POLITECNICO DI TORINO

Master degree in Data Science and Engineering

Master Thesis

Modelling the background in Incremental Object Detection

Supervisor
Prof.ssa Barbara Caputo

Correlatore:
Dott. Fabio Cermelli
Dott. Dario Fontanel
Prof. Marcello Restelli

Candidate
Antonino Geraci

April 2022

Modelling the background in Incremental Object Detection
Master thesis. Politecnico di Torino, Turin.

April 2022.

Contents

List of Tables

List of Figures

Abstract

Object detection is one of the most central tasks in computer vision, its purpose is to locate objects in space, classifying them and allowing a separation from the background. Traditional object detection architectures are designed to deal with a known set of data and classes that are used to train from scratch the model. Humans naturally learn in a continual fashion: if a new object is discovered, it can be learnt without forgetting old object seen in the past, therefore human brain is intrinsically incremental. On the other hand neural networks and consequently object detectors are not built to easily perform incremental learning. Although a network is perfectly trained, when it is adapted to a new set of classes, performances on old classes degrades and the network face difficulties to learn new classes as well. In object detection another problem arises: the background is shared among old and new classes, thus meaning that a new class to learn, in the old model, could be considered as background; this additional problem leads to confusion for the model between background and objects, thus worsening the problem of forgetting.

This is a practical limit when a model has been trained on a batch of classes and additional kind of objects should be learned by simply feeding new images to the detector.

Many approaches were carried out to solve the problem of continual learning in different computer vision tasks and recently also in object detection.

This work aims to analyze the structure of Faster-RCNN, one of the main used architectures for object detection in order to alleviate the aforementioned problems concentrating on modeling the background. A set of different loss functions is proposed in order to teach to the model what is background and what is an object. Moreover, since Faster-RCNN can be extended to perform instance segmentation, this work aims to prove that the approach used for object detection fits also for another similar task. Experiments made on Pascal VOC [11] and SBD datasets demonstrate that this approach improve the capability of the network to separate objects from background during the incremental steps, improving also overall performances.

Chapter 1

Introduction

In the last decade, the advancement of many different technologies and the introduction of new high tech paradigms in already well established tasks, pushed Artificial Intelligence research with the aim of assisting and sometimes replacing humans solving complex or time wasting-tasks.
There is actually a twofold objective for AI systems : firstly to solve tasks that can be addressed processing tons of data and taking into account many more variables than a human can handle, for instance making prediction on future trends, discover underlying patterns, extract insights to make holistic decisions; secondly to make machines able to achieve everyday tasks that people are able to complete, making computers able to automatically and autonomously complete them. Some example of the latter are : Computer Vision and Natural Language Processing.
In just ten years this field saw enormous growing, especially thanks to the advent of Deep Learning, which, by mimicking the structure of human mind enables in computers the concept of abstraction, that is not possible using standard algorithms. The main difference lays in how these algorithms are built: in standard programming the coder knows how the computer behave in every possible scenario, therefore the choice that an algorithm makes are deterministic and decided *a priori* by the programmer , on the other hand, artificial intelligence algorithms are based on learning, the system learns from data as a person learns from experience. Machines would be capable of carrying out complex actions and reasoning, learning from mistakes, and performing functions that until now were exclusive to human intelligence.
The task in which deep learning started to grow substantially is Computer Vision. Its main goal is to extract high-level knowledge from an image to solve many different tasks. The most investigated task and surely the most

trivial is Image Recognition: understanding what an image is representing. From this tasks tons of disparate algorithms were carried out, going deeper in the understanding of the content of an image. Therefore two main areas are starring in today research: Object detection and Semantic Segmentation. Both tasks aim to use computer vision in real scenarios, in which the objective is not to predict the category of a single object image, but to recognize in an image different objects and rather than just deciding the category these algorithms are able to spatially describe the objects recognized. Object detection algorithms are able to put a bounding box on the object and deciding which is the label, on the other hand semantic segmentation is able to color the pixels of the image basing on the class to which those pixels belong. Moreover, enhancing the concept of semantic segmentation and mixing it with object detection, another task can be carried out : instance segmentation, that, besides coloring the right pixels, is able to distinguish the colored objects, thanks to object recognition. A visual explanation is given in fig.4.3.3
Therefore, these two tasks, in contrast to simple image classification are applied on multi-object images, making them suitable for many applications. To cite some of them: autonomous driving, medical image processing, facial recognition, anomaly detection in plants for quality control.

In deep learning there is a strong assumption that makes automatic learning far from human capabilities: the algorithm is trained once knowing in advance all the categories that the detector is supposed to recognize. Humans have the ability to continuously learning and discover new objects, without forgetting what they already learnt, on the other hand deep learning architectures have not the correct structure to achieve the same goal. Therefore one of the main challenges in computer vision is incremental learning: namely, the ability of an agent to learn new classes without forgetting how to recognize the ones learned before.

This problem adds complexity to standard algorithms and the solution to it is deeply connected with the computer vision task to which it is applied. As recognition tasks, also incremental learning have many applications in real world. The trivial one is to make the learning of automatic agent continual, allowing them to discover on their own what they do not know and then automatically learn new concepts.

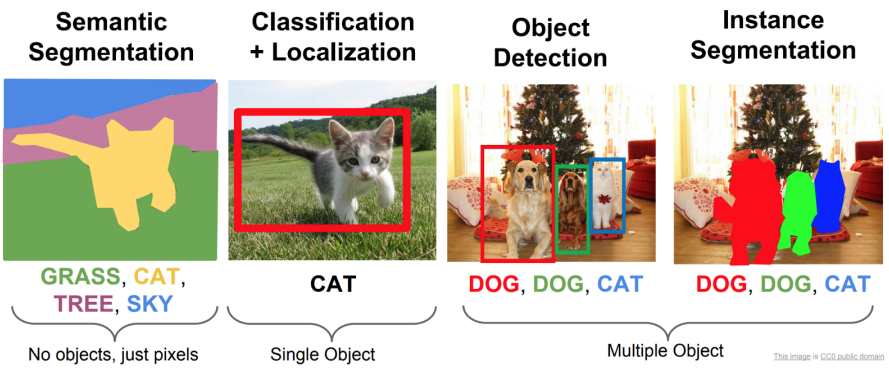

Figure 1.1. Difference between image classification, object detection and segmentation [2]

1.1 Research goals and main contributions

This work is on the lookout for exploring the behavior of object detection and instance segmentation when casted in the incremental learning setting. Since the framework used for the aforementioned tasks is composed by many different modules, the main aim of this work is to understand which of these modules are particularly affected by the incremental setting and try to avoid the drop in performances designing corrective strategies to enhance the reliability of the detector when new classes are added.

Object detection is a key task in computer vision that has seen significant development in recent years. The advances were made possible by the rise of deep neural network architectures [12, 13, 9, 14, 15, 16], which improved results while reducing computation time. Despite the advances, these architectures assume that they already know all of the classes they will encounter and are not designed to incrementally update their knowledge to learn new classes over time. A naïve method would be to restart the training process from scratch, obtaining a new dataset containing all of the classes and retraining the architecture. This is impractical, however, because it would need a considerable computational overhead to re-learn previously learned classes, as well as the usage of previously trained data that may no longer be available due to privacy issues or intellectual property rights.

A more effective option is to employ incremental learning and to update models continuously in order to extend their knowledge to new classes by training exclusively on fresh data and avoiding catastrophic forgetting [17].

Incremental learning has initially been studied in the context of image classification [18, 19, 20, 21, 22, 23, 24] but it has only recently been applied to more complex tasks like object detection [25, 26, 27, 28, 29] and semantic segmentation [30, 31, 32, 33, 34]. Performing incremental learning in object detection (ILOD) imposes additional challenges because each image contains multiple objects and, following the definition in [25], only objects belonging to new classes are annotated while the rest (objects belonging to either old or future classes) are ignored, introducing missing annotations.

Prior research has focused on the implementation of regularizations to minimize catastrophic forgetting, but the impact of missing annotations has been disregarded. Regions that lack annotations, in particular, are frequently referred to as background areas, and the model classifies them as such. As a result, unannotated objects will be associated with the backdrop, increasing catastrophic forgetting in previous classes and complicating training in subsequent classes.

To overcome this issue, inspired by [30], we revisit the common knowledge distillation framework in ILOD [25, 26, 29] proposing MMA, that **M**odels the **M**issing **A**nnotations in both the classification and distillation losses.

To avoid catastrophic forgetting, we allow the model to predict either an old class or the background on any location not related with an annotation on the classification loss. Alternatively, because current classes may have been annotated as background in a previous learning step, we revisit the distillation loss, matching the teacher model's background probability to the probability of having either a new class or the background, thereby facilitating the acquisition of new classes.

On the Pascal-VOC dataset [11], we demonstrate the utility of our method by examining a variety of single-step and multi-step tasks. Without using any image from previous training steps, we show that our method outperforms the current state-of-the-art.

Finally, we show that by adding an additional knowledge distillation term to our framework, we can easily extend it to the task of instance segmentation. On the Pascal SBD 2012 dataset [35], we show that our method outperforms the other baselines.
To summarize, the contributions of this thesis are as follows:

- We identify the peculiar missing annotations issue in incremental learning for object detection.
- We propose to revisit the standard knowledge distillation framework to cope with the missing annotations, showing that our proposed MMA

outperforms previous methods on multiple incremental settings.

- We extend our method to instance segmentation and we show that it outperforms all the other baselines.

