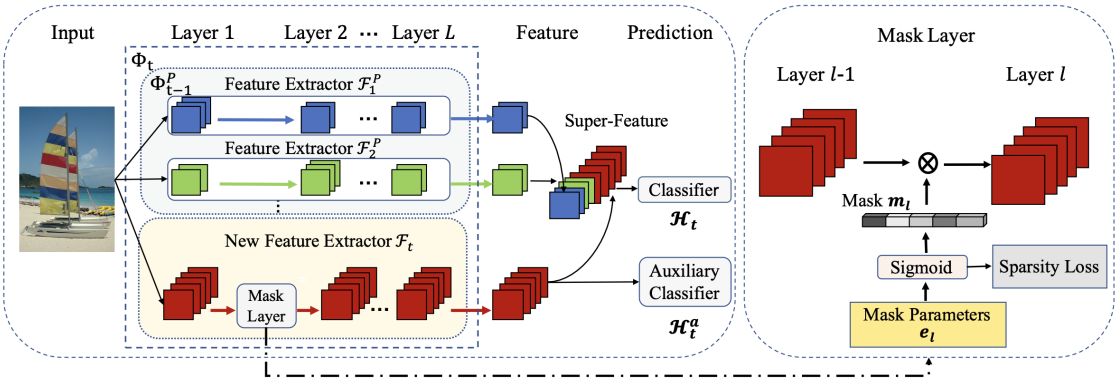


Figure 2.5: Dynamically Expandable Representation Learning method: at incremental learning step t, the model is composed of the super-feature extractor Φ_t and the classifier $\mathcal{H}_t$. Φ_t is created using the previous super-feature extractor Φ_{t-1} and the newly created feature extractor $\mathcal{F}_t$. An auxiliary classifier $\mathcal{H}_t^a$ is introduced to regularize the model. Moreover, a channel-level mask-based pruning strategy is used to maintain a compact representation of the model [86].

Each incremental learning iteration of the DER algorithm described in [86] can be divided into three stages (see Figure 2.5):

1. **Representation Learning Stage**: the architecture of the Convolutional Neural Network (CNN).
2. **Masking and Pruning**: the CNN introduced in the last incremental learning stage is pruned using a channel-level masked-based method.
3. **Classifier Learning Stage**: retraining of the classifier with currently available data $\tilde{\mathcal{D}}_t = \mathcal{D}_t \cup \mathcal{M}_t$ at step t.

Expandable Representation Learning

At step t of incremental learning, the algorithm defines a super-feature extractor Φ_t and the classifier $\mathcal{H}_t$. The algorithm expands the neural network architecture at each step by considering the previous super-feature extractor Φ_{t-1} and a newly created feature extractor

$\mathcal{F}_t$. The feature extractor $\mathcal{F}_t$ consists of a CNN, and the classifier $\mathcal{H}_t$ is the Fully Connected (FC) layer after the super-feature extractor Φ_t.

Given an image $\mathbf{x} \in \tilde{\mathcal{D}}_t$, the feature $\mathbf{u}$ extracted by Φ_t is the result of the following concatenation:

$$\mathbf{u} = \Phi_t(\mathbf{x}) = [\Phi_{t-1}(\mathbf{x}), \mathcal{F}_t(\mathbf{x})] \tag{2.2}$$

The feature $\mathbf{u}$ is then fed into the classifier $\mathcal{H}_t$, which is defined in such a way that the input and output dimensions for step t match. Then, the prediction is produced as follows:

$$p_{\mathcal{H}_t}(\mathbf{y}|\mathbf{x}) = \text{Softmax}(\mathcal{H}_t(\mathbf{u})) \tag{2.3}$$

Finally, the actual predicted class is given by:

$$\hat{y} = \arg\max \mathcal{H}_t(\mathbf{y}|\mathbf{x}), \quad \hat{y} \in \tilde{\mathcal{Y}}_t \tag{2.4}$$

The stability-plasticity tradeoff is taken into account considering these two aspects:

- To **retain old knowledge**, the parameters of $\mathcal{H}_t$ associated with the old features are inherited from $\mathcal{H}_{t-1}$, and its newly added parameters are randomly initialized.
- To **reduce catastrophic forgetting**, the parameters $\theta_{\Phi_{t-1}}$ and the statistics of Batch Normalization [30] of the old super-feature extractor Φ_{t-1} are frozen at step t, in this way it is possible to maintain the previously learned knowledge.

To force the network to learn new and discriminative features at each incremental step, the authors of the algorithm introduce an auxiliary classifier $\mathcal{H}_t^a$. The label space of $\mathcal{H}_t^a$ is $|\mathcal{Y}_t| + 1$, corresponding to the the set $\mathcal{Y}_t$ of all new categories, and treating all the old concepts as a single category. This is done to encourage the network to learn features that discriminate old and new concepts. The classifier $\mathcal{H}_t^a$ predicts the probability:

$$p_{\mathcal{H}_t^a}(\mathbf{y}|\mathbf{x}) = \text{Softmax}(\mathcal{H}_t^a(\mathcal{F}(\mathbf{x}))) \tag{2.5}$$

Masking and pruning

To maintain a compact representation of the model and remove redundancy, the super-feature extractor is pruned after the training of each task t, the method proposed by the authors is adapted from [71]. This is done using a differentiable channel-level mask-based method to prune filters of the extractor $\mathcal{F}_t$, in which the masks are learned jointly with the representation. The mask is then binarized after the learning procedure, and the feature extractor $\mathcal{F}_t$ is pruned using the binary mask, producing as output the pruned network $\mathcal{F}_t^P$.

Given a new feature extractor $\mathcal{F}_t$, the input feature map of the convolutional layer l for an image $\mathbf{x}$ is denoted as $\mathbf{f}_l$. The channel mask $\mathbf{m}_l \in \mathbb{R}^{c_l}$ controls the size of the layer

l, where $m_l^i \in [0,1]$ and c_l is the number of channels of layer l. The mask m_l is used to modulate f_l as follows:

$$\mathbf{f}_l{}' = \mathbf{f}_l \odot \mathbf{m}_l \tag{2.6}$$

where $\mathbf{f}_l{}'$ is the masked feature map and $\odot$ means channel-level multiplication. To make the values of $\mathbf{m}_l$ fall into the interval [0,1], the mask is given by:

$$\mathbf{m}_l = \sigma(s\mathbf{e}_l) \tag{2.7}$$

where $\mathbf{e}_l$ is the learnable mask parameters, $\sigma(\cdot)$ is the sigmoid used as gating function, and s is a scaling factor to control the sharpness of the function.

Using the pruning mechanism, the super-feature $\tilde{\mathbf{u}}$ of step t can be rewritten as:

$$\tilde{\mathbf{u}} = \Phi_t^P(\mathbf{x}) = [\mathcal{F}_1^P, \mathcal{F}_2^P, ..., \phi_t(\mathbf{x})] \tag{2.8}$$

During training, $\phi_t(\mathbf{x})$ corresponds to $\mathcal{F}_t(\mathbf{x})$ with the associated masks. At inference time, thanks to the scaling factor s, the masks associated to the filters of the convolutional layers are pruned assigning a large value to s. In this way it is possible to obtain $\mathcal{F}_t^P = \phi_t(\mathbf{x})$.

The parameters $\mathbf{e}_l$ which define the masks are learned during training. During an epoch, a linear annealing schedule is applied for the scaling factor s as follows:

$$s = \frac{1}{s_{max}} + (s_{max} - \frac{1}{s_{max}})\frac{b-1}{B-1} \tag{2.9}$$

where b is the batch index, B is the number of batches in one epoch, and $s_{max} \gg 1$ is a hyperparameter.

In this way, at the beginning of the training epoch, s has a small value. Thus, given that $\mathbf{m}_l = \sigma(s\mathbf{e}_l)$, the gating mechanism leads to values close to 0.5, regardless of the $\mathbf{e}_l$ vector. Then the mask is progressively binarized as the batch index b increases within an epoch.

As described in [71], to solve the problem of the unstable gradient of $\mathbf{e}_l$ due to s, the gradient $\mathbf{g}_{\mathbf{e}_l}$ with respect to $\mathbf{e}_l$ is compensated as follows:

$$\mathbf{g}'_{\mathbf{e}_l} = \frac{\sigma(\mathbf{e}_l)[1-\sigma(\mathbf{e}_l)]}{s\sigma(s\mathbf{e}_l)[1-\sigma(s\mathbf{e}_l)]}\mathbf{g}_{\mathbf{e}_l} \tag{2.10}$$

where $\mathbf{g}'_{\mathbf{e}_l}$ is the compensated gradient.

Classifier Learning

After the representation learning stage, the classifier head is re-trained to reduce the bias in the classifier weight introduced by the imbalanced training set $\tilde{\mathcal{D}}_t$, this is due to the difference in sample size between the old classes in the memory $\mathcal{M}_t$ and the currently available data $\mathcal{D}_t$. To this purpose, the classifier weights are randomly initialized and the classifier is trained on a class-balanced sample of $\tilde{\mathcal{D}}_t$ using the cross-entropy loss with a temperature δ in the Softmax presented in [92].

Loss function

The loss function used to change the parameters of the model is defined by three components:

1. **Classifier Loss**: the first loss component is responsible for the correct classification of all the images. Hence, based on the Equation 2.3, given an image and its corresponding label $(\mathbf{x}_i, y_i) \in \tilde{\mathcal{D}}$, the loss function penalizes wrong class prediction as follows:

$$\mathcal{L}_{\mathcal{H}_t} = -\frac{1}{|\tilde{\mathcal{D}}_t|} \sum_{i=1}^{|\tilde{\mathcal{D}}_t|} log(p_{\mathcal{H}_t}(y = y_i|\mathbf{x}_i)) \tag{2.11}$$

2. **Auxiliary Loss**: the second loss component is relative to the auxiliary classifier $\mathcal{H}_t^a$. Consistently with Equation 2.5, given an image $\mathbf{x}_i$ and the associated label $y_i \in \mathcal{Y}_i \cup \{"old"\}$, the auxiliary loss component is defined as follows:

$$\mathcal{L}_{\mathcal{H}_t^a} = -\frac{1}{|\tilde{\mathcal{D}}_t|} \sum_{i=1}^{|\tilde{\mathcal{D}}_t|} log(p_{\mathcal{H}_t^a}(y = y_i|\mathbf{x}_i)) \tag{2.12}$$

3. **Sparsity Loss**: the third loss component encourages the model to maximally reduce the number of parameters with a minimal performance drop. To this reason, the sparsity loss considers the ratio of used weights in all available weights:

$$\mathcal{L}_S = \frac{\sum_{l=1}^{L} K_l \|\mathbf{m}_{l-1}\|_1 \|\mathbf{m}_l\|_1}{\sum_{l=1}^{L} K_l c_{l-1} c_l} \tag{2.13}$$

   where L is the number of layers and K_l is the kernel size of the convolutional layer l.

The final loss function is given by the weighted average of Equation 2.11, Equation 2.12 and Equation 2.13:

$$\mathcal{L}_{DER} = \mathcal{L}_{\mathcal{H}_t} + \lambda_a \mathcal{L}_{\mathcal{H}_t^a} + \lambda_s \mathcal{L}_S \tag{2.14}$$

where λ_a and λ_s are the hyper-parameter to control the effect of the auxiliary classifier and the pruning aspect.

2.4 Weight aligning

Weight aligning (WA) proposed in [92] is a method to alleviate catastrophic forgetting in Deep Neural Networks (DNNs). In their work the authors show how a model trained for a CIL task tends to classify objects into new classes, treating old classes unfairly. This is due to the weights in the FC layer of the trained model, which are heavily biased. WA aims to correct bias in the FC layer.

Biased Weights in the FC Layer

At the t-th incremental learning step, the model is given by:

$$o(\mathbf{x}) = \mathbf{W}^T \Phi(\mathbf{x}) \tag{2.15}$$

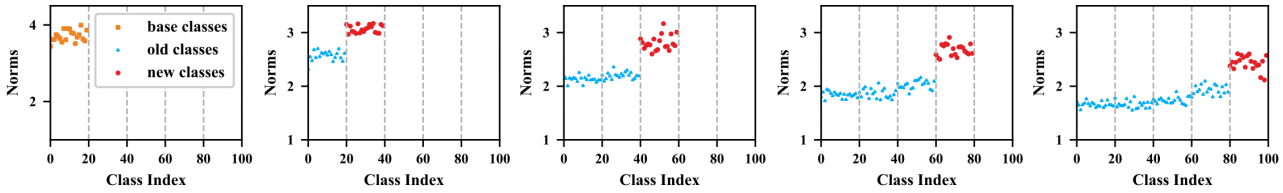
where:

- $\Phi(\mathbf{x})$ is the super-feature extractor which outputs a d-dimensional feature vector.
- $o(\mathbf{x})$ is the $(C^t_{old} + C^t)$-dimensional vector of the logits produced in output by the model. C^t_{old} and C^t represent respectively the number of old and new classes at the t-th incremental task.
- $W \in \mathbb{R}^{d \times (C^t_{old} + C^t)}$ represents the parameters of the FC layer. W can be expressed as $W = \{w_c,\ 1 \leq c \leq C^b_{old} + C^b\}$, where w_c is a d-dimensional weight vector for the c^{th} class.

The output logits for the c^{th} class are calculated as:

$$o_c(\mathbf{x}) = \mathbf{w}_c^T \Phi(\mathbf{x}) \tag{2.16}$$

The motivation behind WA is the observation that the norms of the weight vectors for new classes are larger, consequently the output logits for new classes tend to be larger as well. As a result, the biased FC layer in the model tends to predict an input image as belonging to a new class. This analysis emerges from the experiments performed in [92] and shown in Figure 2.6.



(a) $C^0 = 20, C^0_{old} = 0$ (b) $C^1 = 20, C^1_{old} = 20$ (c) $C^2 = 20, C^2_{old} = 40$ (d) $C^3 = 20, C^3_{old} = 60$ (e) $C^4 = 20, C^4_{old} = 80$

Figure 2.6: Norms of the weight vectors $\{\mathbf{w}_c\}$ after each incremental step. (a) Represents the initial step; (b), (c), (d), (e) are the 1^{st}, 2^{nd}, 3^{rd} and 4^{th} incremental step. The figure shows the difference in the norms of the weight vectors between the old and new classes [92].

Correcting biased weights in the FC Layer

The bias in the weights of the FC layer described in section 2.4 is corrected using the WA technique. The FC layer can be rewritten as $\mathbf{W} = (\mathbf{W}_{old}, \mathbf{W}_{new})$, where

$$\mathbf{W}_{old} = (\mathbf{w}_1, \mathbf{w}_2, \ ..., \mathbf{w}_{C^t_{old}}) \in \mathbb{R}^{d \times C^t_{old}},$$

$$\mathbf{W}_{new} = (\mathbf{w}_{C^t_{old}+1}, ..., \mathbf{w}_{C^t_{old}+C^t}) \in \mathbb{R}^{d \times C^t}.$$

The norms of the weight vectors of old and new classes can be denoted by:

$$\mathbf{Norm}_{old} = (\|\mathbf{w}_1\|, ..., \|\mathbf{w}_{C^t_{old}\|}),$$

$$\mathbf{Norm}_{new} = (\|\mathbf{w}_{C^t_{old}+1}\|, ..., \|\mathbf{w}_{C^t_{old}+C^t}\|).$$

Then, the weights for new classes are normalized as follows:

$$\hat{\mathbf{W}}_{new} = \gamma \cdot \mathbf{W}_{new} \tag{2.17}$$

where

$$\gamma = \frac{Mean(\mathbf{Norm}_{old})}{Mean(\mathbf{Norm}_{new})} \tag{2.18}$$

$Mean(\cdot)$ returns the mean value of the elements in the vector.

This correction on the FC layer directly affects the output logits produced by the model. In fact, the output logits of the model before WA can be expressed as:

$$\begin{aligned} \mathbf{o}(\mathbf{x}) &= (\mathbf{o}_{old}(\mathbf{x}), \mathbf{o}_{new}(\mathbf{x}))^T \\ &= (\mathbf{W}^T_{old}\Phi(\mathbf{x}), \mathbf{W}^T_{new}\Phi(\mathbf{x}))^T \end{aligned} \tag{2.19}$$

after applying WA on the FC, the corrected output logits are given by:

$$\begin{aligned} \mathbf{o}_{corrected}(\mathbf{x}) &= (\mathbf{W}^T_{old}\Phi(\mathbf{x}), \hat{\mathbf{W}}^T_{new}\Phi(\mathbf{x}))^T \\ &= (\mathbf{W}^T_{old}\Phi(\mathbf{x}), \gamma \cdot \mathbf{W}^T_{new}\Phi(\mathbf{x}))^T \\ &= (\mathbf{o}_{old}(\mathbf{x}), \gamma \cdot \mathbf{o}_{new}(\mathbf{x}))^T \end{aligned} \tag{2.20}$$

The final effect of aligning the weights is to rescale the output logits of new classes, thus leading to a fair classification between old and new classes.

Chapter 3

Datasets for logo recognition

A crucial part in the development of deep learning models is the dataset used for training. There are several datasets proposed in the literature for logo detection and recognition, yet some of them, such as BelgaLogos [56], lack in variety of the classes and have a small number of images. For this reason, some works aimed to construct larger datasets, such as WebLogo-2M [74] or Logos in the wild [79]. The issue with these types of large scale datasets is that they are in part automatically collected and labeled, leading to some noise in the target label.

A list of datasets proposed in the literature is given by Li et al. in SeeTek [40], shown in Table 3.1.

Dataset	Logos	Images	Annotation	Noisy	Public	Scalability
TopLogo-10 [75]	10	700	Object-Level	no	yes	Weak
BelgaLogos [33]	37	10,000	Object-Level	no	yes	Weak
FlickrLogos-32 [67]	32	8,240	Object-Level	no	yes	Weak
FlickrLogos-47 [67]	47	8,240	Object-Level	no	yes	Weak
Logo-NET [29]	160	73,414	Object-Level	no	yes	Weak
WebLogo-2M [74]	194	2,190,757	Image-Level	yes	yes	Medium
QMUL-OpenLogo [76]	352	27,083	Image-Level	no	yes	Medium
Logos in the wild [79]	871	11,054	Object-Level	yes	yes	Medium
BLAC [6]	2,800	6,200	Object-Level	no	no	Medium
LogoDet-3K [83]	3,000	158,652	Object-Level	no	yes	Strong
PL8K [40]	7,888	3,017,146	Object-Level	no	no	Strong

Table 3.1: Statistics and characteristics of existing logo detection datasets [40].

Some logos datasets such as LogoDet-3K and PL8K are considered more robust for what concerns the annotation of the images (noisy column in Table 3.1). In particular, each image of LogoDet-3K is manually examined and reviewed, this guarantees high-quality annotations.

Consequently, the only two large-scale and non-noisy datasets in Table 3.1 are LogoDet-3K and PL8K. However, only LogoDet-3K is publicly available, thus being the only viable choice as the target dataset to use for the experiments in this thesis (see chapter 5).

3.1 LogoDet-3K: dataset description

LogoDet-3K consists of 3,000 logo classes, 158,652 images and 194,261 logo objects. The objects correspond to the bounding boxes in each image, and since an image can contain more than one logo, the number of objects is greater than the number of images. The dataset is divided into 9 categories, each category contains several brands and for each brand there is a set of images. Table 3.2 shows the number of brands, images and objects for each category.

In this context, a brand is intended as a logo class. In fact, the same brand can have multiple versions of logos, for example, a symbolic logo and a textual logo. These types of logos are treated as different classes in the LogoDet-3K dataset, a suffix such as '*brand*-1' and '*brand*-2' is used to deal with these cases, as shown in Figure 3.1. It is important to point out how an image can contain multiple logo objects and the objects in the same image are not necessary of the same class.

Figure 3.1: Same brand treated as different class using the suffixes '-1' and '-2' [83].

Category	Brand	Images	Objects
Food	932	53,350	64,276
Clothes	604	31,266	37,601
Necessities	432	24,822	30,643
Electronic	224	9,675	12,139
Transportation	213	10,445	12,791
Leisure	111	5,685	6,573
Sports	66	3,953	5,041
Medical	47	3,945	5,185
Others	371	15,513	20,016
Total	3,000	158,652	194,261

Table 3.2: Brand, images and objects number for each category in the LogoDet-3K dataset [83].

3.1.1 Dataset statistics

As we can see from Figure 3.2 showing the distribution of the number of objects for each logo class, the main issue of LogoDet-3K is the low number of objects for some classes. Analyzing the boxplot in Figure 3.3 and the statistics in Table 3.3 we can see how the 25% of the classes have a number of images lower or equal to 27, and half of the classes have a number of images lower or equal to 54. This is the main issue with this dataset: since several classes have a low number of images, it could be challenging for the model to correctly predict these classes.

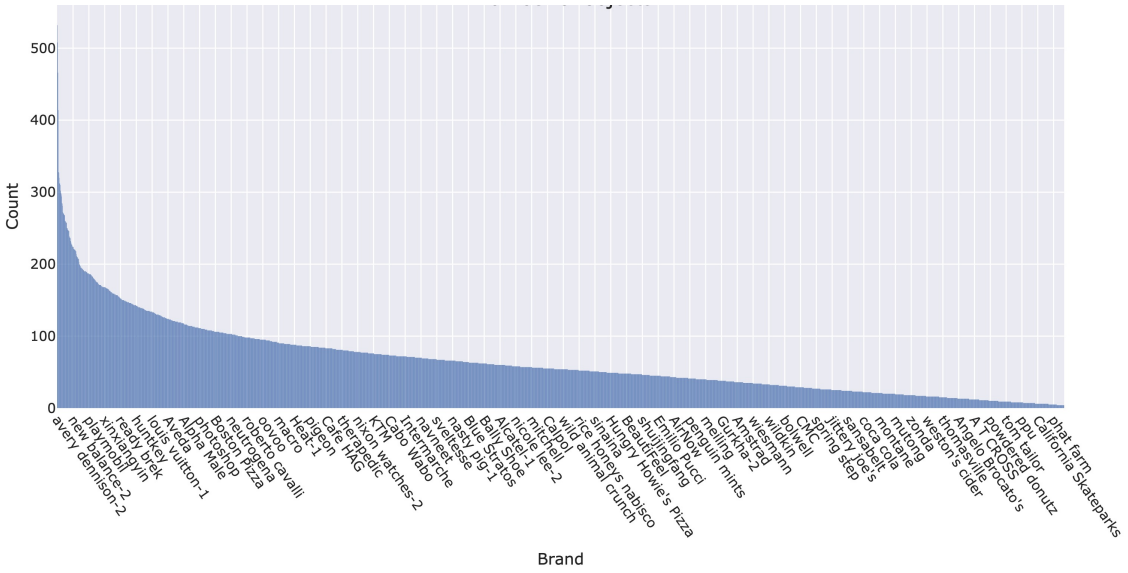


Figure 3.2: Number of objects in LogoDet-3K for each class.

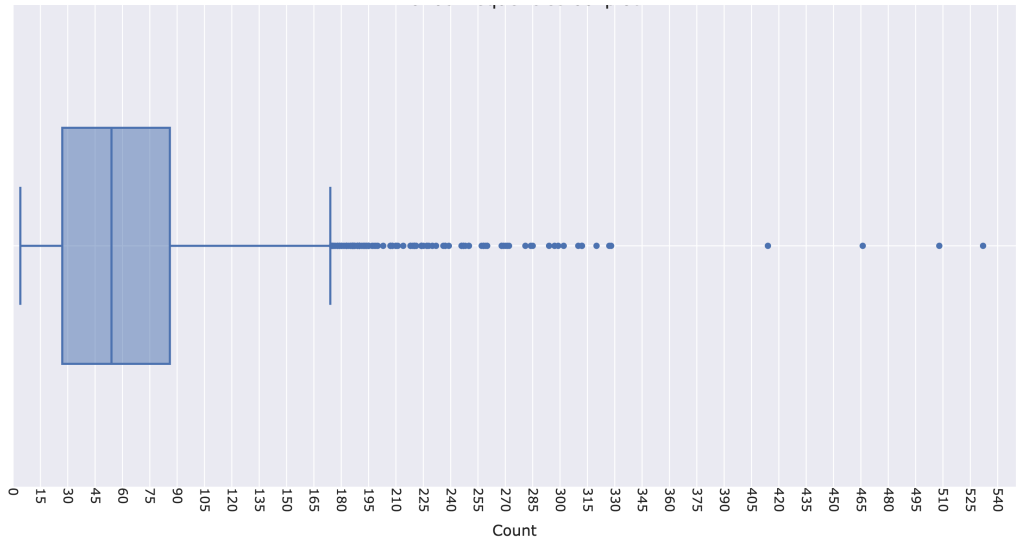


Figure 3.3: Boxplot of the number of objects in LogoDet-3K for each class.

LogoDet-3K statistics			
Q_1	Q_2	Q_3	Q_4
27	54	86	532

Table 3.3: Quartiles relative to the number of objects for each class in LogoDet-3K.

3.2 Inconsistencies in the dataset

The LogoDet-3K dataset is available at the GitHub repository[1] mentioned in the original paper [83]. The dataset is provided as 9 directories corresponding to each category (as described in section 3.1), and for each category there are several subdirectories corresponding to each logo class. Within the folder of each class are images and metadata containing the bounding box coordinates and the actual class of the logo, since an image can contain logos of different classes. The dataset structure can be summarized as follows:

[1]PyCIL GitHub repository: https://github.com/G-U-N/PyCIL

LogoDet-3K
- Food
- Clothes
- Necessities
- Electronic
- Transportation
- Leisure
- Sports
- Medical
- Others
  - avery dennison-1
    - 1.jpg
    - 1.xml
    - ...
  - avery dennison-1
    - 1.jpg
    - 1.xml
    - ...
  - ...

Although the number of subdirectories in each category sums up to 3,000 (i.e. the number of classes of the dataset), using the dataset provided in the GitHub repository, when analyzing the metadata of the '.xml' files, the number of unique logo classes is 2,993. This is a discrepancy in the dataset between what is described in the original paper and the dataset downloaded from the GitHub repository.

Another issue related with this dataset is that some images contain errors regarding the annotations in the metadata. For example, the two images:

1. `Others/avery dennison-1/27.jpg` labeled in `Others/avery dennison-1/27.xml`
2. `Others/avery dennison-2/56.jpg` labeled in `Others/avery dennison-2/56.xml`

are labeled as shown in Figure 3.4.

(a) Image `Others/avery dennison-1/27.jpg` (b) Image `Others/avery dennison-2/56.jpg`

Figure 3.4: Example of errors in the annotation of the images.

From Figure 3.4 it is clear how the labeling is not consistent between the two images. In the image `Others/avery dennison-1/27.jpg` the logos are correctly treated as different classes. On the other hand, in the image `Others/avery dennison-2/56.jpg` the two logos are treated as the same class.

The vast majority of images are correctly labeled, however, especially for those classes related to the same brand and distinguished by the use of suffixes, some errors such as the one just discussed are present.

Chapter 4

Developed methods for logo recognition

As discussed in chapter 1, the system developed in this thesis follows a pipeline consisting of two stages:

1. **Object Proposal**: detection of all the logos in the image
2. **Classification**: recognition of the class of each logo

The first stage is done by YOLO (see subsection 2.1.1), while the actual logo recognition is performed by the classifier in an incremental learning setup. The system pipeline is shown in Figure 4.1.

The detector used in the first stage is a class-agnostic logo detector: it is a logo detector since it only detects logos in the image; and it is class-agnostic since the number of classes detected is only one. In fact, in this first stage the detector will be responsible for finding any generic logo (i.e., a single class) in the image. This is a crucial aspect if we seek to achieve incremental learning logo recognition, because in this way detection can be decoupled from recognition. If the logo detector localizes any generic logo in the image, and delegates the actual recognition to the CIL classifier, there is no need to develop an incremental learning detector as well.

An important aspect of the classifier is the size of the model (in terms of the number of parameters). To this reason, a part of this work aims to decrease the number of parameters of the classifier using two different techniques. The first one is described in subsection 4.2.6: using masking and pruning introduced in section 2.3.3 it aims to prune the model after the training of each incremental step, thus reducing the number of parameters. The second technique is the Knowledge Distillation (KD) described in section 4.3, where a smaller model is trained with the supervision of a bigger model trained on the same task.

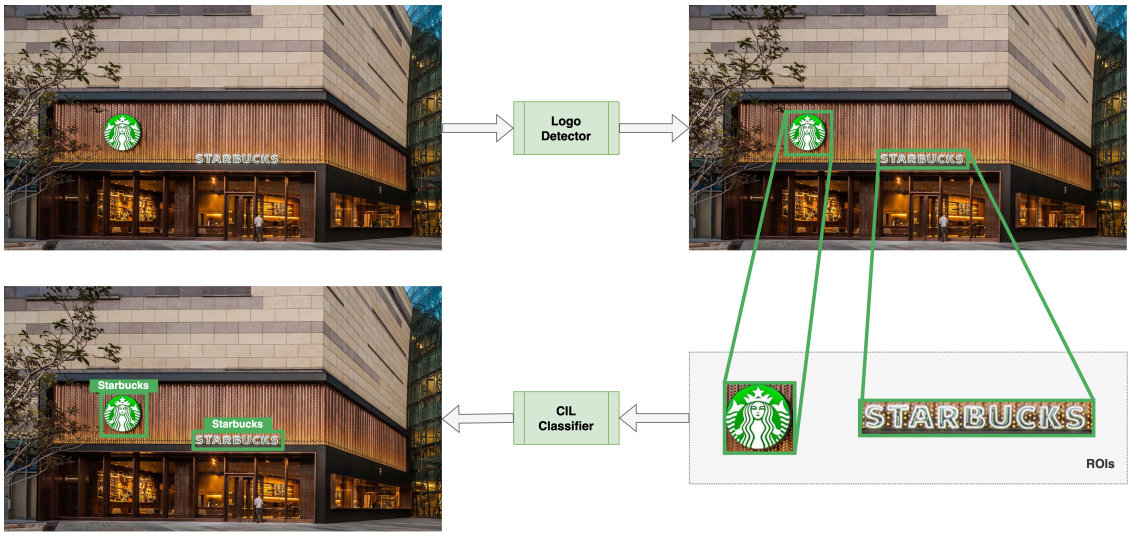


Figure 4.1: Pipeline of the system: the class-agnostic logo detector generates Regions of interest (ROIs), then the recognition is performed by the CIL classifier.

The code developed in this thesis is available in the GitHub repository[1] linked below. The system is developed using the Python programming language, and the code is implemented using PyTorch [57], a widely used deep learning framework. The training of the models is done using the NVIDIA Tesla K80 GPU with 12GB of memory.

This chapter describes all the details about the logo detector in section 4.1 and continues with the classifier in section 4.2. Then, in subsection 4.2.6 and section 4.3 are discussed some techniques to reduce the number of model parameters. In order to evaluate the drop in performance of the CIL classifier when compared to a standard close-set classification task (i.e. all classes are available from the beginning and no new ones are introduced over time), the chapter ends in section 4.4 with the introduction of different baselines.

4.1 Region proposal

A key component of the system is the logo detector. Formally, this first step is an object detection task, where the objects to be detected are the logos in the image. The logo detector outputs the coordinates of the bounding boxes corresponding to the logos in the image. Bounding boxes can be used to crop the portions of the image creating regions of

[1] GitHub repository with the code developed in this thesis:
https://github.com/gianlucagiudice/logo-detection-recognition

interest (ROIs). ROIs correspond to what the detector considers a generic logo and can be used directly as input to the CIL model that classifies (i.e. recognizes) the logo.

4.1.1 YOLOv5m6

The class-agnostic logo detector is based on YOLOv5-v6.1 [1]. There are different versions of YOLOv5-v6.1, depending on the size of the model and the size of the input image. In general, larger models perform better, with the disadvantage of longer training time, longer inference time and more computational resources required to train the model. On the other hand, a small model might achieve very low performance, thus misleading the final evaluation of the system that relies heavily on the detector. Therefore, a model that is too small could act as a bottleneck for the whole system. A more detailed analysis of this trade-off is shown in Figure 4.2 and Table 4.1.

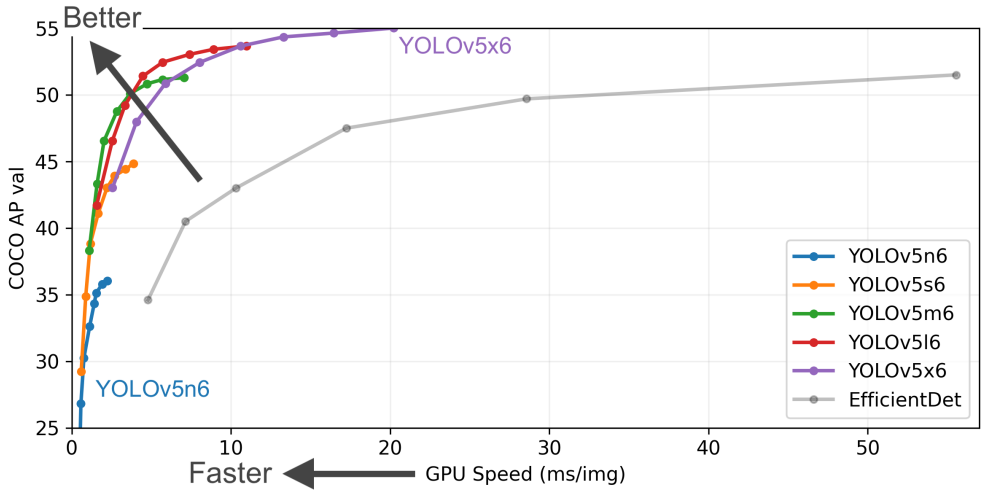


Figure 4.2: Performance and speed comparison for the YOLOv5-v6.0 family of models considering the mAP@0.5:0.95 metric measured on the 5000-image COCO val2017 [44] dataset. The plot shows the performance of different models varying the input size of the image from 256 × 256px to 1536 × 1536px. Image from [1].

The class-agnostic logo detector is based on YOLOv5m6, shown in Table 4.1, with an image input size of 512 × 512px and pre-trained on the COCO dataset [1]. The model is then finetuned for 30 epochs on the task of class-agnostic logo detection.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU (ms)	Speed GPU (ms)	# params (M)	FLOPs @640px (B)
YOLOv5n	640	28.0	45.7	45	6.3	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	76.8	111.4
YOLOv5x6	1280	55.0	72.7	3136	26.2	140.7	209.8

Table 4.1: Performance comparison considering the mAP@0.5:0.95 metric of all checkpoints of YOLOv5-v6.0 trained for 300 epochs on COCO dataset [1].

4.2 Classification

Given the ROIs, the classification of each cropped region is performed by a model trained using CIL techniques. To achieve this goal, the classifier is trained using the DER algorithm [86] described in subsection 2.3.3.

4.2.1 ResNet-34

The DER algorithm introduces a new feature extractor $\mathcal{F}_t$ at each incremental step. This feature extractor consists of a CNN. In the developed system the backbone of the CNN is ResNet-34 [26] pre-trained on ImageNet1000 [16].

The peculiar aspects of ResNet architecture are the Residual Connections, shown in Figure 4.3. These connections are types of skip-connection that, instead of learning unreferenced functions, learn residual functions with respect to the layer inputs. Formally, denoting the desired underlying mapping as $\mathcal{H}(x)$, we let the stacked nonlinear layers fit another mapping of $\mathcal{F}(x) := \mathcal{H}(x) - x$. The original mapping is recast into $\mathcal{F}(x) + x$.

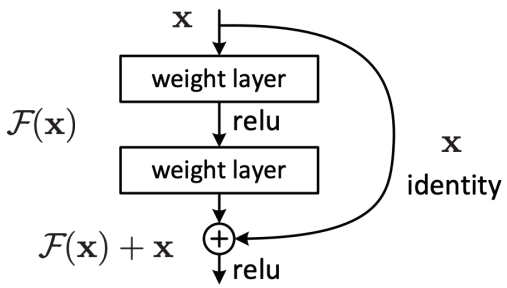
The intuition is that it is easier to optimize the residual mapping instead of optimizing the original, unreferenced mapping. In the extreme case, if an identity mapping is optimal, it is easier to push the residual to zero than to fit an identity mapping using a stack of non-linear layers [26].

The architecture of ResNet-34, shown in Table 4.2, consists of multiple stacked building blocks. Each building block, represented by the brackets in Table 4.2, adopts the residual connection described above. The combination of both skip-connection and batch-normalization (BN) [30] in ResNet enable the training of a very deep neural network, achieving high performance. The total number of parameters of ResNet-34 used as feature

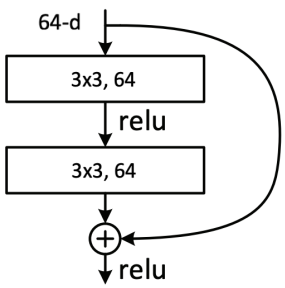
extractor at each incremental step is 23 million.