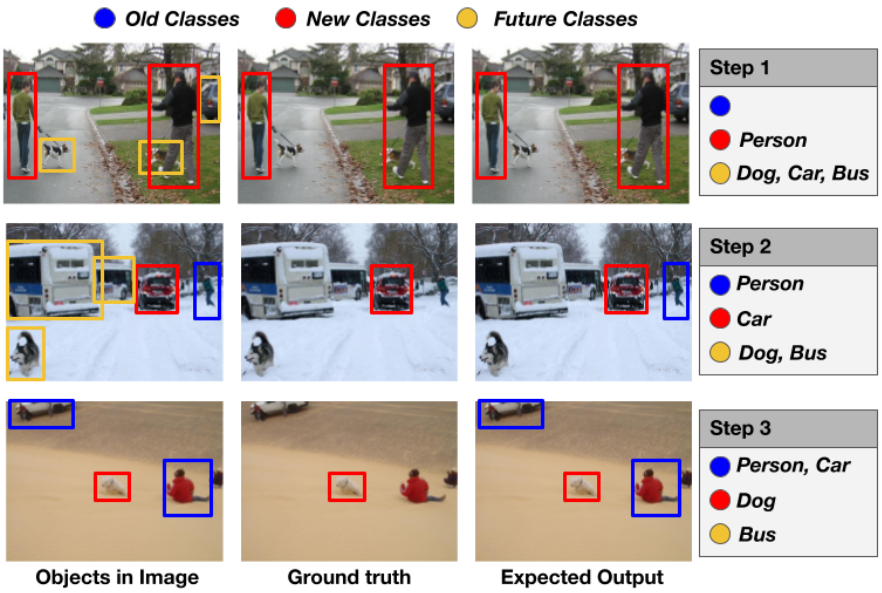

Figure 1.2. An illustration of the missing annotation issue of object detection in different time steps. At training step t, the annotations are provided only for new classes (red), while all the other objects, both from old (blue) and future (yellow) steps are not annotated.

Chapter 2

Deep Learning

This chapter aims to provide an overview of deep learning base modules that are relevant for this work, deepening aspects that are fundamental for object detection, incremental learning and instance segmentation.

Deep learning emerges as a branch of machine learning to avoid costly manual pre-processing trying to mimic the organization of information as the human brain does. Deep learning is concerned with the development of algorithms for learning several layers of representation in order to characterize complicated data connections. As a result, higher-level features and ideas are defined in terms of their lower-level equivalents, and this form of feature hierarchy is known as deep architecture. Deep learning's present popularity may be attributed to three factors: considerably improved chip processing capabilities enabled by the introduction of GPUs, vastly increased training data size, and recent advances in machine learning. All of these developments have enabled deep learning systems to successfully exploit complex nonlinear functions, to rapidly train distributed and hierarchical feature representations, and to make effective use of both labeled and unlabeled data. Deep Learning relies heavily on neural networks. They are, in fact, specific structures meant to function similarly to the human brain.

2.1 Neural networks

In shallow learning one of the main used algorithms is the perceptron [36], that is able to find a separation in training data between different classes, in order to decide how to categorize new coming test data. The problem of this architecture and all the variants and extensions is that they are not able to

learn non linear functions. Generally many problems can be considered as linearly separable and especially when handcrafted features are selected and cleaned in the correct way, these algorithm perform satisfactorily.
Many years before the birth of deep learning many works have demonstrated the insufficiency of such algorithms and the need to add non linearity and complexity to architectures, since the world that they want to discover is as well non linear and complex. The classic example of non linear function that a perceptron cannot approximate is the XOR problem [37]. In order to correctly make decisions in this case a layer must be added. the layer conceptually represents an additional step of decision, that, if intermediated with non linearity should be able to overcome the aforementioned problems. In this way the MLP (Multi layer perceptron) was born.
Iterating this process the concept of neural network arises. A neural network is a function approximator that taking in input many data points $\mathbf{x}$ tries to approximate the function { by learning parameters $\boldsymbol{\theta}$ to define a mapping between a data point x and the corresponding label y, ending in the equation: $\mathbf{y} = f(\mathbf{x};\boldsymbol{\theta})$. The number of these parameters $\boldsymbol{\theta}$ defines the depth of the network: the more parameters, the more the network is complex and deep. Since usually, in real applications the chains are very long, the field of machine learning that uses these architectures is called deep learning.
Diving in these deep architecture we cannot talk anymore about handcrafted features, because the presence of many layers and non linear activation functions allows the network to create different levels of representation that are by definition *hidden*. Hence, deep architectures enhance the capability in learning patterns but at the same time lowers the interpretability of the decision

Neural network training

2.1.1 Activation functions

The element of neural networks that makes the whole system non-linear is the activation function. When compared to a neuron-based model seen in human brains, the activation function is responsible for determining what is to be fired to the next neuron at the conclusion of the process. In an neural network, it performs the same job. It takes the preceding cell's output signal and turns it into a format that may be used as input to the following cell.

Despite the similarity with human brain, they are fundamental to keep the neurons' outputs restricted to some limits. If we just use linear activation (i.e. $\mathbf{W}x + b$), the growth of the output would be uncontrolled, leading both to computational issues and poor learning stability.
The main contribution, as mentioned before, is to make the network non linear. A linear classifier is able to output a pattern that can approximate linear behaviors but not to exactly adapt to the function we want to approximate, what basically a linear classifier does is to learn a group of multipliers parameters $\mathbf{W}$ and a group of biases $\mathbf{b}$, such that $\mathbf{y} = \mathbf{W}x + \mathbf{b}$. The latter would be the final formula if we want to use a single cell neural network, If we stack multiple layers, representing the i^{th} layer as $f_i(x)$, we have: $o(x) = f_i(f_i(....f_1(x))$, these function remains linear and therefore, although it is more complex, it is not able to approximate non linear functions. In order to make the model get the power to learn the non-linear patterns, specific non-linear layers are added in between.
Some of the most used activation functions are:

Sigmoid

It is defined as:

$$o(i) = \frac{1}{1 - e^i} \tag{2.1}$$

The sigmoid function ranges between 0 and 1. Notably, in contrast to the binary step and linear functions, the sigmoid is a nonlinear function. Thus, when having multiple neurons with sigmoid activation functions, the output would be nonlinear as well. This activation function is mainly used as a normalization for final output layers and not in the middle of deep architechtures, since, as noticeable from the graph, the derivatives, when |x| becomes higher vanish, leading to problem in the backpropagation algorithm (vanishing gradient problem).

Hyperbolic tangent

It is defined as:

$$o(i) = tanh(i) \tag{2.2}$$

where o is the output and i is the input. A desirable behavior for an activation function is to be zero centered, in this way the gradient flowing through it would not shift in a particular direction. On the other hand it carries the same problem of the sigmoid: vanishing gradient in the tails.

ReLU (Rectified Linear Unit)

It is defined as:

$$o(i) = max(i,0) \tag{2.3}$$

This is one of the most used activation function, especially with Convolutional Neural networks. Contrarily to the aforementioned functions it does not saturate and consequently does not cause the Vanishing gradient problem. It has some issues:

- It is not zero centred
- it suffers from "dying ReLU" problem: Since the output is zero for all negative inputs, some cells are not able to learn anything, because their gradient would always be zero
- Exploding gradient: since it is not a limited function the activations could assume very high values, leading to instability

Leaky ReLU

It is defined as:

$$o(i) = \begin{cases} x & x >= 0 \\ \alpha x & x < 0 \end{cases} \tag{2.4}$$

where α is an hyperparameter, smaller than 1 (0.1, 0.01 are typical values) It solves the "dying ReLU" problem, by not zeroing negative output, but just smoothing them with a modulation value α

2.1.2 Training of neural networks

The aim of neural networks is to find the best set of parameters θ that allows to best approximate the complex non linear function that binds input data and desired output. To this extent an optimization algorithm is the best way to update those parameters during different iterations (i.e. every iteration is the input of a new data point in the network).
In neural networks the function to optimize is called *loss*, and it is a function of the output and of the ground truth (i.e. the expected output of the network). Mathematically: $L(o(x), y)$. This function must be 0 if o(x) exactly match the ground-truth y and should grow proportionally to the misalignment between y and o(x).

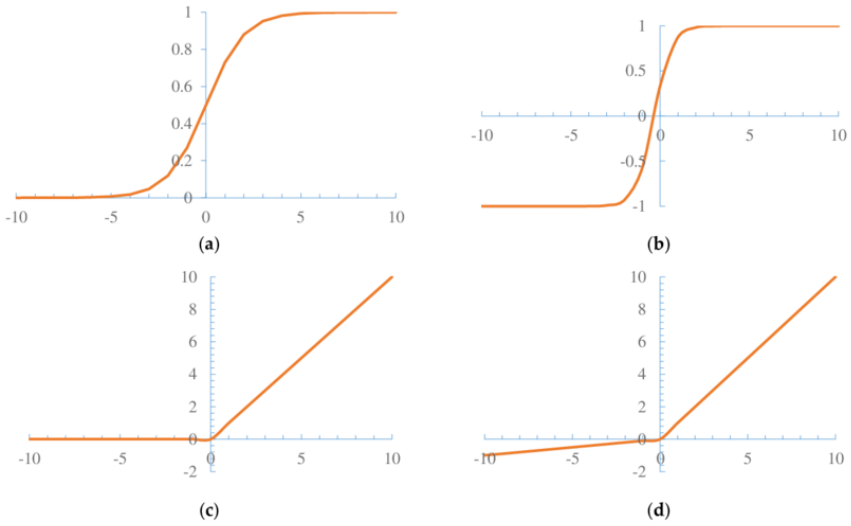

Figure 2.1. Graphs of different activation functions (a) Sigmoid, (b) Tanh (c) RELU (d) Leaky RELU [3]

Gradient Descent

As mentioned before, the aim of a training is to minimize the loss function. One of the most used algorithm to accomplish this task is the Gradient Descent, as its name underlines, it involves calculating the gradient of a certain function : the loss. The objective is to minimize $L(o(x), y)$ in order to find the best set of parameters θ, recalling that o(x) is actually o(x;θ), meaning that the function o is dependent from the updatable parameters of the network. The derivative of the loss is a very important spot, because it gives hints to know where the function is reducing and where it is growing. So, we can minimize L by gradually increasing x in the opposite direction of the derivative. Due to the fact that we are dealing with multidimensional data, the derivative's generalization is the gradient ∇, and so we can decrease f by travelling in the direction of the negative gradient.