ResNet-34 architecture		
Layer name	**Output size**	**Layer**
conv1	112×122	$7 \times 7, 64$, stride 2
	56×56	3×3 max pool, stride 2
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
	1000×1	average pool
FC	1000×1	1000-d FC layer, softmax

Table 4.2: Architecture of ResNet-34, the brackets represent a stack of building blocks.



(a) Residual learning: a building block



(b) An example of residual connection in a building block. This building block corresponds to conv2_x in Table 4.2

Figure 4.3: Residual connection in ResNet architecture [26].

4.2.2 CIL classifier

An implementation of the DER algorithm is provided by PyCIL[2] (see subsection 2.3.2), and is used as a starting point for the development of the CIL classifier. All the changes made to the implementation of DER in PyCIL are described in the following sections, and can be found in this[3] GitHub repository.

The implementation of DER in PyCIL reflects the description given in the DER paper, with only one difference to the third phase of the DER algorithm (called 'classifier learning phase' in subsection 2.3.3). In fact, in PyCIL the 'Classifier Learning Stage' is performed using the WA described in section 2.4; in this way it is possible to solve the problem of biased weights in the FC layer without the need to train the model further. It is important to emphasize that, although this step of the algorithm is modified, the final performance reported by PyCIL is very similar to that obtained from the original DER implementation (see Table 2.1).

4.2.3 Regularization techniques

To avoid model overfitting, the architecture of the NN created with the DER algorithm is modified. Starting from the original architecture, a dropout layer is added before the FC layer represented in Figure 2.5 by the classifier $\mathcal{H}_t$.

Dropout

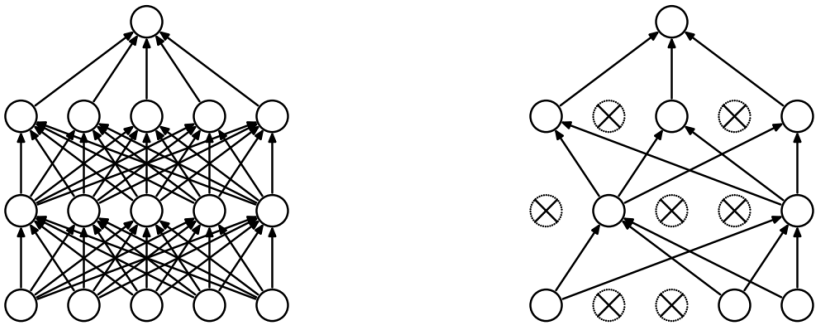
DNNs with a large number of parameters are prone to overfitting the training data and do not generalize well on the test set. Dropout [73] is a technique to address this problem.

At training time, the units of the NN, along with their connections, are dropped randomly with a given probability p (a common value is $p = 0.5$). At test time, all units are present, but with weights scaled by p (i.e. w becomes pw). This procedure is shown in Figure 4.4.

The idea is to prevent co-adaptation, where the NN becomes too reliant on particular connections, as this could be a symptom of overfitting. Intuitively, the dropout can be considered as the creation of an implicit ensemble of NNs.

[2]PyCIL GitHub repository:
https://github.com/G-U-N/PyCIL

[3]GitHub repository relative to the CIL model of this thesis:
https://github.com/gianlucagiudice/PyCIL



(a) Standard neural network

(b) Neural network after applying dropout

Figure 4.4: Dropout mechanism in a Neural Network with 2 hidden layers [73].

4.2.4 Data augmentation

As discussed in subsection 3.1.1, one problem with the LogoDet-3K dataset is that some classes have a low number of images. Moreover, as shown in Table 3.3, these classes are not an exception but the majority of those in the dataset. In fact, 50% of the dataset has a number of images less than or equal to 54.

Training a model using a small number of examples per class can be a problem in machine learning in general, but this is particularly true in the deep learning domain, where models require a huge amount of data.

Image augmentation is a data augmentation method that generates more training data from the existing training samples. This is a very useful technique to address the problem of low number of images per class in LogoDet-3K. In addition, data augmentation is another way to avoid overfitting, as it creates synthetic samples by adding random perturbations to the original image, yet preserving its main characteristics, thus achieving better generalization.

The type of data augmentation adopted during the training of the model is called online data augmentation. This technique consists in applying a random transformation to the original image before feeding it to the model. In this way, it is theoretically possible to obtain an infinitely large dataset. The process of data augmentation is heavily dependent on the type of transformations applied to the images: an excessive transformation of the original image may mislead the model in learning the main features of the image, while keeping the image as similar as possible to the original one may have the same effect as not applying data augmentation at all.

The transformations used to augment the original dataset, shown in Figure 4.6, and used to train the CIL classifier are the following:

- **Geometric transformations**:
  - **Random affine**: transforms the image while keeping the centre invariant.
  - **Random perspective**: perspective transformation of the image.
- **Color transformations**:
  - **Adjust sharpness**: transforms the original image into a sharper one according to the sharpening factor. This parameter is chosen a priori and is fixed for each transformation.
  - **Posterize**: reduces the number of bits for each color channel.
  - **Color jitter**: randomly changes the brightness, contrast, saturation and hue of an image.

As previously stated, the data augmentation is performed in an online manner, in this way it is possible to obtain a slightly different image at each training epoch of the model. The original image is transformed using the transformations listed above, but the final image is a combination of geometric and color transformations. This procedure is shown in Figure 4.5, and is applied to different logos in Figure 4.7.

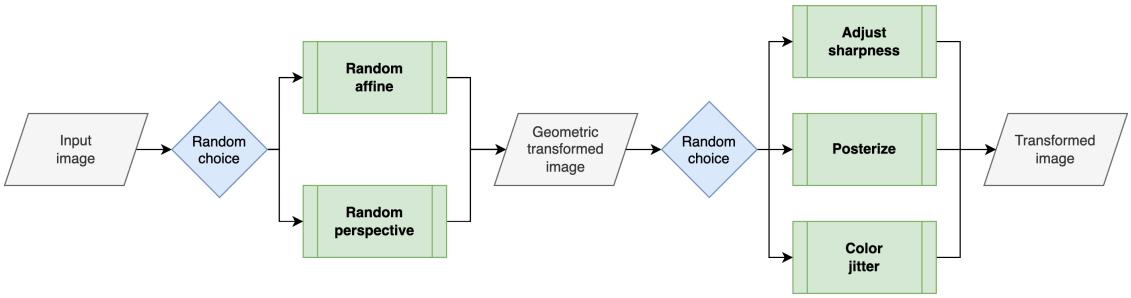


Figure 4.5: Image transformation pipeline for data augmentation: first the input image is transformed via a geometric transformation, then a color transformation is applied.

(a) Original image

(b) Random affine transformation

(c) Random perspective transformation

(d) Adjust sharpness

(e) Posterize

(f) Random color Jitter

Figure 4.6: Examples of image transformations. The transformations (d) and (e) are shown in a single image since, given the sharpness factor and the number of bin for the posterization, the result of the transformation is always the same.

(a) Logo 1

(b) Logo 2

(c) Logo 3

(d) Logo 4

(e) Logo 5

(f) Logo 6

Figure 4.7: Examples of data augmentation on different logos: the first column represents the original image, the other columns are some transformation applied to the original image following the pipeline described in Figure 4.5.

4.2.5 Training

The training of the CIL model can be divided into two main steps: the initial task and the iterations of CIL. At the initial task the model is trained for 200 epochs, while for the incremental learning iterations it is trained for 150 epochs.

The number of training epochs at the initial task is higher because in this phase the network requires a longer time to converge, this is due to these two factors: the neural network must adapt the weights trained on ImageNet to the logo classification task; it is assumed that the number of classes at the initial task is greater than the number of classes introduced at each incremental learning iteration.

Early stopping

Early stopping is used to avoid overfitting of the model. This is done by monitoring the accuracy of the model during training on the validation set. The hyperparameters of early stopping are the following: the number of epochs with no improvement after which training will be stopped is set to 30 (i.e. patience); the minimum change in the monitored accuracy to be considered as an improvement is set to 0.5% (i.e., the minimum delta).

SGD optimizer

To update the model weights two different optimizers are tested. One of them is Stochastic Gradient Descent (SGD), an iterative method for optimizing an objective function.

As described in [89], in deep learning the objective function is usually the average of the loss functions for each example in the training dataset. Given a training dataset of n examples, we assume that $f_i(\mathbf{w})$ is the loss function with respect to the training example of index i, where $\mathbf{w}$ is the parameter vector. Then the objective function is given by:

$$f(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w}) \tag{4.1}$$

The gradient of the objective function at $\mathbf{w}$ is computed as:

$$\nabla f(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{w}) \tag{4.2}$$

Using gradient descent, the computational cost for each independent iteration is $\mathcal{O}(n)$, which grows linearly with n. Therefore, when the training dataset is larger, the cost of gradient descent for each iteration will be higher.

SGD reduces the computational cost at each iteration by uniformly sampling an index $i \in \{1, ..., n\}$ from data examples, and computes the gradient $\nabla f_i(\mathbf{w})$ to update $\mathbf{w}$ as follows:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f_i(\mathbf{w}) \tag{4.3}$$

where η is the learning rate ($\eta = 0.1$ in the experiments described in chapter 5). In this way the computational cost for each iteration drops from $\mathcal{O}(n)$ of the gradient descent to the constant $\mathcal{O}(1)$ of SGD. Moreover, the stochastic gradient $\nabla f_i(\mathbf{w})$ is an unbiased estimate of the full gradient $\nabla f(\mathbf{w})$, since:

$$\mathbb{E}_i \nabla f_i(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{w}) = \nabla f(\mathbf{w}) \tag{4.4}$$

This means that, on average, the stochastic gradient is a good estimate of the gradient.

A compromise between computing the true gradient and the gradient with a single sample is to compute the gradient against more than one training sample (called a 'mini-batch'). This means replacing the gradient $\nabla f_i(\mathbf{w})$ on a single observation with one over a small batch $\mathcal{B}_t = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ (for the experiments described in chapter 5, $|\mathcal{B}_t| = 2048$). Then, the gradient $\mathbf{g}_t$ over a small batch $\mathcal{B}_t$ is given by:

$$\mathbf{g}_t = \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \nabla f_i(\mathbf{w}) \tag{4.5}$$

Note that, during an epoch, the examples are not reused for multiple batches. The mini-batch SGD algorithm is shown in Algorithm 1, as described by Goodfellow et al. in [23].

Algorithm 1 SGD algorithm with minibatches

Require: Learning rate η
Require: Initial parameters $\mathbf{w}$
while stopping criterion not met **do**
Sample a minibatch $\mathcal{B}$ of m examples from the training set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ with corresponding labels $\mathbf{y}^{(i)}$.
Compute gradient estimate: $\hat{\mathbf{g}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}_t} \nabla f_i(\mathbf{w})$.
Apply update: $\mathbf{w} \leftarrow \mathbf{w} - \eta \hat{\mathbf{g}}$.
end while

Adam optimizer

The second optimizer used in the experiments is Adam [35], which can be considered as a combination of RMSProp [27] and momentum [77]. Adam is an adaptive learning rate optimizer, i.e. it uses a separate learning rate for each parameter and automatically adapts these learning rates throughout the course of learning. A key component of Adam is that it uses exponential weighted moving averages (also known as leaky averaging) to obtain an estimate of the momentum of the gradient.

To estimate the first-order and the second-order moments of the gradient, Adam uses two state variables:

$$\begin{aligned} \mathbf{v}_t &\leftarrow \beta_1 \mathbf{v}_{t-1} + (1-\beta_1)\mathbf{g}_t \\ \mathbf{s}_t &\leftarrow \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \end{aligned} \tag{4.6}$$

where β_1 and β_2 are nonnegative weighting parameters. In the experiments in chapter 5, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Initially, both $\mathbf{v}_0$ and $\mathbf{s}_0$ are set to 0, resulting in a bias towards smaller values. For this reason, Adam computes the bias-corrected $\hat{\mathbf{v}}_t$ and $\hat{\mathbf{s}}_t$ as follows:

$$\begin{aligned} \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1-\beta_1^t} \\ \hat{\mathbf{s}}_t &= \frac{\mathbf{s}_t}{1-\beta_2^t} \end{aligned} \tag{4.7}$$

Then, the gradient is rescaled accordingly to consider both the first-order and second-order moments:

$$\mathbf{g}'_t = \eta \frac{\hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon} \tag{4.8}$$

where, η is the learning rate (set to 0.001 in the experiments in chapter 5) and $\epsilon = 10^{-8}$ is a constant to achieve numerical stability.

Finally, the parameters are updated as follows:

$$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \mathbf{g}'_t \tag{4.9}$$

The Adam algorithm is shown in Algorithm 2, as described by Goodfellow et al. in [23].

Learning rate decay

For both SGD and Adam, an exponential decay scheduling is applied to the learning rate: once one epoch reaches one of the predetermined epochs (called milestones) the learning rate of the optimizer is multiplied by a factor $\gamma = 0.1$. For the training of the CIL classifier, the milestones are set to 60, 100 and 150 for the initial task, and to 40, 75 and 100 for each incremental learning iteration.

Adjusting the learning rate is an important aspect of the optimization process. Some of the motivations behind learning rate scheduling are described below:

- The magnitude of the learning rate is a key factor in the optimization procedure: if the learning rate is too large, optimization diverges, while if it is too small, either training takes too long or a suboptimal result is achieved.
- If the learning rate remains high, the algorithm may bounce around the minimum or even jump over it and thus failing to reach optimality.

Algorithm 2 The Adam algorithm

Require: Learning rate η
Require: Exponential decay rates for moment estimates, β_1 and β_2 in [0, 1) (Suggested defaults: 0.9 and 0.999 respectively).
Require: Small constant ϵ used for numerical stabilization (Suggested default: 10^{-8}).
Require: Initial parameters $\mathbf{w}$
Initialize 1st and 2nd moment variable $\mathbf{s} = 0$, $\mathbf{r} = 0$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch $\mathcal{B}$ of m examples from the training set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ with corresponding labels $\mathbf{y}^{(i)}$.
Compute gradient estimate: $\hat{\mathbf{g}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}_t} \nabla f_i(\mathbf{w})$.
$t \leftarrow t + 1$
Update biased first moment estimate: $\mathbf{v} \leftarrow \beta_1 \mathbf{v} + (1 - \beta_1)\hat{\mathbf{g}}$
Update biased second moment estimate: $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2)\hat{\mathbf{g}}^2$
Correct bias in first moment: $\hat{\mathbf{v}} = \frac{\mathbf{v}}{1-\beta_1^t}$
Correct bias in second moment: $\hat{\mathbf{s}} = \frac{\mathbf{s}}{1-\beta_2^t}$
Compute update: $\Delta\mathbf{w} = \eta \frac{\hat{\mathbf{v}}}{\sqrt{\hat{\mathbf{s}}}+\epsilon}$
Apply update: $\mathbf{w} \leftarrow \mathbf{w} - \Delta\mathbf{w}$.
end while

The learning rate decay is used to train models for the experiments described in chapter 5. An example of the effect of the learning rate decay is shown in Figure 4.8. By plotting the accuracy of the model at each training epoch, it is possible to notice that, when a milestone is reached (epoch 60) and the learning rate is scaled by a factor of 10^{-1}, there is an improvement in the accuracy of the model in subsequent epochs.

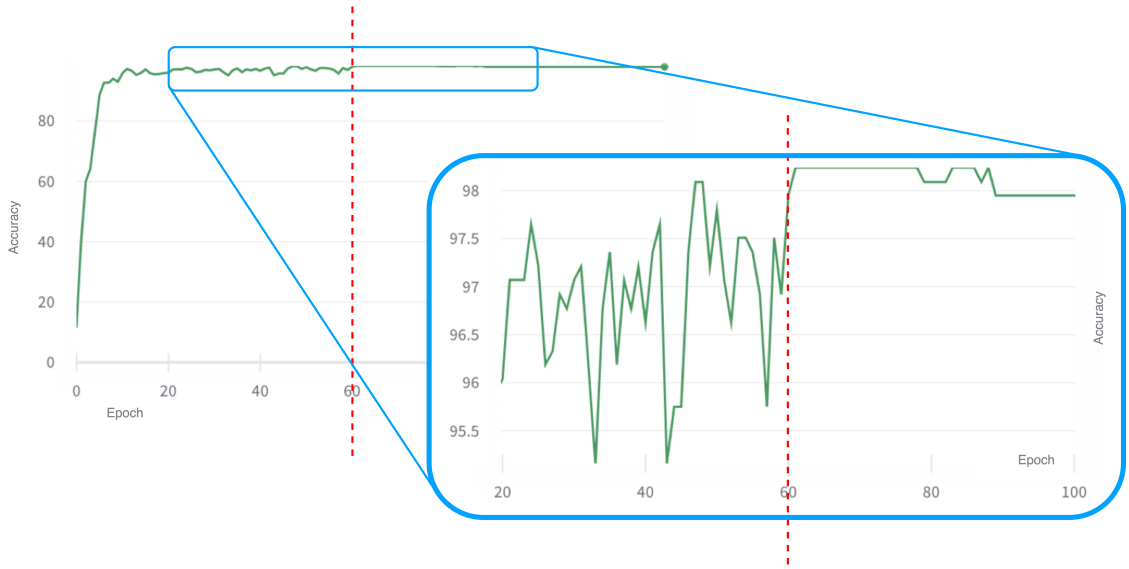


Figure 4.8: Effect of the learning rate decay: the figure shows the accuracy of the model at each training epoch. At epoch 60, the learning rate is scaled by a factor of 10^{-1}, this leads to an increase in the model's accuracy in subsequent epochs.

4.2.6 Pruning

The DER algorithm, described in subsection 2.3.3, addresses the problem of class incremental learning by expanding the neural network architecture at each incremental step. The super-feature extractor at step t of incremental learning is given by the following concatenation $\Phi_t(\mathbf{x}) = [\mathcal{F}_0, \mathcal{F}_1(\mathbf{x}), ..., \mathcal{F}_t(\mathbf{x})]$. As discussed in subsection 4.2.1, the feature extractor is a CNN, which in this thesis is implemented using ResNet-34.

Given the definition of the super-feature extractor, it is possible to see how the parameters of the neural network grow linearly as a function of the number of incremental learning steps. It is important to note that as the parameters of the super-feature extractor increase, those of the classifier $\mathcal{H}_t$ do as well. The classifier $\mathcal{H}_t$ corresponds to the fully connected layer of the neural network. At each incremental learning task new classes are introduced, thus new connections are created between the classifier predicting the output class and the super-feature extractor.

This means that, using the LogoDet-3K dataset described in chapter 3, and the incremental learning setup described in subsection 5.2.2, at the 8th step of CIL the total number of parameters is given by: the number of parameters at task 0 (initial task), in which 22 million parameters (number of parameters of ResNet-34) are used; the 8 incremental learning steps, each adding 22 million parameters to the total.

After only 8 iterations of incremental learning, the final neural network reaches 205 million parameters, which is a large number of parameters compared to a single ResNet-34 used as backbone of each feature extractor. For this reason, there is a need to reduce the number of parameters of the neural network. To do so, the pruning of the network is implemented using masks, as described in section 2.3.3.

Each mask $\mathbf{m}_l$ associated with the channels of each layer l of the CNN is computed as shown in Equation 2.7. At the incremental learning task t, after the training of the feature extractor $\mathcal{F}_t(\mathbf{x})$, the variable s is set to a really high value (10^4 in the implementation). Since s is fed into the sigmoid used as gating function, a steep activation mechanism is obtained (which leads to a binary activation when the limit of s tends to infinity, but this behavior is achieved in practice using high value of s). So, if a value e_{li} of the mask $\mathbf{e}_l$ is positive, m_{li} tends to 1, else if e_{li} is negative, m_{li} tends to 0. To perform the actual binarization of each mask value, a threshold τ with a low value set to 10^{-4} is used as follows:

$$m_{li} = \begin{cases} 1, & \text{if } m_{li} > \tau \\ 0, & \text{if } m_{li} \leq \tau \end{cases}$$

An important aspect to consider during the pruning of each convolutional layers l of the feature extractor $\mathcal{F}_t(\mathbf{x})$ is the input and output dimension of the feature maps. Since $\mathbf{m}_l$ is a channel-level mask, the effect of pruning is to reduce the number of channels in the convolutional layers of the CNN. This results in a change in the output dimension of the convolutional layer l, so the input dimension of the convolution $l + 1$ must be consistent with that output dimension.

Another point to consider is the architecture used to implement the feature extractors $\mathcal{F}_t$. In fact, each feature extractor uses ResNet-34 as its backbone, which is an architecture based on residual connections (see Table 4.2 and Figure 4.3). This is a limitation from a pruning perspective, as the residual connection adds a new constraint relative to the input dimension of convolutional layers. This issue is shown and described in Figure 4.9.

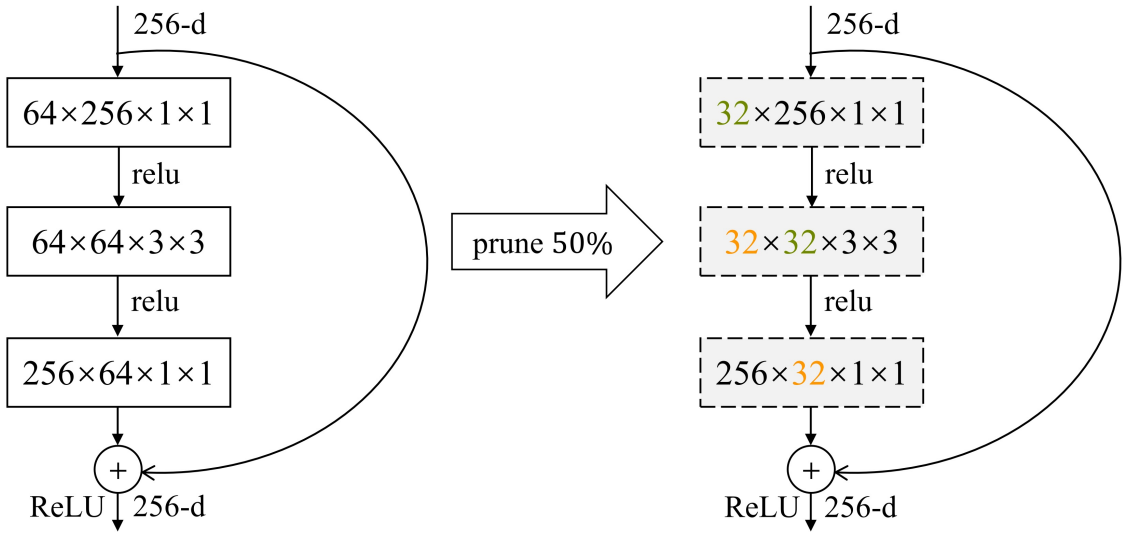


Figure 4.9: Pruning strategy of ResNet. The pruning is performed in such a way that the final size of the output is maintained, so that the residual connection can be added. Image from [49].

The use of residual connections requires pruning at the block level. The definition of a block depends on the specific architecture of ResNet. In the case of ResNet-34, a block consists of two convolutional layers, as shown in Table 4.2, so pruning is applied to pairs of convolutions in order to add the identity carried by the residual connection.

4.3 Knowledge distillation

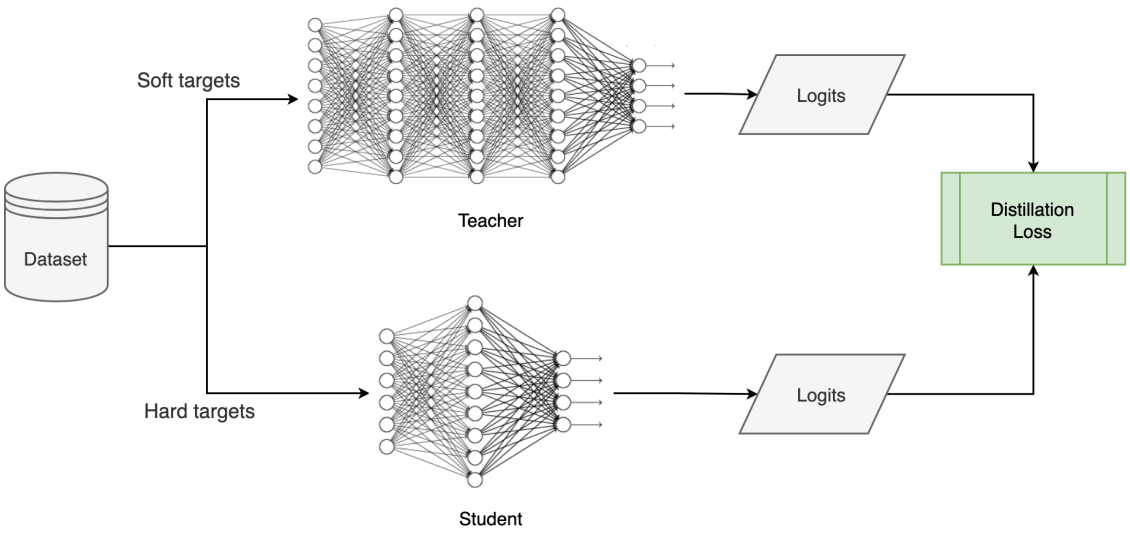


Figure 4.10: Knowledge distillation pipeline. The student model is trained by exploiting the teacher model using the distillation loss. This loss considers both the logits of the teacher and the real data labels.

Another approach used to tackle the problem of the large number of model parameters is Knowledge Distillation (KD) [28]. KD is the process of transferring knowledge from a pre-trained large model to a smaller one. Even if large models (such as very deep neural networks like ResNet-152) have higher knowledge capacity than small models, this capacity might not be fully exploited. A large model with regularization (e.g. dropout) generalizes better than a small model when trained directly on the data and labels, however, a small model can be trained to generalize better with the help of a large model. This training setting is referred to as 'teacher-student', where the smaller model is the student and it is trained to mimic the larger one which is the teacher.

As described by Gou et al. in [24], Response-Based Knowledge is a type of KD which refers to the output layer of the teacher model. In this thesis, the implementation of the KD is based on the paper of Hinton et al. in [28]. The main idea is to directly mimic the final prediction of the teacher model. The reason behind this is that the output probabilities of a trained model give more information than the raw labels, because they assign non-zero probabilities to incorrect classes. Knowledge is transferred from the teacher to the student using a loss function, known as the distillation loss, which captures the differences between the logits of the student and teacher models. As the loss is reduced over time, the student becomes more accurate in making predictions similar to those of the teacher. The

teacher-student setting for KD matching the logits between the two models is shown in Figure 4.10.

The distillation loss used to train the student model is computed considering both the logits produced in output by the teacher (soft targets) and the real data labels (hard targets). The probability p_i of each class i, is computed by the teacher model using its logit z_i as follows:

$$p_i = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{4.10}$$

The small model is trained to minimize the KL Divergence between its output probability distribution and the one of the large network. One issue with this technique is that the probabilities assigned to incorrect classes by the large network are often very small and do not contribute to the loss. For this reason, the logits are softened using a 'temperature' T in the softmax, smoothing the probability distribution and revealing inter-class relationships learned by the teacher. Then, the softened probability is given by:

$$p_i^t = \frac{exp(\frac{z_i^t}{T})}{\sum_j exp(\frac{z_i^t}{T})} \tag{4.11}$$

where higher values of T produce softer probabilities. The prediction p_i^t is called soft target, shown in Figure 4.10. Note that the teacher's output probability is denoted by p_i^t using the letter t, as opposed to the student's output probability denoted by p_i^s with the letter s.

Similarly, the Equation 4.11 is used to compute the class probability p_i^s predicted by the student using its logit z_i^s and the same temperature T. In addition to the soft targets of the teacher, Hinton et al. in [28] show that it is beneficial to train the student model to produce the real data labels, called hard targets and shown in Figure 4.10. Therefore, by setting $T = 1$, the student uses Equation 4.11 to compute the 'standard' class probability $\hat{y}_i^s$.

Thus, considering both the KL divergence of the output probability distribution, and the Cross-Entropy between $\hat{y}_i^s$ and the true data label y_i, the final distillation loss is given by:

$$\mathcal{L}_{KD} = \alpha \left[\sum_{i=1}^{\text{output size}} p_i^t \, log \left(\frac{p_i^t}{p_i^s} \right) \right] + (1 - \alpha) \left[- \sum_{i=1}^{\text{output size}} y_i^s \, (\log \hat{y}_i^s) \right] \tag{4.12}$$

where $\alpha \in [0, 1]$ is a hyper-parameter controlling the weighted average between the two components.

4.4 Proposed baseline for the classifier

To evaluate the CIL models and compare their performance with a model in a traditional setup, it is necessary to define a baseline. The CIL setup is very useful for introducing

new classes after training, but in general this advantage is at the expense of the model's performance. Consequently, a CIL model performs worse than a traditional one, but a good CIL algorithm should narrow this gap in performance as much as possible.

Without considering the CIL aspect, different approaches can be defined to handle a classical classification problem. This chapter discusses the different baselines introduced with the aim of comparing them with the CIL models.

4.4.1 Baseline without incremental steps

The most direct approach is to define a baseline using the DER algorithm. This means taking all the classes introduced at each iteration of incremental learning and using them from the very beginning, at task 0, to train the model with the DER algorithm. The process described above defines the first type of baseline.

Although this approach appears to be simple and straightforward, it has a problem related to what is described in subsection 4.2.6. In fact, such an approach is not very fair, as the baseline has a number of parameters N equal to the CNN used as the backbone for the feature extractor, while the CIL model trained with the DER algorithm has a number of parameters equal to $N + t \times N$ (where t is the number of iterations of incremental learning). As a result, the CIL model may perform better than the baseline only because it has a significantly larger number of parameters.

4.4.2 ResNet-152 architecture

To solve the problem of the first baseline definition, concerning the difference in parameters between the baseline and the CIL model, a second type of baseline is defined. In this second approach, the DER algorithm is no more exploited, instead it is used ResNet-152 which is a network with a significantly higher number of parameters compared to ResNet-34. This is done to bridge the gap of the number of parameters between the baseline and the CIL model. The CNN based on ResNet-152 is then trained for the classification task considering all the classes.

The architecture of ResNet-152 is very similar to the one described in table Table 4.2, in fact it maintains all the main features such as skip-connections, but has each block composed of more convolutional layers, and the number of stacked blocks increases. The architecture of ResNet-152 consists of 60 million parameters and is described in Table 4.3.

Although this new baseline is an improvement over the previous one, the number of parameters is still lower than in the CIL model, as in the case of the baseline of the first type.

ResNet-152 architecture		
Layer name	**Output size**	**Layer**
conv1	112×122	$7 \times 7, 64$, stride 2
	56×56	3×3 max pool, stride 2
conv2_x	56×56	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 256 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1000×1	average pool
FC	1000×1	1000-d FC layer, softmax

Table 4.3: Architecture of ResNet-152, the brackets represent a stack of building blocks.

4.4.3 DER-based architecture

The third introduced baseline solves the problem of the different number of parameters between the CIL model and the baseline. To this purpose, referring to the exact same setup used for training the CIL model and considering the number of iterations t, the DER algorithm is exploited to define another neural network by performing t incremental learning steps without actually training the new network. Subsequently, since at timestamp t of incremental learning the DER algorithm freezes the weights of the previous feature extractors, all weights, both related to the feature extractors $\mathcal{F}_i$ and to the classifier $\mathcal{H}_i$, are allowed to be trained.

This new network will serve as the baseline, and unlike the DER algorithm, the optimization process will aim to classify all the classes (intended as all those introduced incrementally during the training of the CIL model) without taking into account the auxiliary classifier $\mathcal{H}^a$ and the masks for pruning described in subsection 2.3.3.

Chapter 5

Experiments

This chapter presents the experiments designed to evaluate the system described in chapter chapter 4. The chapter is structured as follows: section 5.1 specifies the setup used for the class incremental learning task; section 5.2 discusses the experiments related to the CIL model; section 5.3 describes the experiments aimed to reduce the number of model parameters using Knowledge Distillation (KD); section 5.4 discusses the experiments related to the logo detector; the chapter ends in section 5.5 reporting the results of experiments combining the logo detector and the CIL classifier.

5.1 Setup

The developed system is tested considering two different CIL setups: the first consists of a subset of 100 classes out of 2993 in the dataset; the second tests the model's scaling capabilities using the entire dataset. In detail, the CIL setups are defined as follows:

1. **100 Classes**: from the initial dataset 100 classes are considered. For experiments, a distinction is made between the case where classes are randomly sampled or the 100 classes with the highest number of images are taken.

   Of these 100 classes, 30 are used for the initial task, then the remaining 70 are used 10 at a time through 7 iterations of incremental learning.

2. **2993 Classes (entire dataset)**: for experiments considering the entire dataset, the first 1000 classes are used for the initial task, followed by 8 iterations of incremental learning, each of which adds 250 new classes.

Training, validation and test sets are built from the individual classes. For each class, the instances are divided as follows:

- **Training set**: 70 %

- **Validation set**: 10 %

- **Test set**: 20 %

5.2 Classifier: CIL model

Top-k accuracy is used to evaluate the performance of the CIL model, focusing on cases where $k = 1$ and $k = 5$. Using this performance metric, a classification is considered correct if the label is present among the first k predictions to which the model assigns the highest probability. Accuracy is then calculated as the percentage of correct predictions.

5.2.1 100 Classes

100 Classes randomly sampled

For the first experiments, the 100 classes are randomly sampled from the 2993 classes in the dataset.

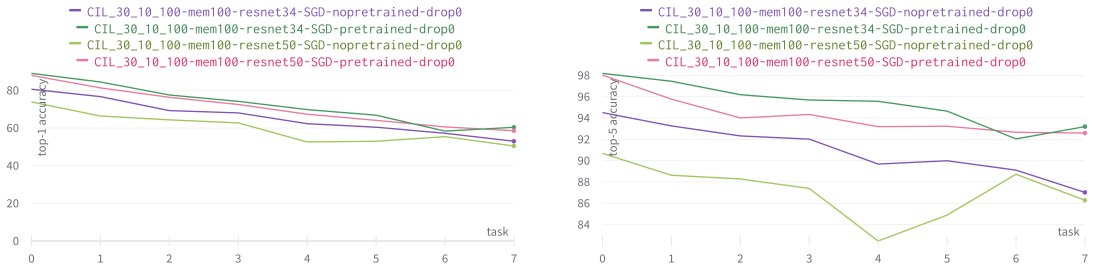
As described in subsection 2.3.3, the DER algorithm saves some samples of the 'old' classes and reuses them during incremental learning iterations. In the following experiments, the memory dedicated to each old class is 100 samples.

The following experiments aim to compare ResNet-34 and ResNet-50 architectures used as feature extractors $\mathcal{F}_t$ (see subsection 2.3.3), distinguishing between cases where CNNs are pre-trained or not on ImageNet. For these experiments, the optimizer is SGD and neither model regularization by dropout layer nor data augmentation are used.

The results of the experiments are shown in Figure 5.1 and in Table 5.1, which report the top-1 and top-5 accuracy calculated on the test set as the CIL tasks vary. As we can see, the pre-trained CNNs perform better, but there is not much difference between ResNet-34 and ResNet-50.

Based on these results, the architecture chosen for the next experiments is ResNet-34 (so that the network is slightly smaller than ResNet-50) with its weights pre-trained on ImageNet.

Other useful insights can be derived from the training history of a task (e.g. task 7) reporting the top-1 accuracy on the training and validation set. Indeed, as we can see from Figure 5.2, the accuracy on the training set is much higher than the one on the validation set, which is a clear sign of overfitting of the model.



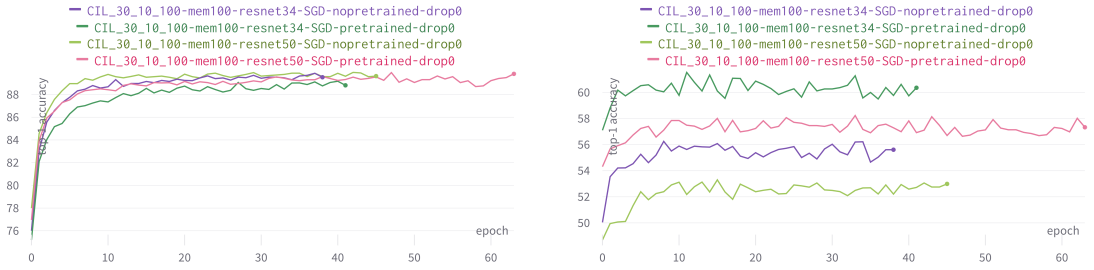
(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.1: Top-1 and top-5 accuracy of models on the test set at different CIL tasks.