Accordingly, the formula to update at each step θ is:

$$\theta_t = \theta_{t-1} - \alpha \nabla_\theta L(\theta) \tag{2.5}$$

where α is the learning rate: a small constant that weight the importance of the gradient in the update and that determines the entity of the change of θ: the higher α the higher the change in θ.

The main problem of this approach is that if the function to minimize is not convex, thus having a more complex shape, the optimization could

remain stacked in local minima or saddle point, constraining the problem to not explore other set of parameters that would decrease the value of the loss. Therefore in deep learning *Stochastic gradient descent* is used: instead of calculating the gradient for the whole training set, the set of data is divided in mini-batches and the update is calculated on the results of that mini-batch, in this way the update is stochastic, since the gradients of the target function with respect to the input variables are noisy. This means that the evaluation of the gradient may have statistical noise that may obscure the true underlying gradient signal, caused because of the sparseness and noise in the training dataset. The update rule becomes:

$$\theta_t = \theta_{t-1} - \frac{1}{B_s}\alpha\nabla_\theta \sum_{i=0}^{B_s} L(\theta) \tag{2.6}$$

where B_s is the mini-batch size (i.e. the number of data points x used to calculate the approximate gradient).

Back-Propagation Algorithm

Even if the algorithm to update the weights is defined, it includes the calculation of the gradient of the loss function with respect to the parameter $\boldsymbol{\theta}$. It is not easy to calculate the whole gradient, but there is the need of a trick allowing to compute the partial derivatives efficiently. Back-Propagation Algorithm is used to this extent. It is so important because it shows how much the parameter x needs to change to minimize the loss function.
The gradient is calculated using the *chain rule*: denoting w^l_{jk} the weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} in the l^{th} layer, the activation of a neuron j in layer l is:

$$z^l_j = (\sum_k (w^l_{jk} a^{l-1}_k + b^l_j) \tag{2.7}$$

$$a^l_j = \sigma(z^l_j) \tag{2.8}$$

where σ is an activation function, a^l_j is the activation of neuron j in layer and z^l_j is the output for neuron j in layer l. To update the network parameters correctly, we are interested in calculating:

$$\frac{\partial L}{\partial w^l_{jk}} = \frac{\partial L}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}} \tag{2.9}$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \tag{2.10}$$

$$\frac{\partial L}{\partial w_{jk}^l} = \frac{\partial L}{\partial z_j^l} a_k^{l-1} \tag{2.11}$$

This rule allows a computational efficient calculation of the gradient and it is the bulk of the *backpropagation algorithm*. Iterating the process going backward in the network all the partial derivatives can be obtained and consequently use that quantity to upadate the weights and biases (i.e. $\boldsymbol{\theta}$).

Learning rate and schedule

In equation 2.5 the parameter α is crucial to scale the effect of the gradient. This parameter is named *learning rate*. As the word says it describes the entity of the change in the network weights: the higher α, the higher the change of the parameters. This is a fundamental hyper parameter that need to be tuned depending on the task and the dataset. The rationale is that at the beginning of the training there is the need of an high learning rate, in order to explore more widely the space of solution and try to reach a state in which the loss is lower quickly. Going forward in iterations this value should be *scheduled* in a monotonic descendent way. Lowering the learning rate means changing less the parameters, therefore, when α decreases the optimization algorithm is searching in a reduced neighborhood, assuming that the global optimum is much closer after some epochs.

Overfitting and regularization

Since a neural network should understand underlying patterns and generalize the acquired knowledge to new incoming data, it is not convenient that the model learns also the noise in training data, therefore the model should understand that some particularities are proper of the single data point and avoid to learn them. Therefore a portion of data is always sidelined to form the *test set*. In this way the model can be tested on unseen data. If the error on the training set is very low and not comparable with the one on the test set, overfitting occurred.
There are many apporaches to avoid overfitting:

- *Early stopping* : Cutting the training iterations when the network is learning too much on training data

- Data augmentation : Modify data randomly, not altering dramatically their characteristics, in order to feed different points to the network.
- *Weight decay* : In order to avoid some parts of the network (i.e. some weights) to be too large and take over the other parameters, a regularization expression is added to the update rule described in 2.5. The most used is an *L2 penalty*:

$$L2_{penalty} = \frac{\lambda}{2} \sum_i w_i^2 \tag{2.12}$$

  in this equation λ is named weight decay.

2.2 Convolutional neural networks

Standard neural networks (i.e. MLP) suffer of a problem: each neuron of layer l is connected with all neurons of layer $l-1$ and $l+1$. If some particular data (e.g. images) are fed in MLP, the quantity of parameters could be huge, since the input size is already high. This aspect could cause overfitting (since the network tries to work pixel by pixel) and also computational issues in training a network with too many parameters. The main observation that leads, for these kind of data, to change the architectures are the local connections: in images, audio recording, natural language, each feature is strongly linked with the surroundings. For instance, in an image is more useful to explore the together close pixels instead of considering the relationship between farther parts of the picture (as MLP does). Therefore the concept of convolutional neural network (CNN) was carried out. It is based on the convolution operation.

2.2.1 Convolutional layer

The objective of convolutional layers is to capture patterns in the starting image and generating another image called activation map. This layers are called *feature extractor layers*, in fact, contrarily to other models CNN are able to extract autonomously features not needing handcrafted approaches to decide which features are more important. The image is just fed in the network and the convolutional layers will transform the input autonomously to make it suitable for classification. A convolutional layer may contain many different filters that can be computed in parallel. Each of these filters has 4 fundamental parameters:

- *Size* : The size (H_k, W_k) of the square filter (usually a common choice is 3x3)

- *Depth*: How many filters to use (D)

- *Stride*: Denote how many pixels to move the filter (S)

- *Padding*: This is a parameter of the image itself, it describes the size of the frame to apply on the image to enlarge it. (P)

Filters are applied to the image, obtaining as many feature maps as the number of filters, they are then stacked together and passed for the next convolutional layer. The convolutional operation can be finally described by the following

$$Conv(p_h, p_w) = \sum_{i}^{H_k} \sum_{j}^{W_k} k_{i,j} x_{i+p_h, j+p_w} \tag{2.13}$$

where p_h and p_w is the position in the image of the filter, x is the input image and k is the kernel. Particularly, output dimensions are described by these equations:

$$\begin{aligned} H_{out} &= \frac{H - H_k + 2P}{S} + 1 \\ W_{out} &= \frac{W - W_k + 2P}{S} + 1 \end{aligned} \tag{2.14}$$

specifically, the final output would be D feature maps, called also channels of the feature map. Going from the input to the output the network is able to extract different levels of knowledge. The first layer model the high level features, hence general characteristics of images (e.g. edges). Going deeper the network find features that are more related to the class to be predicted.

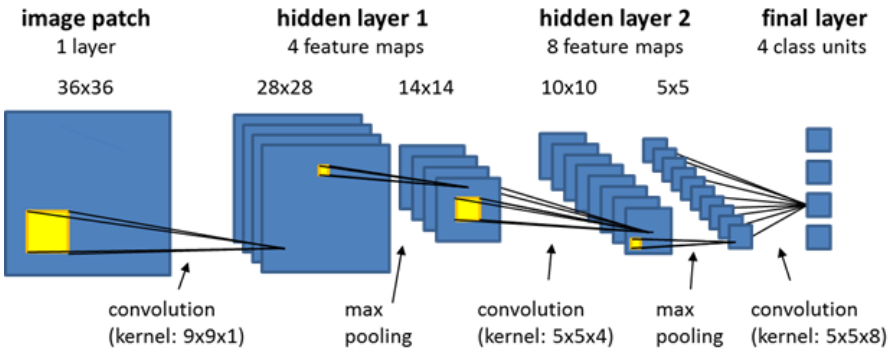

Figure 2.2. Schema of a classic convolutional neural network

2.2.2 Pooling layer

The feature maps' dimensions are reduced by using pooling layers. As a result, the number of parameters to learn and the amount of processing in the network are both reduced. The features contained in an area of the feature map created by a convolution layer are summed up by the pooling layer. As a result, rather than precisely positioned features created by the convolution layer, following actions are conducted on summarised features. As a result, the model is more resistant to changes in the location of features in the input picture.

The most used pooling layers are:

- *Max pooling*: Pooling that chooses the maximum element from the region of the feature map covered by the filter is known as max pooling. As a result, following the max-pooling layer, the output would be a feature map comprising the most prominent features of the preceding feature map.
- *Average pooling* : The average of the items present in the region of the feature map covered by the filter is computed using average pooling. As a result, while max pooling returns the most prominent feature in a feature map patch, average pooling returns the average of all features present in that patch.