Model name	Backbone	Pre-trained	Top-1 acc. (%)	Top-5 acc. (%)
resnet34-SGD-nopretrained-drop0	ResNet-34	no	52.97	87.02
resnet34-SGD-pretrained-drop0	ResNet-34	yes	**60.37**	**93.19**
resnet50-SGD-nopretrained-drop0	ResNet-50	no	50.43	86.28
resnet50-SGD-pretrained-drop0	ResNet-50	yes	58.54	92.59

Table 5.1: Top-1 and top-5 accuracy of models at task 7.



(a) Accuracy on the training set

(b) Accuracy on the validation set

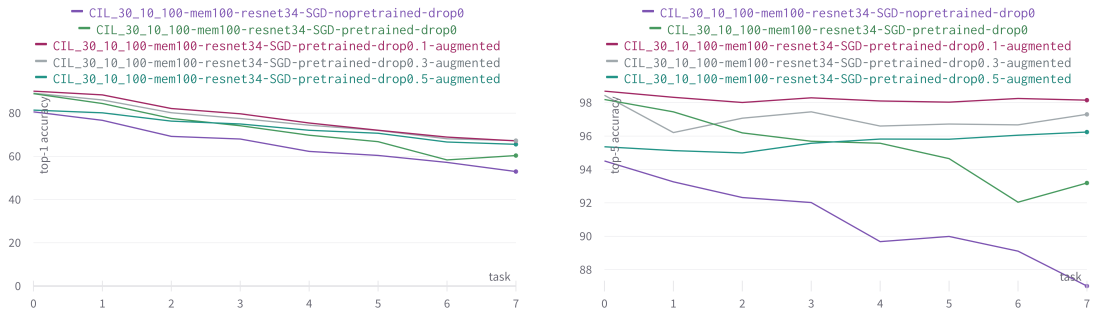
Figure 5.2: Comparison of the accuracy of models at each training epoch. The images show the accuracy on training and validation sets at task 7.

Regularization and data augmentation

Following the analysis discussed above, it is necessary to regularize the model. To do so, the next experiments are performed using the dropout layer (see section 4.2.3) and data augmentation (see subsection 4.2.4). As before, the backbone of the model is ResNet-34 pre-trained on ImageNet.

Using these techniques, the performance is higher than before, as shown in Figure 5.3. From Figure 5.4 we can see that the training accuracy of the models with regularization is lower than that of models without regularization, but the validation accuracy is higher. This is a sign that regularization works as expected.

As shown in Table 5.2, the top-1 accuracy of the best regularized model adopting data augmentation is 7% higher than that without regularization and data augmentation.



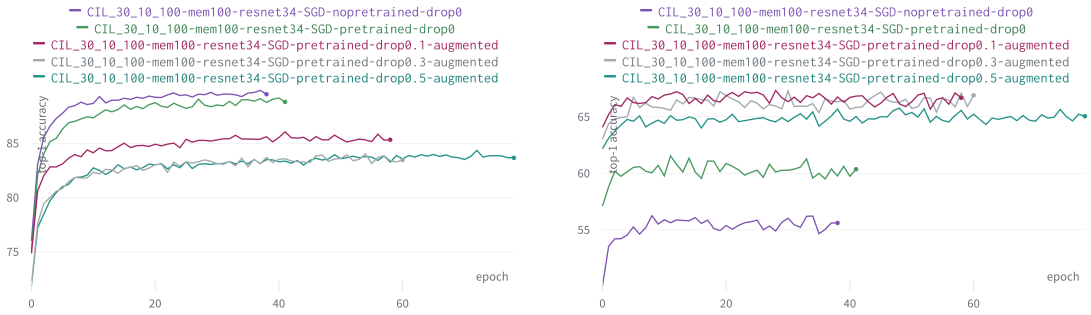
(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.3: Top-1 and top-5 accuracy of the regularized models using data augmentation at different CIL tasks.

Model name	Data augm.	Dropout rate	Top-1 acc. (%)	Top-5 acc. (%)
SGD-nopretrained-drop0	no	0.0	52.97	87.02
SGD-pretrained-drop0	no	0.0	60.37	93.19
SGD-pretrained-drop0.1-augmented	yes	0.1	67.17	**98.15**
SGD-pretrained-drop0.3-augmented	yes	0.3	**67.28**	97.3
SGD-pretrained-drop0.5-augmented	yes	0.5	65.57	96.24

Table 5.2: Regularized models with data augmentation. Top-1 and top-5 accuracy at task 7.



(a) Accuracy on the training set

(b) Accuracy on the validation set

Figure 5.4: Regularized models with data augmentation. The images show the training history of models at task 7, comparing the top-1 accuracy at each training epoch calculated on the training and validation set.

Type 1 baseline

To compare the performance of the CIL model with a standard non-incremental learning setup, a baseline is trained using the same training, validation and test sets as the CIL model. Specifically, these experiments exploit the type 1 baseline definition described in subsection 4.4.1, i.e. the baseline uses the DER algorithm and trains the model at task 0 with all 100 classes. As with the CIL model, the baseline uses: data augmentation, the SGD optimizer, a dropout layer before the FC layer and ResNet-34 pre-trained on ImageNet as the backbone for feature extraction.

The performance comparison between CIL models and baselines considers the top-1 accuracy on the test set, results are shown in Table 5.3. As we can see, the best CIL model performs better than the best baseline, which is the opposite of what is expected. However, as shown in Table 5.4, this is due to the difference in the number of model parameters described in subsection 4.4.1. For this reason, to achieve a more reliable comparison, a new baseline will be used later on.

Model name	**Dropout rate**	**Top-1 acc. (%)**
SGD-pretrained-drop0.1-augmented	0.1	67.17
SGD-pretrained-drop0.3-augmented	0.3	**67.28**
SGD-pretrained-drop0.5-augmented	0.5	65.57
BASELINE_1-pretrained-drop0.1-augmented	0.1	63.04
BASELINE_1-pretrained-drop0.3-augmented	0.3	61.15
BASELINE_1-pretrained-drop0.5-augmented	0.5	57.14

Table 5.3: Performance comparison between CIL models and type 1 baselines. Top-1 accuracy at task 7 and task 0 respectively.

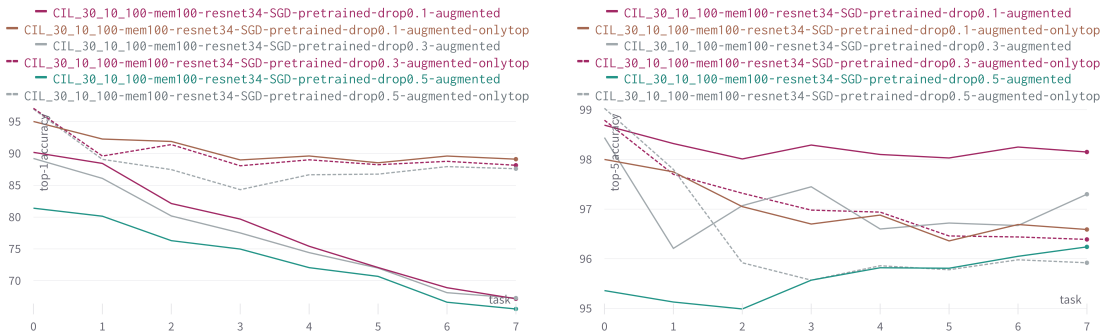
Model name	#Params (M)
SGD-pretrained-drop0.1-augmented	170
SGD-pretrained-drop0.3-augmented	170
SGD-pretrained-drop0.5-augmented	170
BASELINE_1-pretrained-drop0.1-augmented	21
BASELINE_1-pretrained-drop0.3-augmented	21
BASELINE_1-pretrained-drop0.5-augmented	21

Table 5.4: Number of parameters of CIL models and the type 1 baselines at task 7 and task 0 respectively.

Top-100 Classes

In order to assess the extent to which classes with few examples contribute to the deterioration in performance, the following experiments consider only the top-100 classes, i.e. those classes with the most examples (see the left-hand side of the distribution in Figure 3.2). This is useful to be aware of how much the model is limited in performance by the dataset.

Using regularization and data augmentation, models trained on 100 randomly sampled classes and those trained on the top-100 classes are compared in Figure 5.5 and Table 5.5. As we can see, although the top-5 accuracy does not change, considering the top-1 accuracy we can see a drastic improvement in performance. This indicates that classes with few examples significantly deteriorate performance.



(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.5: Comparison between models trained on 100 randomly sampled classes and those trained on the top-100 classes considering top-1 and top-5 at each task.

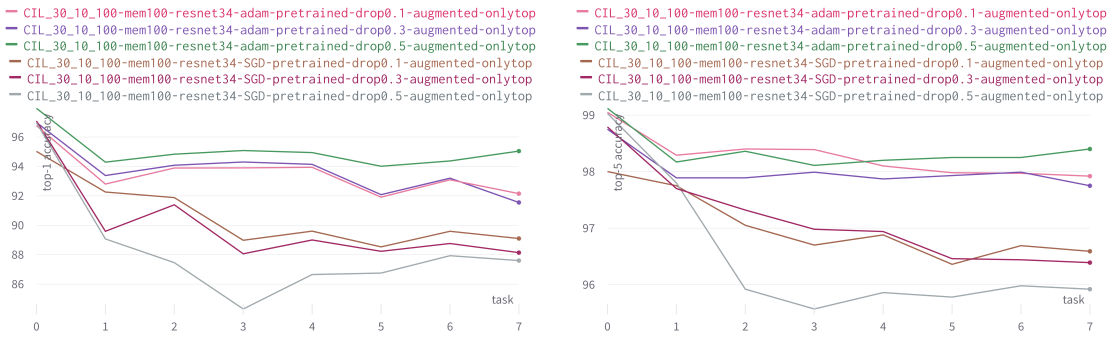
Model name	Dropout rate	Sampling method	Top-1 acc. (%)	Top-5 acc. (%)
drop0.1-augmented	0.1	random	67.17	98.15
drop0.3-augmented	0.3	random	67.28	97.3
drop0.5-augmented	0.5	random	65.57	96.24
drop0.1-augmented-onlytop	0.1	top-100	**89.1**	**96.59**
drop0.3-augmented-onlytop	0.3	top-100	88.14	96.39
drop0.5-augmented-onlytop	0.5	top-100	87.6	95.92

Table 5.5: Comparison between models trained on 100 randomly sampled classes and models trained on the top-100 classes. Top-1 and top-5 accuracy at task 7.

Introduction of Adam optimizer

The next experiments aim to compare SGD (see section 4.2.5) and Adam (see section 4.2.5) optimizers. Using regularization, data augmentation and the top-100 classes, the results of this comparison are shown in Figure 5.6 and Table 5.6.

As we can see, the top-1 accuracy at task 7 improves by more than 6% using Adam as the optimization algorithm. In addition to the performance aspect, the training history in Figure 5.7 shows that Adam leads to faster convergence. In fact, models trained with this algorithm achieve the highest value of top-1 accuracy on the validation set after only a few training epochs.



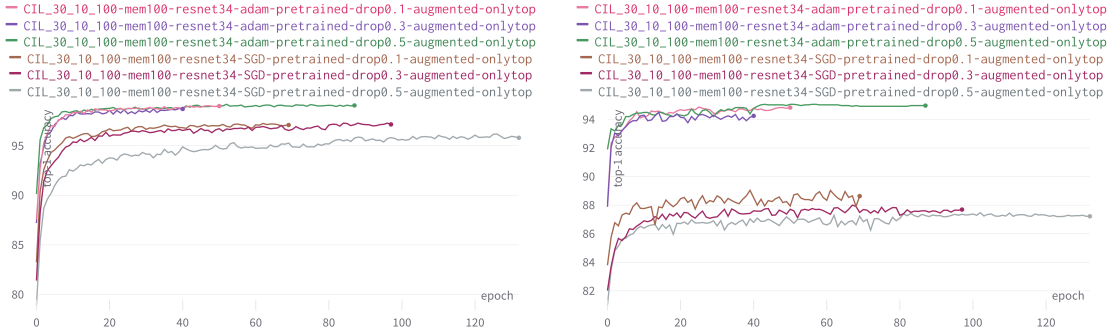
(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.6: Comparison of models trained using SGD and Adam optimizers. Top-1 and top-5 accuracy at each task.

Model name	Dropout rate	Optimizer method	Top-1 acc. (%)	Top-5 acc. (%)
SGD-drop0.1-augmented-onlytop	0.1	SGD	89.1	96.59
SGD-drop0.3-augmented-onlytop	0.3	SGD	88.14	96.39
SGD-drop0.5-augmented-onlytop	0.5	SGD	87.6	95.92
adam-drop0.1-augmented-onlytop	0.1	Adam	92.15	97.92
adam-drop0.3-augmented-onlytop	0.3	Adam	91.55	97.75
adam-drop0.5-augmented-onlytop	0.5	Adam	**95.04**	**98.4**

Table 5.6: Comparison of models trained using SGD and Adam optimizers. Top-1 and Top-5 accuracy at task 7.



(a) Accuracy on the training set

(b) Accuracy on the validation set

Figure 5.7: Comparison of models trained using SGD and Adam optimizers. The images show the training history of models at task 7, comparing the top-1 accuracy at each training epoch evaluated on the training and validation set.

Type 2 baseline

To address the problems relative to the type 1 baseline, the models tested in the previous section are compared with the type 2 baseline described in subsection 4.4.2. This new baseline is trained on the same dataset used for CIL models and consists in fine-tuning ResNet-152 pretrained on ImageNet to the task of logo classification. This new baseline uses data augmentation, a dropout layer before the FC layer and the Adam optimizer.

Considering the top-1 and top-5 accuracy on the test set, the performance of CIL models and type 2 baselines are compared in Table 5.7. As we can see, the performance of the best CIL model is very similar to that of the baseline, but the latter still performs worse, even if only slightly. Although this new baseline alleviates the problem of the number of model parameters, this issue is still present, as shown in Table 5.8, and this is the reason why performance continues to be slightly worse than CIL models.

Model name	Dropout rate	Top-1 acc. (%)
adam-drop0.1-augmented-onlytop	0.1	92.15
adam-drop0.3-augmented-onlytop	0.3	91.55
adam-drop0.5-augmented-onlytop	0.5	**95.04**
BASELINE_2-drop0.1-augmented-onlytop	0.1	82.35
BASELINE_2-drop0.3-augmented-onlytop	0.3	94.11
BASELINE_2-drop0.5-augmented-onlytop	0.5	88.23

Table 5.7: Performance comparison between CIL models and type 2 baselines. Top-1 accuracy calculated on the test set composed of 100 classes.

Model name	#Params (M)
adam-drop0.1-augmented-onlytop	170
adam-drop0.3-augmented-onlytop	170
adam-drop0.5-augmented-onlytop	170
BASELINE_2-drop0.1-augmented-onlytop	60
BASELINE_2-drop0.3-augmented-onlytop	60
BASELINE_2-drop0.5-augmented-onlytop	60

Table 5.8: Comparison between CIL models and type 2 baselines considering the number of model parameters.

Type 3 baseline

The type 3 baseline aims to solve the problem of the number of model parameters. In fact, this approach (described in subsection 4.4.3) defines the baseline architecture using the DER algorithm in the same setup as the CIL classifier, thus having an identical architecture for both the CIL model and the baseline. This ensures that the number of model parameters is the same. The baseline is then trained using data augmentation, regularization, the Adam optimizer and the same dataset.

The performance of the CIL models and this new baseline are compared considering the top-1 accuracy on the test set. As we can see from Table 5.9, the type 3 baseline achieves better performance than the CIL model.

This makes it possible to compare the drop in performance using an incremental learning setup against a standard setup. As shown in Table 5.9, this gap is present, with a 3% drop in top-1 accuracy, but it is expected using an incremental learning setup. However, a 3% drop is acceptable when considering the advantages of an incremental learning approach.

Model name	Dropout rate	Top-1 acc. (%)
adam-drop0.1-augmented-onlytop	0.1	92.15
adam-drop0.3-augmented-onlytop	0.3	91.55
adam-drop0.5-augmented-onlytop	0.5	95.04
BASELINE_3-drop0.1-augmented-onlytop	0.1	**98.52**
BASELINE_3-drop0.3-augmented-onlytop	0.3	97.05
BASELINE_3-drop0.5-augmented-onlytop	0.5	97.64

Table 5.9: Top-1 accuracy of CIL models and type 3 baselines considering all the 100 classes of the test set.

Pruning (100 classes)

An important aspect discussed in subsection 4.2.6 is the number of model parameters. All models tested up to this point add approximately 22 million parameters with each new iteration of incremental learning, thus reaching 170 million parameters at task 7 (22 million for the initial task and 7 iterations of incremental learning).

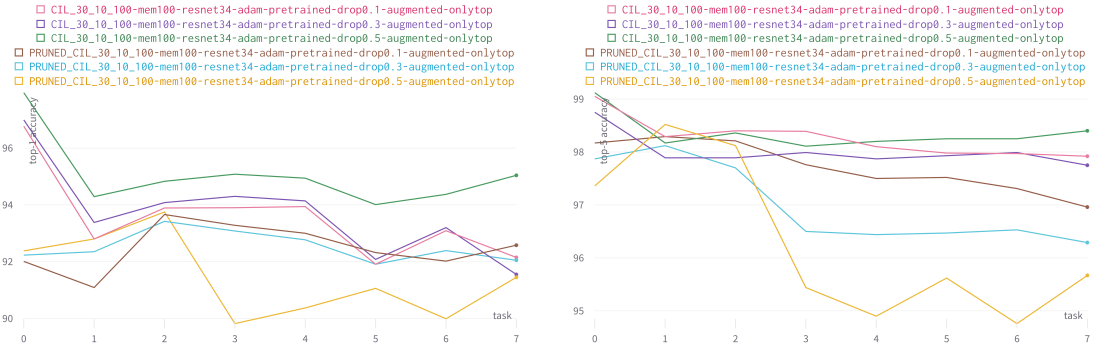
The following experiments are designed to assess the pruning capacity of the channel-level masks and the drop in performance obtained with this method. To this end, the pruned models are compared with those described in the previous section. An important difference in the training procedure of the pruned model consists in monitoring the loss on the validation set instead of the accuracy. This is done because the validation loss decreases with increasing sparsity of the model, and we want to encourage a more sparse model given the same accuracy on the validation set.

The results of the performance comparison are shown in Figure 5.8 and Table 5.10. As we can see, the drop in performance is negligible, with some pruned models performing even better than un-pruned ones. This can be explained by considering pruning as a regularization technique, in fact decreasing the number of parameters effectively reduces the Vapnik-Chervonenkis (VC) dimension [81].

The results of pruning are shown in Table 5.9 and Table 5.11. As we can see, this technique is very effective. The final number of model parameters is reduced by almost a factor of 5x. Note that models which do not use pruning already exceed the total number of parameters of those using pruning from task 1 (43 for the un-pruned models vs. 32 for the pruned ones).

Other interesting insights emerge by analyzing the training history of the models at task 0. The loss function of the DER algorithm (see Equation 2.14) is defined as the sum of: the loss of the classifier, the loss of the auxiliary classifier and loss related to the pruning masks. In Figure 5.10 we can see the classification loss (c), sparsity loss (d) and final loss (b) of the model for each training epoch at task 0. Initially, the classification loss (c) tends

to decrease, but as the training epochs advance, the sparsity loss (d) increasingly masks the channels of the convolutional layers. This leads to a continuous decrease in parameters, but the classification (c) is significantly affected. The deterioration of performance can also be seen in the top-1 accuracy (a) on the validation set.



(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.8: Performance comparison between pruned models and un-pruned ones considering top-1 and top-5 accuracy at each task.

Model name	**Dropout rate**	**Pruning**	**Top-1 acc. (%)**	**Top-5 acc. (%)**
drop0.1-augmented-onlytop	0.1	no	92.15	97.92
drop0.3-augmented-onlytop	0.3	no	91.55	97.75
drop0.5-augmented-onlytop	0.5	no	**95.04**	**98.4**
PRUNED-drop0.1-augmented-onlytop	0.1	yes	92.58	96.96
PRUNED-drop0.3-augmented-onlytop	0.3	yes	92.05	96.29
PRUNED-drop0.5-augmented-onlytop	0.5	yes	91.45	95.67

Table 5.10: Performance comparison between the pruned models and the un-pruned ones. Top-1 and top-5 accuracy at task 7.

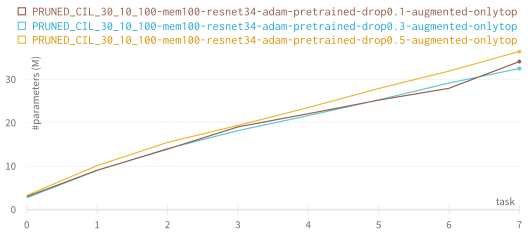
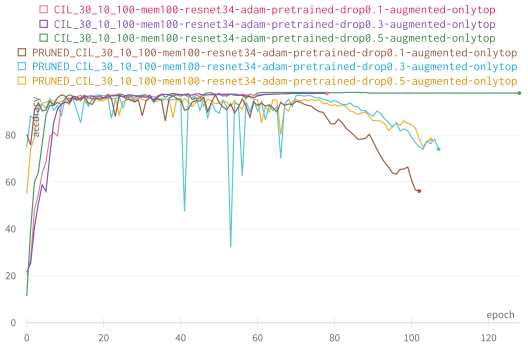


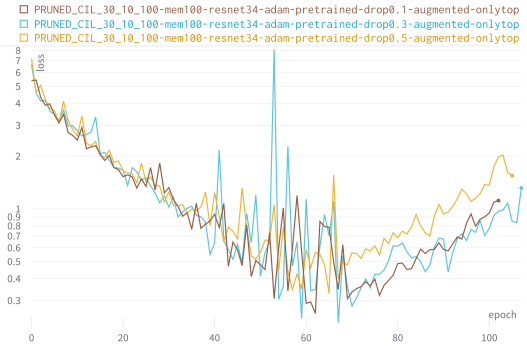
Figure 5.9: Number of model parameters at each task using the pruning technique.

Model name	#Params (M)
UNPRUNED-drop0.1	170
UNPRUNED-drop0.3	170
UNPRUNED-drop0.5	170
PRUNED-drop0.1	34.08
PRUNED-drop0.3	32.48
PRUNED-drop0.5	36.41

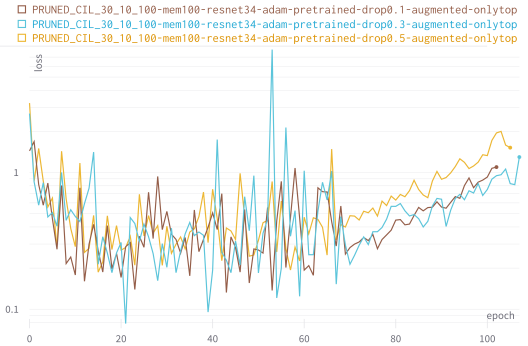
Table 5.11: Number of model parameters at task 7.



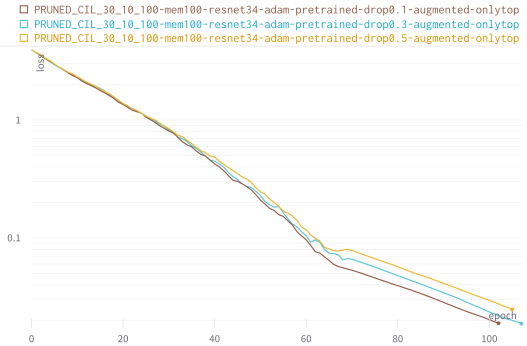
(a) Top-1 accuracy



(b) Final loss



(c) Classification loss



(d) Sparsity loss

Figure 5.10: Training history of the pruned model calculated on the validation set at task 0.

5.2.2 2993 Classes

The following experiments test the scalability capabilities of the model by training it on 2993 classes, i.e. the entire dataset.

Varying the memory size

An important factor for testing the scalability of the model is the number of examples stored for old classes. Indeed, when many classes are introduced, it is necessary to limit the memory used to store old examples. In this first section, models are trained in different configurations to consider various memory sizes used to store the examples of the old classes, in particular 50, 20 and 10 are the memory sizes taken into account.

The models are defined on the basis of the best results previously obtained from experiments on 100 classes: the architecture of each feature extractor is ResNet-34 pre-trained on ImageNet; the optimizer is Adam; regularization with a dropout layer is used as well as data augmentation. No pruning is performed initially.

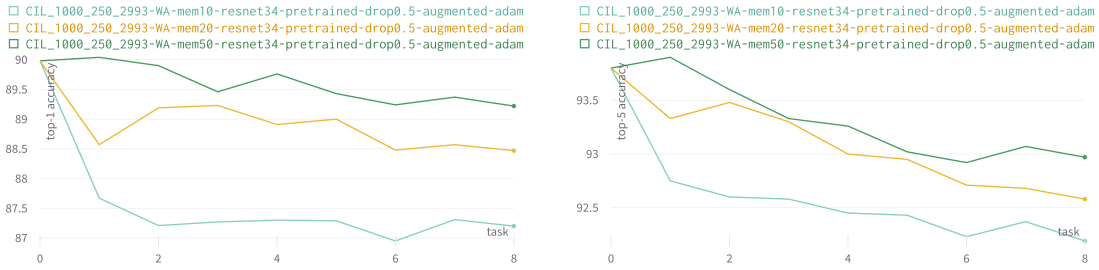
The results of the experiments are shown in Figure 5.11 and Table 5.12. As we can see, the models scale quite well over the entire dataset, by comparison, considering the 100 classes setup shown in Table 5.10, the drop in performance is only about 6%. Moreover, the results show that the greater the number of samples stored for incremental learning iterations, the better the overall accuracy. The total number of examples stored by each model at task 8 is reported in Table 5.13.

Model name	**Dropout rate**	**Mem. size**	**Top-1 acc. (%)**	**Top-5 acc. (%)**
CIL_2993-mem10-drop0.5-adam ♢	0.5	10	87.20	92.19
CIL_2993-mem20-drop0.5-adam	0.5	20	88.47	92.58
CIL_2993-mem50-drop0.5-adam ♣	0.5	50	**89.22**	**92.97**

Table 5.12: Performance of models at each incremental task trained on the entire dataset using different memory sizes. Top-1 and top-5 accuracy at task 8.

Model name	**Mem. per class**	**# Total stored examples (K)**
CIL_2993-mem10-drop0.5-adam	10	29
CIL_2993-mem20-drop0.5-adam	20	53
CIL_2993-mem50-drop0.5-adam	50	102

Table 5.13: Number of total examples stored by each model at task 8.



(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.11: Performance of models at each incremental task trained on the entire dataset using different memory sizes.

Type 3 baseline (2993 classes)

As in the case of 100 classes, the performance of the CIL models is compared with the type 3 baseline (see subsection 4.4.3). For this purpose, the baseline is trained considering the same dataset used to train the models in the previous section. Note that in the case of the baseline, there is no need to use a memory for old classes, as the incremental learning setting does not hold.

The results of the comparison considering the top-1 accuracy between CIL models and baselines are shown in Table 5.14. As we can see, the top-1 accuracy of the best baseline is slightly better than that of the best CIL model, as in the 100 classes setup. The fact that the performance gap is so small proves that the DER algorithm is an effective approach for achieving CIL.

Model name	**Dropout rate**	**Mem. size**	**Top-1 acc. (%)**	**Top-5 acc. (%)**
CIL_2993-mem10-drop0.5-adam	0.5	10	87.20	92.19
CIL_2993-mem20-drop0.5-adam	0.5	20	88.47	92.58
CIL_2993-mem50-drop0.5-adam	0.5	50	89.22	92.97
BASELINE_3-2993classes-drop0.1	0.1	-	89.77	93.98
BASELINE_3-2993classes-drop0.3	0.3	-	88.41	93.38
BASELINE_3-2993classes-drop0.5	0.5	-	**90.22**	**94.36**

Table 5.14: Top-1 and top-5 accuracy of the CIL model and the type 3 baseline considering all 2993 classes of the test set.

Pruning (2993 classes)

Similar to the setup with 100 classes, the pruning strategy is tested for models trained on the entire dataset. As we can see from Figure 5.12 and Table 5.15, when pruning is used in

combination with Weight Aligning (WA), performance drops dramatically. Therefore, only for models using pruning, WA is disabled.

In contrast to the 100-class experiments, pruning now yields poor performance. In fact, the drop in accuracy is considerable compared to models that do not use pruning. Even considering the case where 50 samples are used for the pruned model, performance deteriorates by almost 20%.

Regarding the number of parameters, even in the setup with 2993 classes, the model is significantly reduced in size. However, as we can see from Table 5.16, the pruning factor corresponds to approximately 3x compared to the 5x achieved in the previous setup.

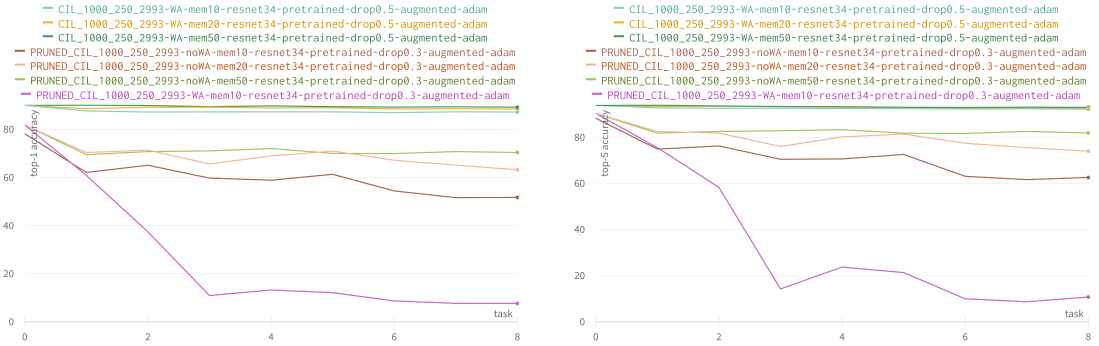
In conclusion, the pruning strategy is not effective in the case of models trained on the entire dataset. For this reason, KD is used instead. The results of the experiments are shown in section 5.3.

Model name	**Mem. size**	**WA**	**Pruning**	**Top-1 acc. (%)**	**Top-5 acc. (%)**
UNPRUNED-WA-mem10-drop0.5	10	yes	no	87.2	92.19
UNPRUNED-WA-mem20-drop0.5	20	yes	no	88.47	92.58
UNPRUNED-WA-mem50-drop0.5	50	yes	no	**89.22**	**92.97**
PRUNED-noWA-mem10-drop0.3	10	no	yes	51.69	62.56
PRUNED-noWA-mem20-drop0.3	20	no	yes	63.2	73.97
PRUNED-noWA-mem50-drop0.3 ♠	50	no	yes	70.37	81.85
PRUNED-WA-mem10-drop0.3	50	yes	yes	7.6	10.67

Table 5.15: Performance comparison between the pruned and un-pruned models. Top-1 and top-5 accuracy at task 8.

Model name	**#Params (M)**
UNPRUNED-CIL_2993-WA-mem10-drop0.5	205
UNPRUNED-CIL_2993-WA-mem20-drop0.5	205
UNPRUNED-CIL_2993-WA-mem50-drop0.5	205
PRUNED-CIL_2993-noWA-mem10-drop0.3	68.79
PRUNED-CIL_2993-noWA-mem20-drop0.3	71.66
PRUNED-CIL_2993-noWA-mem50-drop0.3	68.33
PRUNED-CIL_2993-WA-mem10-drop0.3	70.69

Table 5.16: Number of model parameters at task 8.



(a) Top-1 accuracy

(b) Top-5 accuracy

Figure 5.12: Performance comparison between pruned and un-pruned models trained on the entire dataset considering the top-1 accuracy at each incremental task.

5.3 Knowledge Distillation

To overcome the drastic drop in accuracy when using pruning on models trained on the entire dataset (see Table 5.15), KD is adopted as described in section 4.3. During KD, an un-pruned model is used as the teacher, while the student is a significantly smaller CNN. In the experiments presented in this section, ResNet-50 pretrained on ImageNet is used as the backbone for the student, with a dropout layer before the FC layer.

KD is performed after the training of the teacher model. Since the CIL classifier is trained using the DER algorithm and it requires the storage of a part of the examples of the old classes, in a real situation KD can only be performed with those examples available as a result of the storage. Therefore, in order to replicate a realistic situation, the student model is trained under the supervision of the teacher, but only the samples stored by the teacher are used as training data.

The results of KD experiments are shown in Table 5.17, where different models with varying memory sizes are used as teachers. As we can see, the number of samples used to train the student heavily affects the final performance.

Using 50 samples per class to train the student results in top-1 accuracy of 88.96%, i.e. a decrease of less than 1% compared to the teacher model. It is important to consider that all teacher models have 205 million parameters, while the final student models have 30 million parameters (the number of ResNet-50 parameters). Therefore, the number of parameters is reduced by almost 7 times, with nearly no change in performance.

KD outperforms the pruning method (see Table 5.15 and Table 5.16), increasing top-1 accuracy from 70.37% to 88.96% and the compression factor from 3x to 7x.

Teacher model	Mem. size	Student drop. rate	Top-1 acc. (%)	Top-5 acc. (%)
UNPRUNED-WA-mem10-drop0.5	10	0.1	70.53	79.83
UNPRUNED-WA-mem10-drop0.5	10	0.3	73.52	82.49
UNPRUNED-WA-mem50-drop0.5 ♡	50	0.3	**88.96**	**93.57**

Table 5.17: Top-1 accuracy of the students trained using KD. The backbone of each student model is ResNet-50 pretrained on ImageNet.

5.4 Logo detector

This section describes experiments designed to evaluate the class-agnostic logo detector introduced in section 4.1. The CIL classifier is trained starting from the cropped regions, while the detector predicts the bounding boxes using the entire image as input. Since an image can contain several logos, it is necessary to train the detector using the same training, validation and test sets used by the CIL classifier. This is done by considering training labels for the detector only if the bounding box corresponds to a ROI used as a training example for the CIL classifier, otherwise the bounding box is used in the validation or test set, depending on the split used for the classifier. This process is shown in Figure 5.13. In this way, an unbiased evaluation of both the detector and the CIL classifier is achieved.

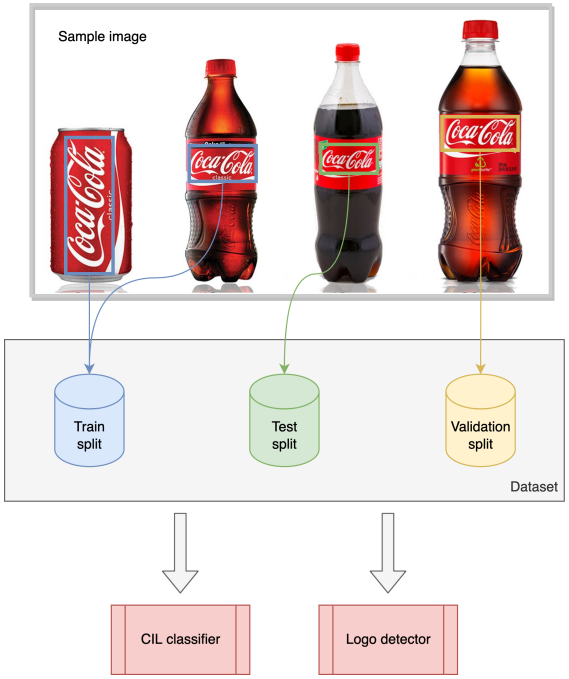


Figure 5.13: Training, validation and test sets used for the detector built on the splits used to train the CIL classifier.

5.4.1 Metrics

Mean Average Precision (mAP) is a metric used to evaluate object detection models. This metric is based on: Intersection over Union (IoU), Recall and Precision.

Intersection over Union

Given a ground-truth bounding box box_{gt} and a detected bounding box box_{pred}, as shown in Figure 5.14, the IoU is computed as the ratio of the overlap and union areas:

$$\text{IoU} = \frac{box_{gt} \cap box_{pred}}{box_{gt} \cup box_{pred}} \tag{5.1}$$

A prediction is considered correct if IoU $\geq \tau$, where τ is a threshold (a typical value for τ is 0.5).