2.2.3 Resnet

Until 2015 research was moving towards making deep convolutional architectures deeper, following the erroneous thought that deeper means more

accurate. Actually [4] demonstrated that increasing the number of layers in a classic convolutional architectures (e.g. AlexNet [38] and VGG [39]) would have led to an increase in both training and test error, thus underlining that the problem was not overfitting. The issue was intrinsic in the nature of deep neural networks, especially in the phenomenon of *vanishing gradient*: if the backward pass traverse too much layers the gradient vanishes, making the network unable to update correctly the parameters. To solve this problem [4] introduces the *residual block*. In this architecture there is a direct link that bypasses several levels in between. The *skip connections* or identity mapping, is the most fundamental adjustment to comprehend. This identity mapping has no parameters and serves just to add the output from the previous layer to the next layer.. This results in the ability to train much deeper networks than what was previously possible, because the gradient can flow backward along the identity layer, not vanishing and thus making the optimization easier to perform. The idea behind this concept is that instead of learning a function $\mathcal{H}(x)$ from scratch the network learns $\mathcal{F}(x) + x$, where $\mathcal{F}(x)$ is called *residual*, therefore it should be easier for the model to learn a residual instead of a completely new mapping. Empirically [4] demonstrated that, in Resnets, both training and test error are reduced increasing the depth of the network.

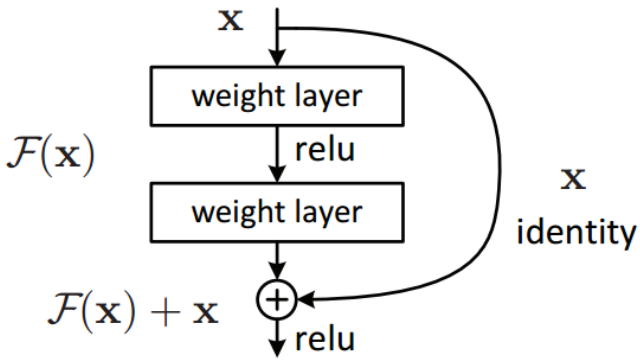

Figure 2.3. Visual representation of a generic residual block ([4]

Chapter 3

Object Detection and Instance segmentation

3.1 Problem definition

Traditional Object Detection. The Object Detection aim is very simple and consists in locating all the objects within an image, labeling them as belonging to one of the *known classes* of the model.

Definition 3.1.1 *Given $\mathcal{K}$ the set of known classes, $\mathcal{X}$ a dataset of images where each image $\mathcal{X}_i$ has size [W_i,H_i], the goal is to build a model $\mathcal{F}$ that maps each image to a a* ***set*** *of pair (**c**, **B**), where **c** and **B** are the label of the detected object and the corresponding bounding box that locates it inside the image. More formally:*

$$\mathcal{F} : \mathcal{X}_i \rightarrow (c, B)^{D_i}$$

$$D_i \equiv \textit{Number of detections for image } \mathcal{X}_i$$

$$c \equiv \textit{Label of one of the known classes}$$

$$c \in \mathcal{K}$$

$$B \equiv \textit{Bounding box}$$

$$B = (x, y, w, h)$$

$$w \equiv \textit{Width of the bounding box}$$

$$w \in [0^+, W_i]$$

$$h \equiv \textit{Height of the bounding box}$$
$$h \in [0^+, H_i]$$
$$x \equiv \textit{Abscissa of the center of the bounding box}$$
$$x \in [\frac{w_c}{2}, W_i - \frac{w_c}{2}]$$
$$y \equiv \textit{Ordinate of the center of the bounding box}$$
$$y \in [\frac{h_c}{2}, H_i - \frac{h_c}{2}]$$

In other words, for each image the model outputs a set of detected objects, each identified both by a label that spatially localizes it ($\boldsymbol{B}$), and by another one that categorizes it ($\boldsymbol{c}$).

Figure 3.1. Object Detection using bounding boxes [5]

3.2 Model Architectures

Modern object detection approaches are dominated by architectures based on convolutional neural networks that differ on whether or not candidate object proposals are used. We can group these works in two different categories: two-stage approaches [40, 6, 41, 9, 8], that generates object proposals which are classified and regressed by a region-of-interest (ROI) head module, and single-stage approaches [42, 43, 15, 44, 45, 46] that simultaneously output

both classification scores and regressed bounding boxes without the need of any object proposal.
While single-stage approaches optimize time as they don't need to generate proposals for each image, two-stage approaches achieve higher performance. For the purpose of this thesis, only two-stage approaches will be analyzed, in particular a two-stage family called *Regions with CNN features* that represent the current state-of-the-art.
Before to that, a brief introduction to deep learning and how it works is given, starting from neural networks and a particular family called **C**onvolutional **N**eural **N**etwork, the most used nowadays in the field of computer vision.

3.2.1 R-CNN

The rise of Deep Convolutional Networks has strongly impacted the computer vision sector, generating a significant increase in performance, especially in image recognition. However, compared to image classification, the difficulties in object detection task remained many, both in qualitative and computational terms. In particular, there were two main challenges:

1. For each image, the model has to generate a set of potential object locations, often called **proposals**.

2. Refine the *proposal* according to the spatial characteristics of each of the known classes in order to achieve better localization (a dining table will almost certainly expand in breadth, but a tree will almost certainly expand in height.)

Different approaches to address both challenges existed before 2014, however some based on CNN were able to detect only a limited set of categories (such as [47],[48],[49]) while others [50] tackled the problem (1) as a regression problem, without guaranteeing good performance.

The first to demonstrate the goodness of CNN for obtaining good results in object detection was [6] through a particular architecture called **R**egions with **CNN** features (R-CNN), which solved the localization problem by operating within the "recognition using regions" paradigm [51]. In particular, its strategy consisted in the generation of a set of category-independent region proposals for the input image through **selective search algorithm** [52], then extracts a fixed-length feature vector from each proposal using a CNN, finally classifies each region with category-specific linear SVMs. Before feeding CNN, a fixed-size CNN input from each proposal is then computed

regardless of the region's shape through a technique called *affine image warping*, as explained in the image below.

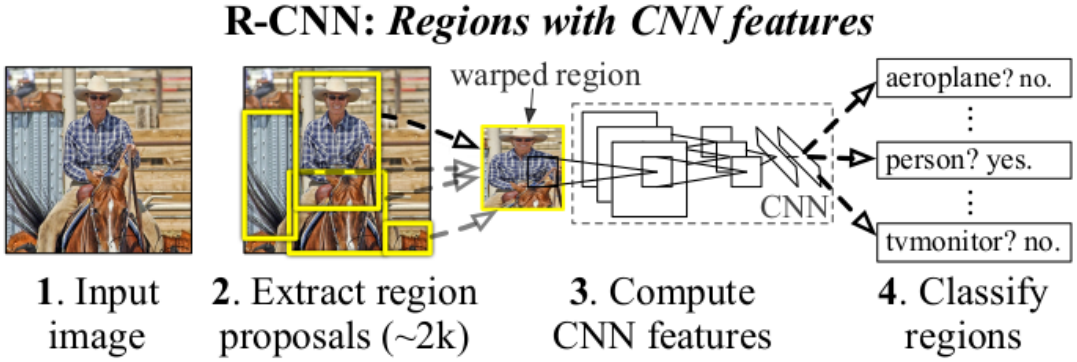

Figure 3.2. R-CNN: Regions with CNN features [6]

Region proposals: selective search. Selective search is a greedy algorithm whose goal is to generate all object locations within an image. Being a heuristic-based algorithm, it guarantees a sub-optimal solution, however some experiments [52] show that it can achieve good performance, by capturing different object at different scales. The basic operation is very simple: initially a region of space within the image that does not spread over several objects is generated, then a similarity is calculated between the starting region and all its neighbors through a generic metric $\mathcal{S}$; the two regions most similar to each other are then joined and the algorithm proceeds iteratively until a single region that spans over the entire image is obtained.
To ensure as much diversity as possible, multiple solutions are generated using both (1) different starting regions and (2) different $\mathcal{S}$-similarity metrics. The different solutions are then combined in such a way that the same locations of objects present in more solutions have a higher probability of being selected.

Intersection-over-Union. After generating a set of *region proposals*, each of them must be assigned a label in order to train the object-category classifier. It is clear that if a region proposal fully overlaps a ground-truth box, the label of that class must be assigned to it; by contrast if it does not intersect any of the ground truth boxes of that image it must be labeled as background. On the other hand, it is more difficult to understand what would

be the best behavior to adopt if the region proposal partially overlaps one or more ground-truth box. In [6], a grid-search over different values shows that a *region proposal* must be labeled as *positive* (i.e. one of the known classes) if it has an **I**ntersection-**o**ver-**U**nion higher than 0.3, which is calculated as follows:

Definition 3.2.1 (Intersection-over-Union (OD)) *Let denote with B_1, B_2 two rectangular boxes whose height is h_1, h_2 and width is w_1, w_2 respectively and left-up corner position (x,y). Let $I : \mathbb{R}^3 \to \mathbb{R}$ a function that receives in input the area of two rectangles and outputs the intersection area; similarly let $U : \mathbb{R}^3 \to \mathbb{R}$ a function that receives in input the area of two rectangles and outputs the union area. Then:*

$$IoU = \frac{I(x, y, w_1 \cdot h_1, w_2 \cdot h_2)}{O(x, y, w_1 \cdot h_1, w_2 \cdot h_2)} \tag{3.1}$$

Definition 3.2.2 (Intersection-over-Union (IS)) *Let denote with M_1, M_2 two segmentation masks. Let $I : \mathbb{R} \to \mathbb{R}$ a function that receives in input the pixels of the two masks and outputs the number of intersection pixels; similarly let $U : \mathbb{R} \to \mathbb{R}$ the pixels of the two masks and outputs the number of intersection pixels. Then:*

$$IoU = \frac{I(M_1, M_2)}{O(M_1, M_2)} \tag{3.2}$$

In other words, the IoU measures the ratio between the intersection area and the union area of two boxes.

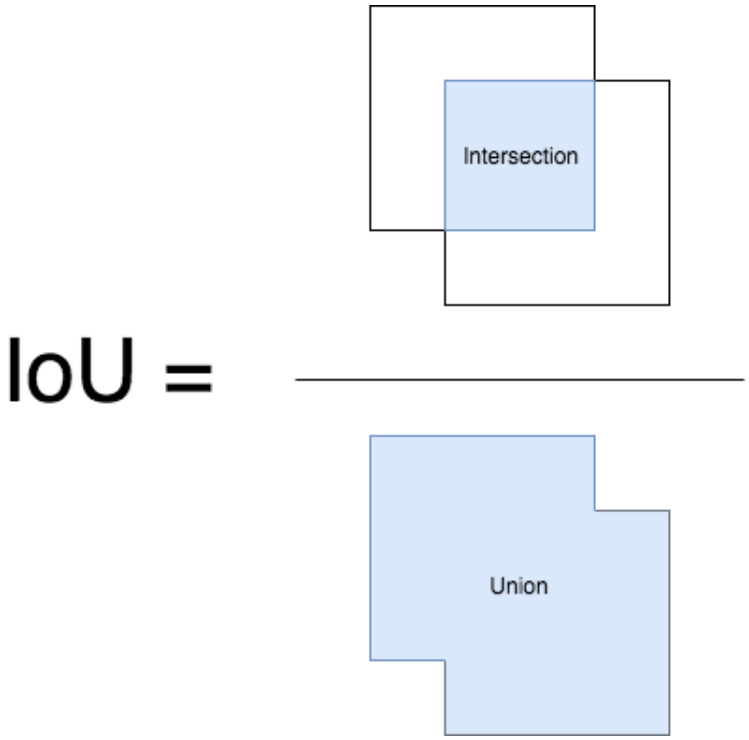

Figure 3.3. Example of Intersection-over-Union between two rectangular boxes [7]

Non-Maximum Suppression. Having now assigned a label to each proposal, the model training phase begins. In particular, each region proposal is fed into the deep convolutional layers of the *R-CNN*, whose last level extracts a fixed-length feature vector of size 4096. Once features are extracted and training labels are applied, a linear SVM per class is optimized by minimizing a *cross-entropy loss*, which is defined as:

Definition 3.2.3 (Cross-Entropy Loss) *Let $\mathcal{X} \in \mathbb{R}^{n,m}$ a generic dataset, $\mathbb{C}$ the set of classes, $y \in \mathbb{C}^n$ the corresponding ground-truth class vector, $p : \mathbb{R}^m \to R^{|\mathbb{C}|}$ the probability score that a data sample belongs to a specific class, then:*

$$CE(\mathcal{X}, y) = -\sum_{x \in \mathcal{X}} \log(p_{y_c}(x)) \tag{3.3}$$

Note that the *cross-entropy loss* looks only at the prediction of the model related to ground-truth class (i.e. y_c), thus the importance of the function $p : \mathbb{R}^m \to R^{|\mathbb{C}|}$ to map the data sample to a probabilistic vector, so the when

the loss is minimized (i.e. when $p_{y_c}(x)$ is equal to 1), the predictions for all the other classes will be 0, that is the desired behavior.

Typically the number of proposals extracted from the selective search is high (about 2000) and contains many overlapping boxes. Subsequently, in order to select at test-time the best box (the one with the highest confidence score) among all those that highly intersect with each other, the **N**on-**M**aximum-**S**uppression algorithm is applied, which works as follows:

Algorithm 1 Non-Maximum-Suppression

Input: A list of Proposal boxes B, corresponding confidence scores S and overlap threshold N.
Output: A list of filtered proposals D.
1. Select the proposal with highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty).
2 Compare this proposal with all the proposals — calculate the IoU (3.2) of this proposal with every other proposal. If the IoU is greater than the threshold N, remove that proposal from B.
3. Again take the proposal with the highest confidence from the remaining proposals in B and remove it from B and add it to D.
4. Once again calculate the IOU of this proposal with all the proposals in B and eliminate the boxes which have high IOU than threshold.
5. This process is repeated until there are no more proposals left in B.

3.2.2 Fast R-CNN

Despite its notable increase in performance with respect to previous approaches, *R-CNN* has three main drawbacks:

1. **Training is a multi-stage pipeline**: First R-CNN fine-tunes the deep convolutional layers on object proposals, then it trains the SVM from scratch by using the CNN feature maps.

2. **Training is expensive in space**: For SVM training, features are extracted from each object proposal and written to disk. This process requires hundreds of gigabytes of storage on 5K images.

3. **Training-Test is expensive in time**: It takes around 2.5 GPU-days to train a model on 5K images; at test-time it takes around 47/s per image.

Figure 3.4. Before and After Non-Maximum Suppression [7]

The main cause of slowdown in the *R-CNN* pipeline is due to the forward process in the deep convolutional layers for each object proposal, without sharing computation. To solve this problem, the same author of [6] in the following years proposed a new pipeline called *Fast-RCNN* [41], in which the training process was speeded up by 3x, while the test process was speeded up by 10- 100x. In particular its main contributes are:

1. **Training is single stage:** it can update all network layers, by making use of a *multi-task loss*;
2. It computes the forward in the deep convolutional layers only once for each image, thus being able to share the computation among all the object proposals generated by an image. As a consequence, **no disk storage** is required.

RoI Pooling Layer. As in 3.2.1, various objects proposals for each image are computed through *selective-search* algorithm. However, in *Fast R-CNN* pipeline the whole image is initially processed with several convolutional and max pooling layers to produce a *conv-feature map*. Then, for each object proposal a **R**egion-**o**f-**I**nterest pooling layer extracts a fixed-length feature vector from the *conv-feature map* itself. In such a way there is no need to process a deep convolutation forward for each object proposal in order to extract the corresponding fixed-length feature vector, as it is calculated starting

from the *conv-feature map*, thus sharing an image-level computation.
As stated in [41], each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w). RoI max pooling works by dividing the h x w RoI window into an H x W grid of sub-windows of approximate size h/H x w/W and then max-pooling the values in each sub-window into the corresponding output grid cell.
Multi-task Loss. Differently from the *R-CNN* in which the classification task is performed by multiple class-specific SVM classifiers, in *Fast-RCNN* [41] the object-proposal feature vectore is fed into a sequence of fully connected layers that finally branch into two sibling output layers: one that produces softmax probability estimates over $\mathcal{K}$ object classes plus a catch-all *background* class and another layer that outputs four real-valued numbers for each of the $\mathcal{K}$ object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes, as defined in **??**. The rationale behind the bounding-box refinition is to adjust, for each class, the spatial coordinates of the shared object location according to intrinsic category characteristics (a dining table will almost certainly expand in breadth, but a tree will almost certainly expand in height).
More formally, given B^p and $\bar{B}^c$ a proposal bounding box and a ground-truth bounding box of class c respectively, the goal is to learn, for each class c, 4 functions $f_x^c(B^p)$, $f_y^c(B^p)$, $f_w^c(B^p)$, $f_h^c(B^p)$ that maps the proposal box B^p to a class-specific box B^c; once learnt these functions, a *scale-invariant transformation* of the center and *log-space translations* of width and height of the proposal box is applied to obtain the predicted class-specific box B^c:

$$\begin{aligned} B_x^c &= B_w^p f_x^c(B^p) + B_x^p \\ B_y^c &= B_h^p f_y^c(B^p) + B_y^p \end{aligned} \tag{3.4}$$

$$\begin{aligned} B_w^c &= B_w^p \exp(f_w^c(B^p)) \\ B_h^c &= B_h^p \exp(f_h^c(B^p)) \end{aligned} \tag{3.5}$$

To learn these function, a *smooth-L_1 regression-loss* L_{reg} is used, that is defined as follows:

Definition 3.2.4 (Smooth-L1 Regression Loss) *Given c the ground-truth class, B^p, B^c and $\bar{B}$ the proposal box, the predicted box for class c and the ground-truth box respectively, then:*

$$L_{reg}(B^c, B^p, \bar{B}) = \sum_{i \in x,y,w,h} smooth_{L_1}(g(B^c, B^p) - \bar{B}) \tag{3.6}$$

where the function g is the function that applies the *scale-invariant transformation* and *log-space translation* and $smooth_{L_1}$ is a robust L_1 loss that is less sensitive to outliers than the typical L_2 loss, both defined as:

$$\begin{aligned} g(B^c, B^p)_x &= \frac{B^c_x - B^p_x}{B^p_w} \\ g(B^c, B^p)_y &= \frac{B^c_y - B^p_y}{B^p_h} \end{aligned} \tag{3.7}$$

$$\begin{aligned} g(B^c, B^p)_w &= \log(\frac{B^c_w}{B^p_w}) \\ g(B^c, B^p)_h &= \log(\frac{B^c_h}{B^p_h}) \end{aligned} \tag{3.8}$$

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{3.9}$$

Finally, the overall *Multi-task Loss* L_{tot} to minimize is:

$$L_{tot}(p_c, c, B^c, B^p, \bar{B}) = L_{cls}(p_c, c) + \lambda[c \geq 1]L_{reg}(B^c, B^p, \bar{B}) \tag{3.10}$$

where p_c and c are the prediction of the model for class c and the ground-truth class respectively, L_{cls} is the standard *cross-entropy loss* defined above (3.3), L_{reg} is the *smooth-L_1 regression-loss* L_{reg}, $[c \geq 1]$ is the Iverson Bracket indicator function that evaluates to 1 when $c \geq 1$, 0 otherwise; λ is a weighing factor to balance both loss terms, that is set to 1 in [41].
Since the *background* class is labeled as 0, the corresponding regression loss is ignored because there is no notion of a ground-truth background box.