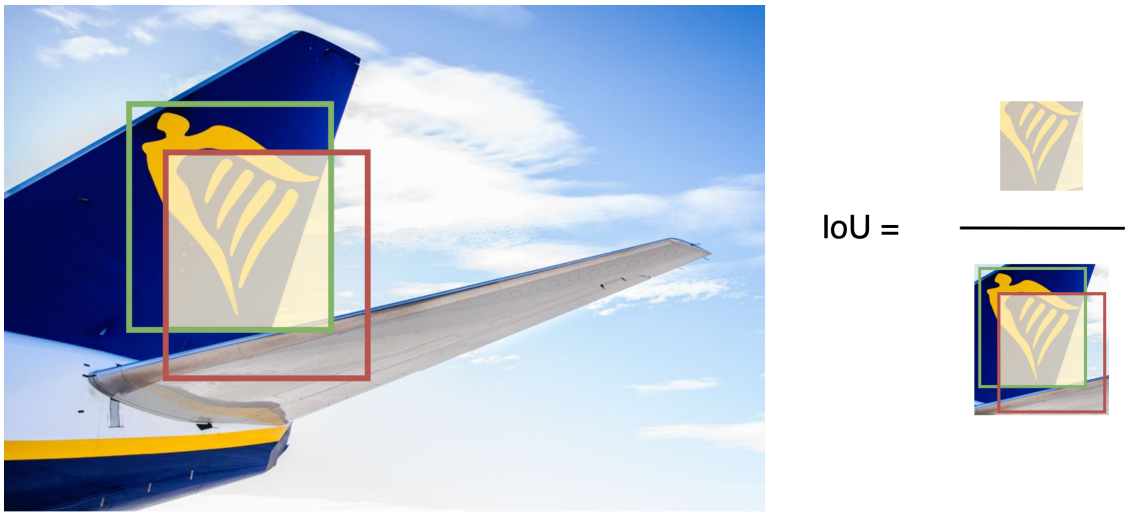


Figure 5.14: Intersection over Union (IoU). The green box represents the ground-truth bounding box, the red one represents the predicted bounding box and the intersection is given by the yellow region. Then, the IoU is computed as the ratio of the overlap and union areas.

Precision and Recall

Given the IoU and a threshold τ it is possible to count the number of:

- True Positive (**TP**): the number of bounding boxes which have a IoU $\geq \tau$ with the ground-truth bounding box.
- False Positive (**FP**): the number of bounding boxes that predict an object which is not actually an object.

- False Negative (**FN**): the number of bounding boxes which have a IoU $< \tau$ with the ground-truth bounding box.

Note that the True Negative (TN) are not considered in an object detection task, as this would mean counting as TN the background of the image that contains no object.

Using TP, FP, FN, the Precision P is given by:

$$P = \frac{TP}{TP + FP} \tag{5.2}$$

and the Recall R is given by:

$$R = \frac{TP}{TP + FN} \tag{5.3}$$

Intuitively, precision is the ability of the detector not to identify as an object a part of the image that actually is not, and recall is the ability of the detector to find all objects in the image. The threshold τ affects the number of TP, FP, FN and thus the value of precision and recall. For this reason, given a value for τ (e.g. $\tau = 0.5$), computing the precision or recall at $\tau = 0.5$ is denoted by $P@.5$ and $R@.5$ respectively.

Mean Average Precision

Average Precision (AP) is calculated as the weighted average of precisions at each threshold, the weight is the increase in recall from the prior threshold. For values of $\tau \in T$, where $T = [0.50, 0.55, 0.60, ..., 0.95]$, the AP computed using the thresholds in T and denoted by AP@.5:.95 is given by:

$$AP = \sum_{i=1}^{|T|-1} (R@\tau_i - R@\tau_{i+1}) * P@\tau_i \tag{5.4}$$

Finally, given k different classes, the mAP is the average of the APs among each class:

$$mAP = \frac{1}{k} \sum_{i=1}^{k} AP_i \tag{5.5}$$

5.4.2 100 Classes

As described in chapter 4, the class-agnostic logo detector represents the first stage of the system and works in combination with the CIL model to achieve incremental learning logo detection and recognition.

Training using the 30 classes of the initial task

As with the CIL classifier, the first experiments with the logo detector focus on the top-100 classes of the dataset (where top-100 refers to the 100 classes with the highest number of

samples, see section 5.2.1). Given the nature of the problem, in a real scenario, the detector is trained using only those classes available for the CIL model at task 0. In the CIL setup described in section 5.1, the initial task consists of 30 classes. For this reason, in the top-100 classes setup, the detector is trained for 30 epochs with the SGD optimizer using only those 30 classes available at task 0.

During the training of the detector, the object loss, box loss (see subsection 2.1.1 and Equation 2.1) and the mAP@.5 are calculated at each epoch on the validation set. The performance shown in Figure 5.15 considers only the 30 classes used for training, which is useful for analyzing the training phase, but given the incremental learning setup, we expect the class-agnostic logo detector to perform well on all the 100 classes. For this purpose, the detector is evaluated considering the mAP@.5 on the test set combining the initial 30 classes and the remaining 70. This evaluation represents a real-world scenario and is therefore the most reliable and most suitable for evaluating the detector. The results of the evaluation are shown in Table 5.18.

Generalization capabilities of the class-agnostic logo detector

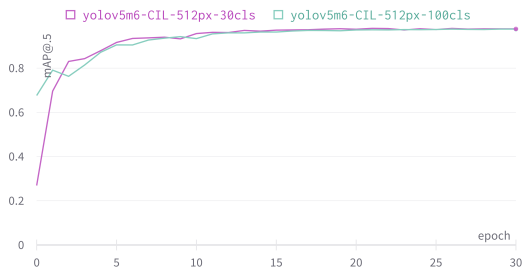
To better study the generalization capabilities of the detector trained on 30 classes to the remaining 70, the detector is compared with a new one trained on all the 100 classes. This detector can be considered as a baseline with respect to the generalization capabilities of the detector trained on 30 classes: the smaller the gap between the performance of these two models, the better the generalization capabilities. This new detector is trained for 30 epochs using the 100 classes and, as before, object loss, box loss and mAP@.5 are monitored on the validation set. The comparison between the two detectors is shown in Figure 5.15.

As we can see from Figure 5.15, the performance of detectors trained on 30 and 100 classes is quite similar. However, a crucial aspect of this training results is that these metrics consider all and only those classes on which the model is trained on, thus 30 classes and 100 classes respectively. The fact that the detectors achieve the same performance is a sign that they accomplish similar results in detecting logos across a different number of classes. However, the real comparison consists in the mAP@0.5 computed on the test set of all the 100 classes. As shown in Table 5.18, the detector trained on 30 classes does not generalize well on the remaining 70. In fact, considering this detector, the mAP@.5 obtained on the validation set at the last training epochs is 0.97 (shown in Figure 5.15), while the mAP@.5 computed on the test set composed of all 100 classes is 0.75.

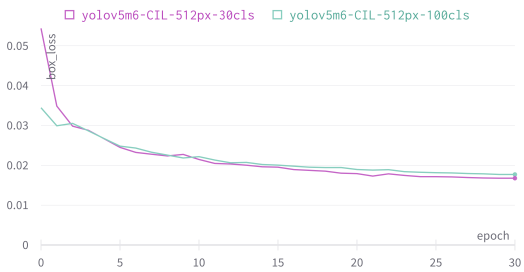
In contrast, the detector trained on all the 100 classes, obtains a mAP@.5 on the test set of 0.982, which is consistent with the performance obtained on the validation set at the last training epochs. However, it must be considered that, since this detector is trained using 70 more classes, it has more training samples at its disposal, hence more data from which to acquire knowledge of a logo.

Model	Precision	Recall	mAP@.5	mAP@.5:.95
DET_class-agnostic_30cls	0.784	0.659	0.75	0.542
DET_class-agnostic_100cls	0.944	0.972	0.982	0.776

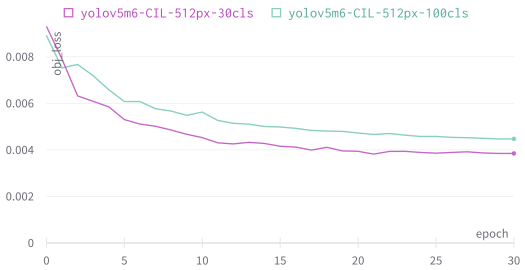
Table 5.18: Precision, Recall, mAP@.5 and mAP@.5:.95 obtained by the detector trained on 30 classes (DET_class-agnostic_30cls) and the one trained on 100 classes (DET_class-agnostic_10cls). The performance refers to the test set composed of all the 100 classes.



(a) mAP@.5



(b) Box loss



(c) Object loss

Figure 5.15: Performance comparison between the class-agnostic logo detector trained on 30 classes and the one trained on 100 classes. The plots show the mAP@.5, box loss and object loss on the validation set at each training epoch. Note that these performance are computed on the validation set of each model, thus using 30 and 100 classes respectively.

5.4.3 2993 Classes

The experiments conducted in the previous section are repeated considering the 2993-class setup (see section 5.1), which consists of 1000 classes used at the initial task and 250 classes added at each incremental learning task. This section evaluates the detector's ability to scale from 100 classes to 2993 classes.

The detector is trained for 30 epochs using the SGD optimizers on the first 1000 classes of the initial task. Then the evaluation metrics are computed on the test set of these 1000 classes combined with the remaining 1993 classes, thus on the entire dataset consisting of

2993 classes in total.

During the training, object loss, box loss and mAP@.5 are monitored on the validation set consisting of the 1000 classes used for training. As we can see in Figure 5.16, the mAP@.5 of the model computed on the validation set is lower than the 30-class or 100-class setup. This is because recognizing 1000 different logos is a more difficult task than recognizing 30 or 100 different logos.

Apart from the performance on the validation set obtained during training, the most useful insights are obtained by evaluating the detector on the test set composed of all 2993 classes. The results are shown in Table 5.19. As we can seen, the mAP@.5 is very similar to the 30-class setup (0.706 vs 0.75).

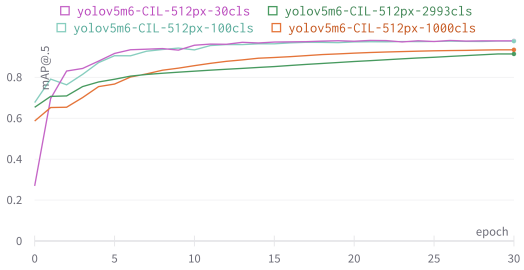
As with the 100-class setup, another important comparison is between the detector trained on 1000 classes and the one trained on all 2993 classes, in order to assess the generalization capabilities of the detector trained on 1000 classes when tested on all classes. The training history of the detector trained on all 2993 classes is shown in Figure 5.16 and the final performance obtained by testing it on the entire dataset is shown in Table 5.19.

From the results shown in Table 5.18 we can see that the gap in mAP@.5 between the detector trained on 30 classes and the one trained on 100 classes is quite large. As shown in Table 5.19, this gap in mAP@.5 is still present in the 2993-class setup, but it is slightly reduced: 0.706 in the 1000-class setup vs. 0.890 in the 2993-class setup. Note that, unlike the setup with 30 and 100 classes, although the difference in performance remains large between the two detectors, in this setup the 1000-class detector is required to generalize to a wide range of new logos (1993 new classes). Furthermore, as in the previous setup, since the 2993-class detector is trained using 1993 more classes, it leverages more training data. In conclusion, these results reveal that even if the detector is only trained on 1000 classes, it is able to effectively detect the remaining 1993 classes (i.e. generalize).

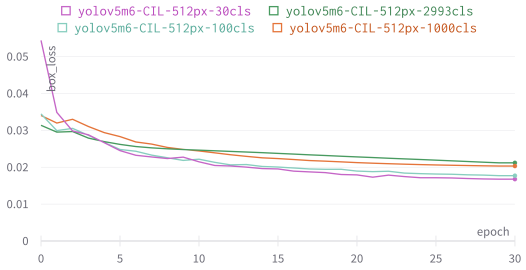
This is a fundamental result for the entire system, as it is evidence in support of the claim made in chapter 1, where it is assumed that it is not necessary to also develop an incremental learning detector, since the idea of what a generic logo is can be learned and well approximated using only an initial set of logos. The subsequent creation of new logos will not disrupt the general idea of a logo.

Model	**Precision**	**Recall**	**mAP@.5**	**mAP@.5:.95**
DET_class-agnostic_1000cls	0.704	0.666	0.706	0.465
DET_class-agnostic_2993cls	0.833	0.846	0.890	0.648

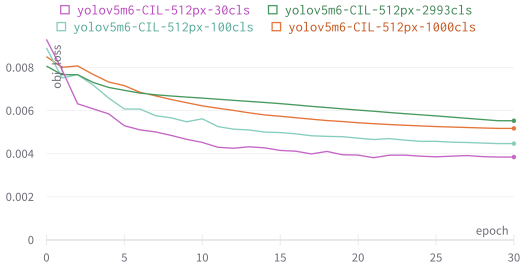
Table 5.19: Precision, Recall, mAP@.5 and mAP@.5:.95 obtained by the detector trained on 1000 classes (DET_class-agnostic_1000cls) and that trained on 2993 classes (DET_class-agnostic_2993cls). The performance refers to the test set composed of all the 2993 classes.



(a) mAP@.5



(b) Box loss



(c) Object loss

Figure 5.16: Performance comparison between the class-agnostic logo detector trained on 1000 classes and the one trained on 2993 classes. The plots show mAP@.5, box loss and object loss on the validation set at each training epoch. Note that these performance are computed on the validation set of each model, thus using 1000 and 2993 classes respectively.

5.5 Full recognition

In this last section, the system is tested as a whole. This consists in combining the first and second stage (see chapter 4): the class agnostic logo detection that generates ROIs, and the actual classification performed by the CIL classifier. Then, the final performance is evaluated considering the precision, recall, mAP@.5 and mAP@.5:.95 on the test set composed of all the 2993 classes.

The model used for the first stage of the system is the class agnostic logo detector presented in subsection 5.4.3, which is trained using only the 1000 classes used at task 0 by the CIL classifier.

With regard to the CIL classifier, the following models are compared:

1. CIL classifier with memory 50 (♣ in Table 5.12).
2. CIL classifier with memory 10 (♢ in Table 5.12).
3. CIL classifier with memory 50 pruned using learnable masks (♠ in Table 5.15).

4. Student model trained with KD by the CIL classifier with memory 50 (♡ in Table 5.17).

The results of this evaluation are shown in Table 5.20. As a comparison, the results obtained in the literature by Li et al. in SeeTek [40] and by Wang et al. in LogoDet-3K [83] (see section 2.2) are reported in Table 5.21. The proposed approach using KD outperforms Logo-Yolo introduced in [83] but achieves lower performance than SeeTek. Still, it must be considered that SeeTek is an open-set retrieval method, thus not an incremental learning approach. In an open-set retrieval approach, the classification of an input logo is performed by assigning the class of the nearest logo in a latent space. By doing so, if an input logo is similar, but does not actually match the training images, it is still recognized, but the system will not be able to integrate new knowledge to recognize new logos.

As we can see from Table 5.21 and Table 5.19, the end-to-end performance obtained by SeeTek (70.46% mAP@.5) is very similar to that obtained by the proposed class agnostic logo detector ignoring the classification stage performed by the CIL model (70.60% mAP@.5). This is a clear indication that the detector is a bottleneck for the final performance. To tackle this problem, a yolo-based detector with a larger number of parameters could be used (e.g. YOLOv5x6 in Table 4.1) or a detector based on transformer architecture could be tested (as proposed by Carion et al. in [9]).

CIL classifier	Precision	Recall	mAP@.5	mAP@.5:.95
CIL_classifier-mem50 ♣	0.578	0.615	0.614	0.417
CIL_classifier-mem10 ♢	0.545	0.591	0.583	0.397
Pruned_CIL_classifier-mem_50 ♠	0.496	0.485	0.489	0.347
KD_classifier-mem_50 ♡	0.558	0.604	0.59	0.403

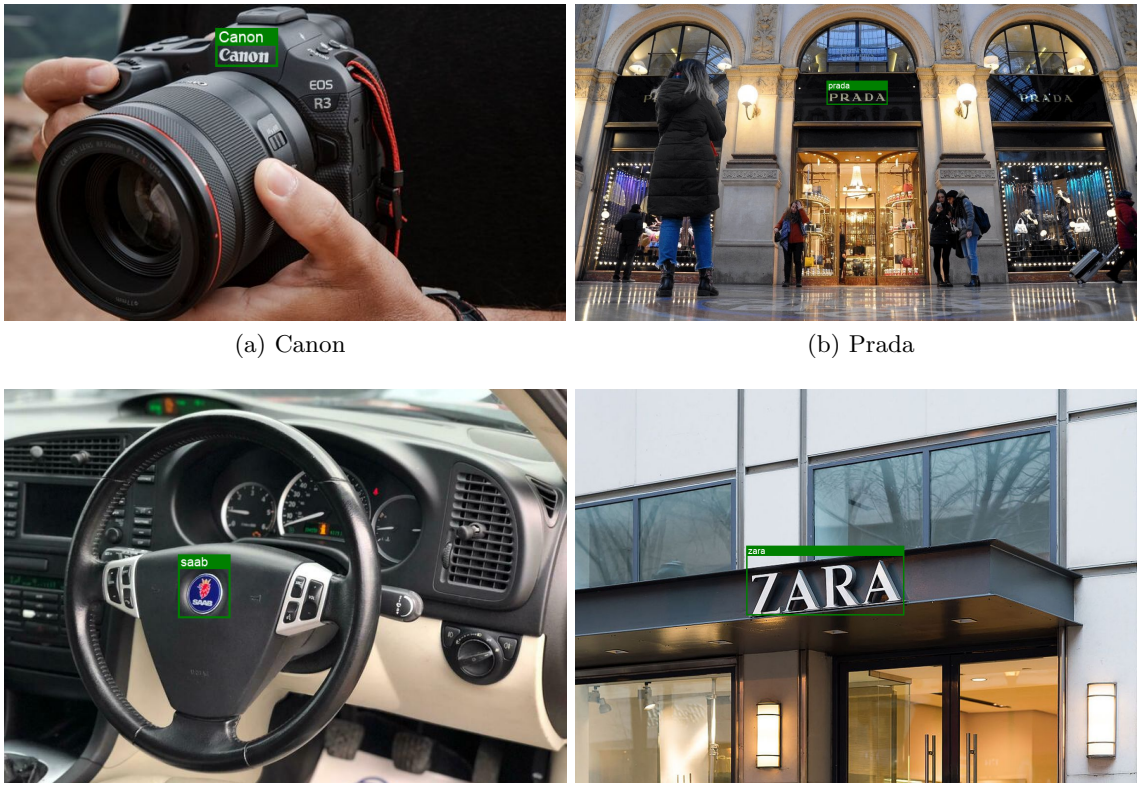
Table 5.20: Full recognition performance of the system using the class-agnostic logo detector and different CIL classifiers. The Precision, Recall, mAP@.5 and mAP@.5:.95 are computed on the test set composed of all the 2993 classes.

Model	mAP@.5 (%)
SeeTek [40]	**70.46**
LogoDet-3K [83]	52.28
Proposed model w/o KD ♣	61.40
Proposed model with KD ♡	59.00

Table 5.21: Full recognition performance comparison on LogoDet-3K comparing two SOTA approaches proposed in the literature.