3.2.3 Faster R-CNN

This section examines in detail the final architectural model of the trilogy that began with *R-CNN*: the *Faster R-CNN* [8].
In the transition from *R-CNN* to *Fast R-CNN* both time and space cost were drastically reduced thanks to sharing convolution across proposals. However, a huge bottleneck still remained due to the generation of the object proposals needed to train the *Fast-RCNN* detector $\mathcal{D}$. In fact, although it could be considered an independent module from the detector training and therefore

did not impact the training time, the same thing cannot be said at test-time, in which for each image to predict the corresponding object proposals must be generated and the time spent on it is approximately 2 seconds per image in a CPU implementation. This therefore constituted a major stumbling block for achieving *real-time object detection* with this architecture.
To this end, *Faster R-CNN* introduces **R**egion-**P**roposal-**N**etwork, a deep learning that produces object proposals and at the same time it shares the convolutional layers of the *Fast R-CNN* detector, thus creating a unified network that can be trained in an end-to-end fashion.

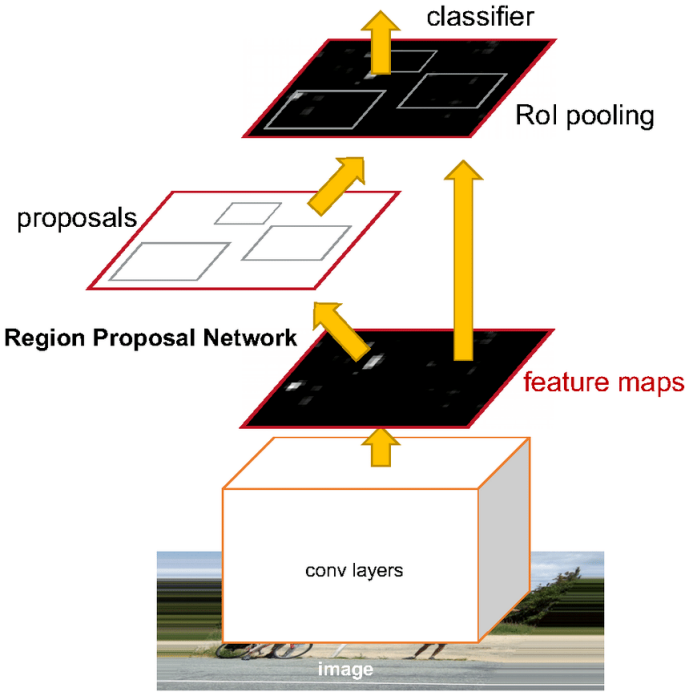

Figure 3.5. Training schema of Region Proposal Network. [8]

Region Proposal Network. As stated in [8], a Region Proposal Network takes and image of any size at input and outputs a set of object proposals, each of them with a probabilistc score of being an object. To generate candidate proposals, a *conv-feature map* is generated as in [41], then a **shared** small convolutional network (3 x 3) slides over the *conv-feature map* and each sliding windows is mapped to a lower-dimensional feature vector of size 256. This feature vector is finally fed into two siblings fully-connected layers: (1)

a *box-regression layers* that outputs the object proposal, and a *classification-layer* that outputs a probabilistic score of that proposal to be an object.
An important novelty of this process is constituted by the concept of **anchors**: in order to build a model that is able to detect same object-categories at different scales and different *height-width* ratios, at each sliding windows the *RPN* simultaneously computes multiple region proposals (called *anchors*). In particular it uses 3 scales (128, 256, 512) and 3 aspect ratios (1:1, 1:2, 2:1) to generate 9 objects proposals at each sliding windows (each possible combination of scale and aspect ratio).

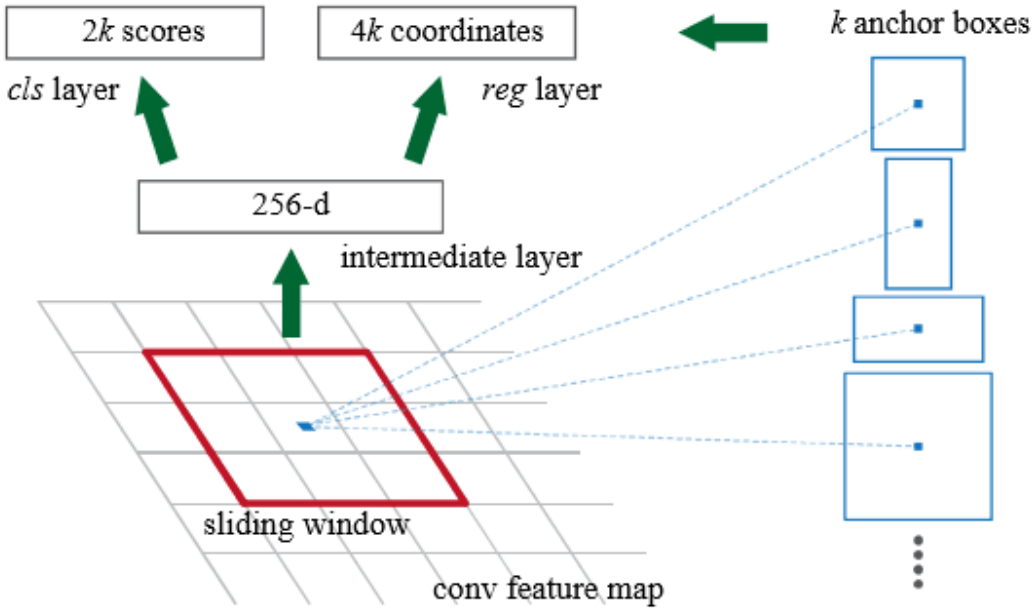

Figure 3.6. Simultaneous computation of anchors starting from the sliding window on conv-feature map. [8]

Thus the *box-regression loss* defined above (3.2.4) has been slightly modified to take into account the anchors:

$$L_{reg}(B^o, B^a, \bar{B}) = \frac{1}{|\mathcal{A}|} \sum_a \sum_{i \in x,y,w,h} smooth_{L_1}(g(B^o, B^a) - g(\bar{B}, B^a)) \quad (3.11)$$

where $\mathcal{A}$ is the set of anchors, B^a, B^o and $\bar{B}$ the anchor box, the predicted box for the object o and the ground-truth box respectively; while the function $g(\cdot, \cdot)$ and the *smooth-L_1 regression-loss* are the same defined in 3.2.4.

This can be thought as a bounding-box regression from an anchor box to a nearby ground-truth box.

Thus the total loss L_{tot} used to train the region proposal network is:

$$L_{tot}(p_o, o, B^o, B^a, \bar{B}) = L_{cls}(p_o, o) + \lambda \cdot o \cdot L_{reg}(B^o, B^a, \bar{B}) \tag{3.12}$$

where o is the ground-truth object class, that is set to 1 if any of: (i) it is the *anchor* with the highest IoU, or (ii) it has an IoU overlap higher than 0.7 with any ground-truth box; otherwise is set to 0. p_o is the probability that the *anchor* is an object, B^a, B^o and $\bar{B}$ the anchor box, the predicted box for the object o and the ground-truth box respectively. As in 3.10, the regression term is calculated only if the *anchor* is positive (i.e. o is set to 1).

Overall Training Procedure. As stated in [8], Both *RPN* and *Fast R-CNN*, trained independently, will modify their convolutional layers in different ways. We therefore need to develop a technique that allows for sharing convolutional layers between the two networks, thus building a single unified network. To accomplish this purpose, to paths can be followed:

1. **Approximate joint training.** In this solution, the *RPN* and *Fast R-CNN* networks are merged into one network during training. This solution is easy to implement. But this solution ignores the derivative w.r.t. the proposal boxes' coordinates that are also network responses, so is approximate.

2. **4-Step Alternating Training.** First the *RPN* is trained in the first step, then the proposals are used to train the *Fast R-CNN* in a second step. At this point the two networks are completely independent and do not share any computation. In the third step, we use the detector network to initialize *RPN* training, but the entire convolutional part is kept frozen and only fine-tune layers unique to the *RPN*. Now the two networks share convolutional layers. Finally, in the fourth step, always keeping the shared convolutional layers fixed, we fine-tune the unique layers of the *Fast R-CNN*. As such, both networks share the same convolutional layers and form a single unified network.

3.3 Instance Segmentation with Faster-RCNN

Mask-RCNN [9] introduces an extension to Faster-RCNN that is able to achieve the instance segmentation task.

A computer vision job for recognizing and localizing an item in a picture is instance segmentation. Instance segmentation is a logical step in the semantic segmentation process, but it is also one of the more difficult approaches to master when compared to other segmentation methods. The purpose of instance segmentation is to provide a view that divides objects of the same class into separate instances. Because the quantity of instances is unknown in advance, and the assessment of the acquired instances is not dependent on pixels, as was the case with semantic segmentation, automating this procedure is difficult. Therefore it goes deeper than object detection, by coloring pixels of the object and not just generating a bounding box for them.