To evaluate how the system behaves on real images, it is tested on some images in the wild, the results of the detection and the classification are shown in Figure 5.17. As we can

see from the example (d) in Figure 5.17, the system has some difficulties when the logo is partially occluded, as in the case of the reflection in this example.



(a) Canon

(b) Prada

(c) Saab

(d) Zara

Figure 5.17: System tested on images in the wild considering different brands.

To further investigate the mispredictions of the CIL classifier, the Confusion Matrix (CM) is analyzed to find out whether one class is often mistaken for another. The CM is an $n \times n$ matrix, where $n = 2993$ is the number of classes. By definition, CM_{ij} is equal to the number of observations belonging to class i and predicted as class j.

Since simply plotting the CM does not lead to useful insights (due to the large number of entries in the matrix i.e. 2993×2993), the CM is analyzed by counting the number of entries $CM_{ij} \geq k$ such that $i \neq j$, where k is a threshold. Then, the number of entries $CM_{ij} \geq k$ as a function of k is shown in Figure 5.18. As we can see, with $k = 10$ (i.e. at least 10 times a class i is predicted as class j), the number of entries $CM_{ij} \geq 10$ is only 7, note that $CM_{ij} < 40$ for each $i \neq j$.

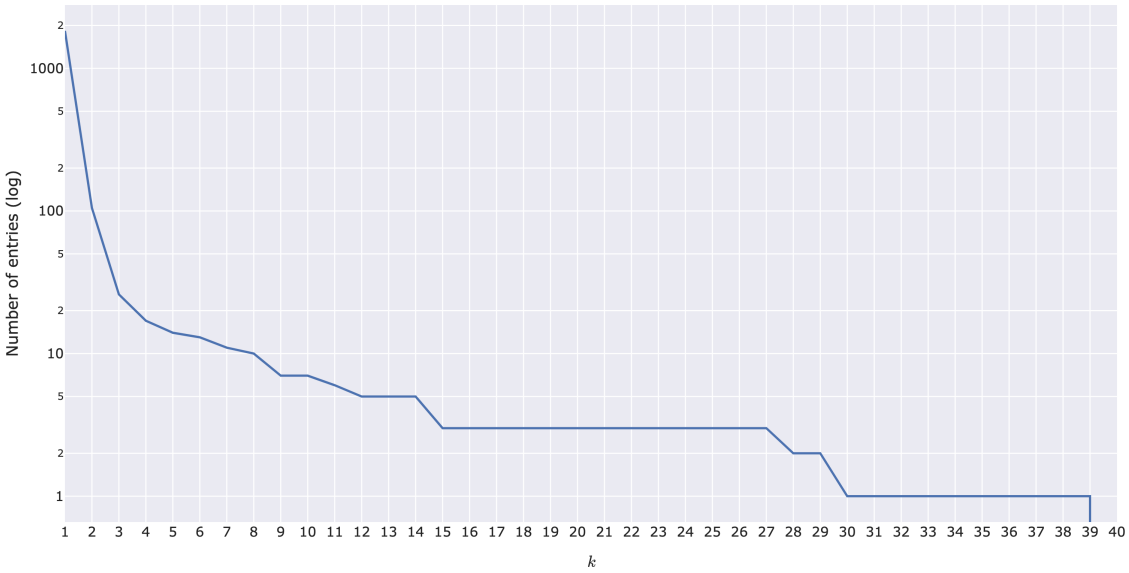


Figure 5.18: Number of entries in the matrix such that $CM_{ij} \geq k$ for each $i \neq j$.

The 7 entries (i.e. pairs of classes) in the confusion matrix obtained with $k = 10$ are the following:

- ("alpinestars-2", "alpinestars-1")
- ("atari 2600-2", "atari 2600-1")
- ("auxx-2", "auxx-1")
- ("iPhone-1", "Apple")
- ("Auntie Anne"s-2", "Auntie Anne"s-1")
- ("maggi noodles", "Maggi")
- ("bauschlomb-2", "bauschlomb-1")
- ("BEST EXPRESS-2", "BEST EXPRESS-1")
- ("BEST EXPRESS-3", "BEST EXPRESS-1")
- ("avery dennison-2", "avery dennison-1")
- ("ANCAP-2", "ANCAP-1")

where, for each tuple, the first element corresponds to the ground truth and the second to the model prediction. From this result we can see that one class is predicted as another class that actually belongs to the same brand. In fact, although the labels are different,

the images corresponding to each pair of classes are indeed the same. This result is due to annotation errors in the LogoDet-3K dataset, as described in section 3.2.

Even though these types of results are obtained for each value of $k \geq 4$, interesting examples of similar classes can be found by analyzing the entries of the CM with $k = 3$. For example, for 3 times, the class "Cocoa Puffs" is predicted as "Cocoa Krispies" and, as we can see from Figure 5.19, these two classes are indeed similar.

(a) Cocoa Krispies (b) Cocoa Puffs

Figure 5.19: Example of similar classes. The image shows an instance of "Cocoa Puffs" on the left and "Cocoa Krispies" on the right.

Chapter 6

Conclusions and future works

This thesis addressed the task of logo recognition by reformulating the problem in a class incremental learning approach. The motivation behind the use of this technique is that new logos are continuously being created. Consequently, an effective model should be able to keep up with the introduction of new logos in order to recognize them, without the need to retrain a model but rather by integrating old knowledge with new one.

To this end, a two-stage logo detector is used to solve the problem, which consists of a first stage relating to the localization of logos in the image and a second stage regarding the actual classification of logos.

The first stage consists in a class-agnostic logo detector (i.e. the logo detector identifies any type of logo, without recognizing the specific class), in this way it is possible to decouple localization from recognition, thus avoiding the need to create a Class Incremental Learning (CIL) logo detector. This approach proves to be successful, as a generic logo detector is able to learn what a logo is by using only an initial set of logos. The logo detector is then capable of generalizing and detect previously unseen logos.

On the other hand, taking advantage of state-of-the-art incremental learning techniques, a CIL classifier is defined to accomplish the actual classification of logos, using the Regions of Interest (ROIs) provided by the agnostic logo detector. Experiments show that a classifier trained in this fashion, storing 50 examples of old classes, performs well without significant drawbacks in terms of performance compared to a standard approach (according to the results of the baseline comparison).

A downside of the algorithm used for CIL is the number of model parameters. The method implemented with learnable masks used to perform channel-level pruning of the Convolutional Neural Network (CNN) layers achieves promising performance when tested on a subset of the dataset. However, this approach performs poorly when trying to scale to the entire dataset. To solve this problem, Knowledge Distillation is used as an alternative approach to channel-level masks. In this way, it is possible to train a significantly smaller

CNN with the supervision of a larger model resulting from the incremental learning steps. Note that, in the experiments conducted in this thesis, the Knowledge Distillation (KD) is applied at the end of all incremental learning tasks, however the Dynamically Expandable Representation (DER) algorithm [86] is flexible enough to be used in combination with KD. In the case where there is a need to recognize new logos after the KD (given the incremental learning nature of the problem), it would be possible to exploit the student architecture resulting from the KD, instead of the initial teacher model. It would be sufficient to freeze the student weights trained with the KD and then expand the architecture with the DER algorithm for subsequent incremental learning iterations, like any other type of feature extractor in the standard DER configuration. However, this is true from a theoretical point of view and further experiments should be conducted to verify the effectiveness of this technique and evaluate its performance.

The final performance obtained by the proposed approach is tested on the LogoDet-3K dataset. The results show how, considering the class-agnostic logo detector trained on 1000 classes and the CIL classifier trained using KD, the proposed approach outperforms the end-to-end model proposed by the authors of LogoDet-3K [83]. On the other hand, the model proposed by Amazon in SeeTek [40] performs better on the LogoDet-3K dataset. However, it should be taken into account that SeeTek considers an open-set retrieval environment as opposed to an incremental learning setup considered in this thesis. Furthermore, from the final results obtained by the proposed system emerge that the class-agnostic logo detector acts as a bottleneck for the whole system, thus severely limiting the final performance.

Future improvements of this work could tackle the problem related to the class-agnostic logo detector by considering a version of the YOLO detector [1] with a higher number of parameters and a larger input image size. However, this would lead to higher computational requirements to train the model as well as longer training and inference times. Another way to address this problem is to experiment with different architectures for the detector. Additional improvements of the system are related to the second stage of the pipeline performed by the CIL classifier. The classification task could be improved by enriching the representation of each logo by considering the textual component in addition to the visual feature. This approach adopted in the literature is very effective, since many logos have a text component that can be used to distinguish them. Further studies can be conducted in the direction of explainability, which is a hot topic in artificial intelligence when using deep learning models. This research could investigate what the detector considers as logos and explain why such detection applies to a given input.

Bibliography

[1] Glenn Jocher et. al. *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*. Version v6.1. Feb. 2022. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936.

[2] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. "Expert gate: Lifelong learning with a network of experts". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3366–3375.

[3] Rahaf Aljundi et al. "Memory aware synapses: Learning what (not) to forget". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 139–154.

[4] Rahaf Aljundi et al. "Online continual learning with no task boundaries". In: *arXiv preprint arXiv:1903.08671* (2019).

[5] C Atkinson et al. "Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. arxiv 2018". In: *arXiv preprint arXiv:1802.03875* 2 (1802).

[6] Muhammet Bastan et al. "Large scale open-set deep logo detection". In: *arXiv preprint arXiv:1911.07440* (2019).

[7] Simone Bianco et al. "Deep learning for logo recognition". In: *Neurocomputing* 245 (2017), pp. 23–30.

[8] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection". In: *arXiv preprint arXiv:2004.10934* (2020).

[9] Nicolas Carion et al. "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer. 2020, pp. 213–229.

[10] Arslan Chaudhry et al. "Continual learning with tiny episodic memories". In: (2019).

[11] Arslan Chaudhry et al. "Efficient lifelong learning with a-gem". In: *arXiv preprint arXiv:1812.00420* (2018).

[12] Arslan Chaudhry et al. "Riemannian walk for incremental learning: Understanding forgetting and intransigence". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 532–547.

[13] Zhi-Qi Cheng et al. "Video eCommerce++: Toward large scale online video advertising". In: *IEEE transactions on multimedia* 19.6 (2017), pp. 1170–1183.

[14] Matthias De Lange and Tinne Tuytelaars. "Continual prototype evolution: Learning online from non-stationary data streams". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8250–8259.

[15] Matthias Delange et al. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[16] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[17] David S Doermann, Ehud Rivlin, and Isaac Weiss. "Logo recognition using geometric invariants". In: *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93)*. IEEE. 1993, pp. 894–897.

[18] Arthur Douillard et al. "Podnet: Pooled outputs distillation for small-tasks incremental learning". In: *European Conference on Computer Vision*. Springer. 2020, pp. 86–102.

[19] István Fehérvári and Srikar Appalaraju. "Scalable logo recognition using proxies". In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2019, pp. 715–725.

[20] Chrisantha Fernando et al. "Pathnet: Evolution channels gradient descent in super neural networks". In: *arXiv preprint arXiv:1701.08734* (2017).

[21] Robert M French. "Catastrophic forgetting in connectionist networks". In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[22] Yue Gao et al. "Filtering of Brand-Related Microblogs Using Social-Smooth Multiview Embedding". In: *IEEE Transactions on Multimedia* 18.10 (2016), pp. 2115–2126. DOI: 10.1109/TMM.2016.2581483.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[24] Jianping Gou et al. "Knowledge distillation: A survey". In: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819.

[25] Stephen Grossberg. "Adaptive Resonance Theory: How a brain learns to consciously attend, learn, and recognize a changing world". In: *Neural networks* 37 (2013), pp. 1–47.

[26] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[27] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012), p. 2.

[28] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[29] Steven CH Hoi et al. "Logo-net: Large-scale deep logo detection and brand recognition with deep region-based convolutional networks". In: *arXiv preprint arXiv:1511.02462* (2015).

[30] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[31] David Isele and Akansel Cosgun. "Selective experience replay for lifelong learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[32] Xuan Jin et al. "The Open Brands Dataset: Unified brand detection and recognition at scale". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 4387–4391.

[33] Alexis Joly and Olivier Buisson. "Logo retrieval with a contrario visual query expansion". In: *Proceedings of the 17th ACM international conference on Multimedia*. 2009, pp. 581–584.

[34] Heechul Jung et al. "Less-forgetting learning in deep neural networks". In: *arXiv preprint arXiv:1607.00122* (2016).

[35] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[36] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[37] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[38] Frantzeska Lavda et al. "Continual classification learning using generative models". In: *arXiv preprint arXiv:1810.10612* (2018).

[39] Sang-Woo Lee et al. "Overcoming catastrophic forgetting by incremental moment matching". In: *Advances in neural information processing systems* 30 (2017).

[40] Chenge Li et al. "SeeTek: Very Large-Scale Open-set Logo Recognition with Text-Aware Metric Learning". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, pp. 2544–2553.

[41] Lufan Li et al. "An incremental face recognition system based on deep learning". In: *2017 Fifteenth IAPR international conference on machine vision applications (MVA)*. IEEE. 2017, pp. 238–241.

[42] Zhizhong Li and Derek Hoiem. "Learning without forgetting". In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[43] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[44] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[45] Li Liu et al. "Deep learning for generic object detection: A survey". In: *International journal of computer vision* 128.2 (2020), pp. 261–318.

[46] Xialei Liu et al. "Rotate your networks: Better weight consolidation and less catastrophic forgetting". In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE. 2018, pp. 2262–2268.

[47] Yaoyao Liu, Bernt Schiele, and Qianru Sun. "Adaptive aggregation networks for class-incremental learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2544–2553.

[48] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Advances in neural information processing systems* 30 (2017).

[49] J. Luo et al. "ThiNet: Pruning CNN Filters for a Thinner Net". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 41.10 (Oct. 2019), pp. 2525–2538. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2858232.

[50] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. "Piggyback: Adapting a single network to multiple tasks by learning to mask weights". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 67–82.

[51] Arun Mallya and Svetlana Lazebnik. "Packnet: Adding multiple tasks to a single network by iterative pruning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7765–7773.

[52] Marc Masana et al. "Class-incremental learning: survey and performance evaluation on image classification". In: *arXiv preprint arXiv:2010.15277* (2020).

[53] Cristina Mata et al. "StandardSim: A Synthetic Dataset for Retail Environments". In: *International Conference on Image Analysis and Processing*. Springer. 2022, pp. 65–76.

[54] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[55] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects". In: *Frontiers in psychology* 4 (2013), p. 504.

[56] Jan Neumann, Hanan Samet, and Aya Soffer. "Integration of local and global shape analysis for logo classification". In: *Pattern recognition letters* 23.12 (2002), pp. 1449–1457.

[57] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[58] John M Pierre. "Incremental lifelong deep learning for autonomous vehicles". In: *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE. 2018, pp. 3949–3954.

[59] Qi Qian et al. "Softtriple loss: Deep metric learning without triplet sampling". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 6450–6458.

[60] Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. "Lifelong generative modeling". In: *Neurocomputing* 404 (2020), pp. 381–400.

[61] Amal Rannen et al. "Encoder based lifelong learning". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1320–1328.

[62] Sylvestre-Alvise Rebuffi et al. "icarl: Incremental classifier and representation learning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[63] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[64] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[65] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[66] David Rolnick et al. "Experience replay for continual learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[67] Stefan Romberg et al. "Scalable logo recognition in real-world images". In: *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*. 2011, pp. 1–8.

[68] Amir Rosenfeld and John K Tsotsos. "Incremental learning through deep adaptation". In: *IEEE transactions on pattern analysis and machine intelligence* 42.3 (2018), pp. 651–663.

[69] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[70] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[71] Joan Serra et al. "Overcoming catastrophic forgetting with hard attention to the task". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4548–4557.

[72] Hanul Shin et al. "Continual learning with deep generative replay". In: *Advances in neural information processing systems* 30 (2017).

[73] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[74] Hang Su, Shaogang Gong, and Xiatian Zhu. "Weblogo-2m: Scalable logo detection by deep learning from the web". In: *Proceedings of the IEEE international conference on computer vision workshops*. 2017, pp. 270–279.

[75] Hang Su, Xiatian Zhu, and Shaogang Gong. "Deep learning logo detection with data expansion by synthesising context". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 530–539.

[76] Hang Su, Xiatian Zhu, and Shaogang Gong. "Open logo detection challenge". In: *arXiv preprint arXiv:1807.01964* (2018).

[77] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.

[78] Sebastian Thrun and Tom M Mitchell. "Lifelong robot learning". In: *Robotics and autonomous systems* 15.1-2 (1995), pp. 25–46.

[79] Andras Tüzkö et al. "Open set logo detection and retrieval". In: *arXiv preprint arXiv:1710.10891* (2017).

[80] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[81] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[82] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[83] Jing Wang et al. “LogoDet-3K: A Large-Scale Image Dataset for Logo Detection”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 18.1 (2022), pp. 1–19.

[84] Yue Wu et al. “Large scale incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 374–382.

[85] Ju Xu and Zhanxing Zhu. “Reinforced continual learning”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[86] Shipeng Yan, Jiangwei Xie, and Xuming He. “Der: Dynamically expandable representation for class incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3014–3023.

[87] Syed Sahil Abbas Zaidi et al. “A survey of modern deep learning based object detection models”. In: *Digital Signal Processing* (2022), p. 103514.

[88] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual learning through synaptic intelligence”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3987–3995.

[89] Aston Zhang et al. “Dive into Deep Learning”. In: *arXiv preprint arXiv:2106.11342* (2021).

[90] Chi Zhang et al. “Few-shot incremental learning with continually evolved classifiers”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12455–12464.

[91] Junting Zhang et al. “Class-incremental learning via deep model consolidation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1131–1140.

[92] Bowen Zhao et al. “Maintaining discrimination and fairness in class incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13208–13217.

[93] Zhong-Qiu Zhao et al. “Object detection with deep learning: A review”. In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[94] Zhaohui Zheng et al. “Distance-IoU loss: Faster and better learning for bounding box regression”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 12993–13000.

[95] Da-Wei Zhou, Han-Jia Ye, and De-Chuan Zhan. “Co-transport for class-incremental learning”. In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 1645–1654.

[96] Da-Wei Zhou et al. “PyCIL: A Python Toolbox for Class-Incremental Learning”. In: *arXiv preprint arXiv:2112.12533* (2021).

[97] Zhengxia Zou et al. “Object detection in 20 years: A survey”. In: *arXiv preprint arXiv:1905.05055* (2019).