3.3.1 Architecture

Mask R-CNN, which is used for instance segmentation issues, is an extension of Faster R-CNN, with the inclusion of a third branch that allows for the prediction of an instance's mask. This procedure is performed in parallel with two other branches of Faster R-CNN (the bounding box regressor).
Mask R-CNN will construct a mask with a size of 28x28 for each Region Of Interest (RoI), which will then be enlarged until it meets the dimensions of the appropriate bounding box. Mask R-CNN will output the class to which each instance belongs, as well as its bounding box and a binary mask placed on it, for each instance found. This new branch is composed by two convolutional layers: a transposed convolution and a 1to1 convolutional layer. In order to train the network to place correctly those masks another term to the loss is added.Only one mask is associated with each RoI ground truth and a sigmoid activation function will be applied to each pixel of the mask. The branch associated with the mask prediction will generate binary masks having dimensions m $\times$ m for each of the K possible classes. Therefore, in total it will generate $K \cdot m^2$ possible masks, each associated with a different class. L_{mask} is defined as the average of the binary cross-entropy loss functions, in which the k-th mask is included if the region is associated with the k-th ground mask truth:

$$L_{mask} = -\frac{1}{m^2}\sum_{i,j}^{m}[y_{ij}log(y_{ij}^k) + (1 - y_{ij})log(1 - y_{ij}^k)] \tag{3.13}$$

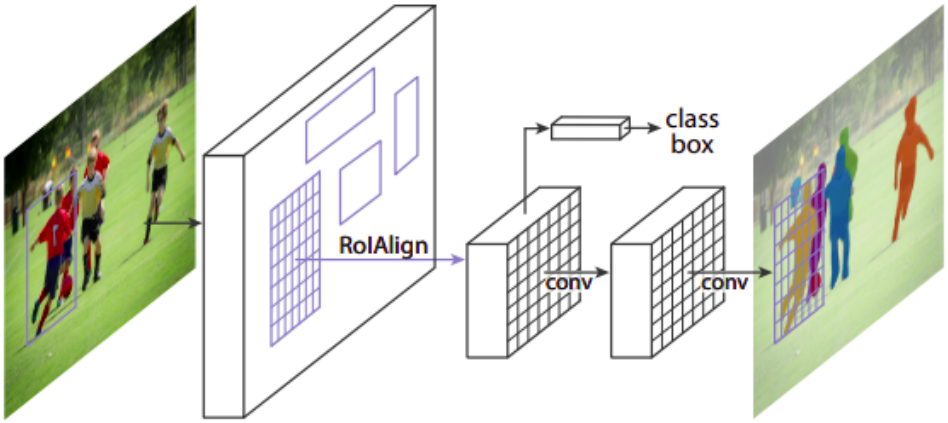

Figure 3.7. Mask-RCNN architecture [9]

Figure 3.8. Instance segmentation example [10]

3.3.2 RoI Align

An important novelty introduced by Mask R-CNN is the use of RoI Align to replace RoI Pooling for the extraction of the RoI of a feature map. Suppose we have a 256 × 256 size image and a feature map to it associated with dimensions 50 × 50. We want to extract a RoI of dimension 30x30 of the original image. Therefore, it would be necessary to extract a region of pixels from the feature map equal to m × m. If we wanto to calculate m it is: $\frac{50*30}{256} = 5.86$.

Using RoI Pooling, it will not be possible to extract a region of 5.86 × size 5.86, but an approximate region equal to 5 x 5 will be considered (i.e. the integer part of m will be considered). This approximation produces a loss of 0.86 pixels by dimension: it is associated with a loss of information as the techniques pooling will not take into account the lost pixels. This pixel misalignment problem has been solved thanks to the RoI Align layer, which no longer approximates the size of the extracted regions, in fact it uses bilinear interpolation to not loose any information. Considering the proposed example again, the extracted region will actually have a dimension 5.86 x 5.86.

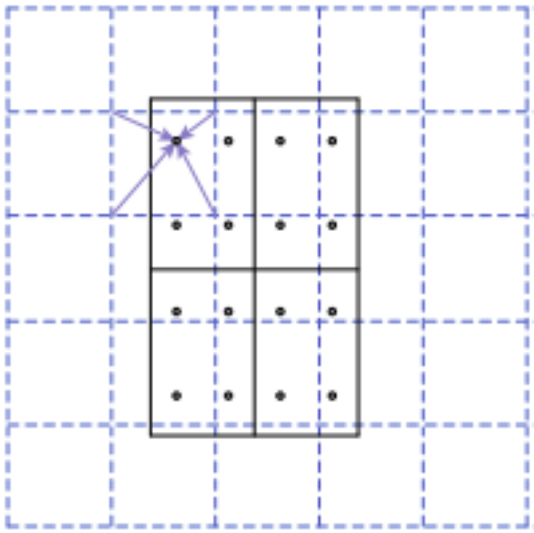

Figure 3.9. Roi align representation, each point represent a sampling point into a single pixel of the feature map, then alignment would be performed through bilinear interpolation [9]

Chapter 4

Incremental learning

4.1 Problem description

The goal of incremental learning is to create artificially intelligent systems that can learn new tasks from fresh input while retaining knowledge from previously learned tasks. It allows for efficient resource usage by overcoming the need to retrain from scratch when new data with different classes arrive, thus reducing memory usage by limiting the amount of data that should be stored. In this way a model can be delivered and extended without having (or partially having) old training data. This is suitable also for application that requires high privacy standards.
The most difficult aspect of incremental learning is catastrophic forgetting, which refers to a sharp reduction in performance on previously acquired tasks after the acquisition of a new one. Moreover there is a contrasting objective between old and new class learning, the model would maintain the knowledge on old classes avoiding catastrophic forgetting and at the same time aims to learn new classes; therefore also new classes are difficult to be learned, because the model is not free to be updated on new classes as it would have done in a standard end to end training on data.[53]
The most difficult aspect of incremental learning is learning from data from the present task while avoiding forgetting previously learned tasks.

4.2 Mathematical formulation

An incremental learning problem P consists of a sequence of n tasks:

$$\mathcal{P} = [(C_1, D_1, (C_2, D_2), ..., (C_n, D_n)] \tag{4.1}$$

where C_t represents a set of classes available in the training dataset D_t. During training for step t, the learning model P_t can use data belonging to D_t, and the tasks do not overlap in classes:

$$C_t \cap C_{t+1} = \emptyset \quad \forall t \tag{4.2}$$

In this case the learner P_t is a deep architecture that should be optimized as described in the previous chapter. This model must incrementally include the classes at the previous step, thus meaning that at training time the model must not process previous classes, but at test time it must be able to recognize them. Therefore at test time we would have a model P_t that is able to classify:

$$\bigcup_{\tau=1}^{t} C_\tau \tag{4.3}$$

4.3 Approaches to solve incremental learning

General approaches to solve the problem of incremental learning are:

- *Regularization approaches*: approaches that use regularization terms together with the classification loss in order to mitigate catastrophic forgetting.
- *Rehearsal approaches*: Rehearsal methods keep a small number of exemplars (exemplar rehearsal), or generate synthetic images or features (pseudo-rehearsal) of old classes to have at training data also a representation of old data.

4.3.1 Transfer learning and finetuning

The naïve finetuning strategy, which has been successful in solving transfer learning issues, suffers from a lack of data from earlier tasks, and the resultant classifier is unable to retain previous knowledge.
Transfer earning is when relevant knowledge from a previously trained AI model is "imported" and utilized as the foundation for a new model. The main problem is that the assumption of having a model trained on a huge dataset is strong itself, moreover, to transfer correctly knowledge the two tasks that we want to solve must be very similar, and when the target network is trained the set of classes is known and do not change. On the other hand, in an incremental setting, where annotations are missing finetuning

and transfer learning plummet in performance, because the two tasks are too much different. For these reasons finetuning strategy is always a lower bound in all incremental learning benchmarks. It is used as a baseline to drive the research and to understand if the model is working properly (i.e. if it is achieving better results than finetuning). Finetuning consist of making the weights of a model free to update,in a training from scratch fashion, to adapt the network to new images. In incremental learning this leads to catastrophic forgetting [54], since the new classes are learnt correctly because the weights update accordingly to the new images, on the other hand old classes are completely forgotten because old classes are not present (or not annotated) in incoming data.

4.3.2 Regularization

Regularization is one of the most used approaches in literature for continual learning, namely one or more terms are added to the classification loss to avoid it to completely forget the old classes representation.

Weight regularization

In the set of parameter θ of the network, some of them are more representative then others, therefore a smart approach is to understand the degree of importance of a weight $w_{i,t-1}$ for the task $\mathcal{P}_{t-1}$ in order to avoid a wrong update in the task $\mathcal{P}_t$. The idea behind this kind of regularization is : if $w_{i,t-1}$ was very discriminatory for the task $\mathcal{P}_{t-1}$ (hence it was discriminatory for the old class correct classification), then the difference$|w_{i,t} - w_{i,t-1}|$ should be small. The aim of these methods is to find a function $\phi(\theta^t)$ able to quantify this importance. Then, in addition to the classification loss another term can be introduced:

$$L_{wreg} = \phi(\theta^{t-1}) \cdot (\theta^t - \theta^{t-1})^2 \tag{4.4}$$

Since the total loss $L_{tot} = L_{cls} + L_{wreg}$ aims to be minimized, this equation minimizes the difference between old and new model weights weighted on the importance of the weights in the previous task. Kirkpatrick et al. [24] proposed Elastic Weight Consolidation (EWC) in which ϕ is calculated is a diagonal approximation of the Fisher Information Matrix.

Knowledge distillation

Another regularization technique, maybe the most famous and used is Knowledge distillation (KD). This methods are based on the usage of a distillation loss that aims to generally keep the representation for model P_t on C_{t-1} classes as close as possible to P_{t-1} ones. This approach was firstly used in learning without forgetting [20] and generally use the following loss, coupled with the classification loss:

$$L_{dist}(x;\theta^t) = \sum_{c \in C^{t-1}} p_c^{t-1} \cdot log(p_c^t) \tag{4.5}$$

where p_c^t are the softmax probabilities for class c and model P_t. In order to use this approach, contrarily to the previous one, during training the old model (P_{t-1}) must be used to provide old output activation to the new model P_t. This technique works well when between the task there is not a huge domain shift, otherwise, constraining the model to act as the previous one could lead to poor results. This approach is recently combined with attention mechanisms that aim to generate a "localization map" to indicate which are the regions that majorly contribute to the prediction. In this way knowledge distillation can be less constrictive and act just on necessary parameters. Some trivial approaches use activation maps retrieved from the feature extraction, other smarter approaches use Grad-CAM [55] algorithm that is able to to produce an attention map basing on gradients back-propagation.

4.3.3 Rehearsal approaches

Although the mathematical definition of the problem avoids the usage of previous classes data to train the new model P_t, in literature this constraint is relaxed by the usage of exemplars. Instead of not using previous data, a small amount of them is kept (exemplar rehearsal), or synthetic images/features are generated in order to help the network to not forget previous examples. This method surely solves the problem of missing previous data, allowing the network at time step t to use data from all classes. In this way at each step the model is able to build optimally discriminatory weights, since the classes are always trained almost jointly.
Of course, it is crucial that the exemplar set is small, otherwise the model is cheating and going towards the joint-training upper bound. To this extent, in literature, when the model uses exemplars the memory has a fixed size decided a-priori. In this way, going forward with new tasks the representative

exemplars for each class are reduced. Formally we define an exemplar set for a class $c \in C_{t-1}$ E_t^c with memory K as:

$$\begin{aligned} E_t^c &\subset \mathcal{X}_{t-1}^c \\ \sum_{c \in C^{t-1}} |E_t^c| &\leq K \end{aligned} \tag{4.6}$$

Usually, exemplars usage is linked with a sampling strategy. Both when selecting new exemplars and reducing the set there is a need for a rule to select them. Actually random selection is demonstrated to be effective and with almost null computational cost. Other approaches proposed by [18] and others try to select the most representative exemplars basing on features, particularly it tries to maintain the images that minimize the distance from the class centroid; other methods try instead to select exemplars in such a way that the distribution on training data is respected. The latter approaches involve more computational power having not a big gain in performances.

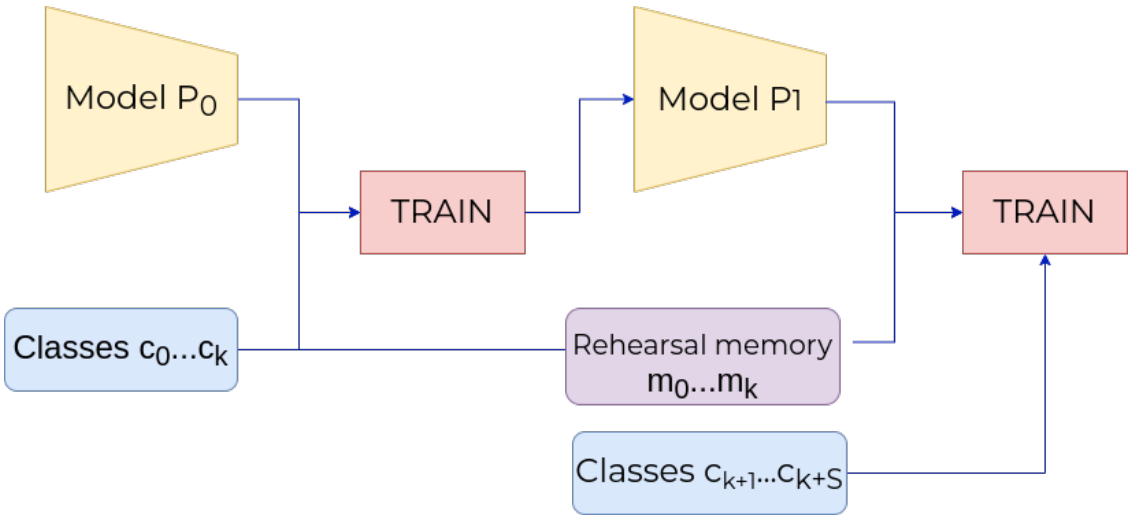

Figure 4.1. Schema of incremental learning general pipeline, in this model the Rehearsal memory is not mandatory

4.4 Background shift: An additional problem in incremental object detection

In image classification, an image contains just one class and furthermore the whole image is classified. On the other hand, in more advanced tasks, such as object detection, an image can contain many different object of different classes. Particularly those objects can belong both to previous or new classes leading to the problem of missing annotations (or background shift).

Let assume an incremental learning pipeline ($\mathcal{P}^0 ... \mathcal{P}^{t^*}$) in which we add many times a single class to a set of already known classes. When the model was trained the first time it was considering the first N classes as foreground and the other classes (not present in the ground thruth) as background. Going forward in incremental steps the new classes learned could be objects that in the previous steps were learnt as background. Moreover, since old classes are not present in the ground truth of the current training step, if they are processed by the model, object from old classes will be recognized as background backpropagating this information and harming the model training. Therefore there is a double problem:

- The model $\mathcal{P}^{t^*}$ is trained considering old objects C^t with $t < t^*$ as background, thus harming the performances on old classes.
- The model $\mathcal{P}^t$ with $t < t^*$ was trained considering new classes as background, thus harming the performances of $\mathcal{P}^{t^*}$ on the new classes, since it will distill the knowledge from the old model that pushes the new classes as background.

Chapter 5

Related works

In this chapter a discussion on methods related to the central task of this thesis are extensively described. The main aim is to analyze their main contributions and find the strengths and weaknesses of these approaches. Particularly, methods are divided by task (Object detection and Instance segmentation), both in an incremental fashion. Moreover the object detection works are divided in those using rehearsal strategies and those that do not have any replay memory; this differentiation in made for the sake of fairness, since when comparing our method with others if a method use exemplars is more likely to achieve better results than methods that does not.

5.1 Incremental object detection

Incremental learning in Object detection has been gaining importance in last years. The first mover in this task [25] proposes a framework based on two stage detectors performing knowledge distillation on the output of Fast-RCNN ROI without any rehearsal strategy. Following this pattern some methods extend the distillation terms by reshaping knowledge distillation on ROI heads [56] and proposing knowledge distillation on RPN and features [26], interestingly [29] proposes a pseudo-positive sampling algorithm to deal directly with background shift. Other methods [57], [58], [59], [60] [61] focused on rehearsal methods to maintain the old task knowledge, both on feature and image replay. Finally [62] and [63] proposed parameter isolated methods to retain knowledge via deep model consolidation, moreover the latter exploit a consistent change in the architecture introducing dilatable ROI heads [63]. As for classic object detection also ICL pipelines show differences in architectures, in fact there is not a clear base learner to the aforementioned

tasks, but they use both one or two stage architectures. In the next sections all these works will be described more in detail.

5.1.1 ILOD

This paper [25] is the first one facing the problem of incremental object detection. It uses the FastRCNN architecture as backbone for performing object detection.
For the authors the main problem in forgetting is linked with the ROI heads, therefore their main contribution in the incremental setting is to add a distillation loss to the ROI heads: cls_{head} and $bbox_{head}$. Particularly, for each image, 64 RoIs out of the 128 sampled from the anchors with the highest objectness score are used to perform knowledge distillation.
The distillation loss is defined as follows:

$$\mathcal{L}_{dist}(p, b_{reg}) = \frac{1}{N|C^{t-1}|} \sum [(p^{t-1} - p^t)^2 + (b_{reg}^{t-1} - b_{reg}^t)^2] \tag{5.1}$$

where N is the number of RoIs sampled, C^{t-1} is the number of old classes, p^t and b_{reg}^t are respectively the class logits and the bounding box regression outputs of the model at time t.
This paper posed the basis for incremental methods, inspired by recent works published for incremental image classification, in fact it use an experimental setting similar to image clssification, in which there is a teacher and the student model the are used together during the learning step t, with the model $\mathcal{P}^{t-1}$ frozen.

5.1.2 Faster ILOD

This paper [26] focuses to apply the aforementioned method [25] to a different backbone: Faster-RCNN Moreover they try to constrict more the learning of the new model by introducing two terms to the distillation loss: feature map KD and RPN KD, maintaining unaltered the RoI distillation loss proposed by ILOD.
The mentioned distillation losses are defined as follows:

$$_{F\_Dist} = \frac{1}{\mathcal{M}} \sum \begin{cases} ||f^{t-1} - f^t||_1 & f^{t-1} > f^t \\ 0 & otherwise \end{cases} \tag{5.2}$$

$$L_{RPN\_Dist} = \frac{1}{N} \sum \begin{cases} ||p^{t-1} - p^t||_2^2 + \beta ||b^{t-1} - b^t||_2^2 & p^{t-1} > p^t \\ 0 & otherwise \end{cases} \tag{5.3}$$

$$\beta = \begin{cases} 1 & p^{t-1} > p^t + \mathcal{T} \\ 0 & otherwise \end{cases}$$

where $\mathcal{M}$ is the total number of activations in the feature map f, $\mathcal{T}$ is a threshold to control regression ($\mathcal{T} = 0.1$) and $\mathcal{N}$ is the total number of anchors. These two additions are actually not too much influential in the training, in fact the highest boost is given by the usage of a Faster-RCNN architecture that generally achieve higher results than the Fast-RCNN.

5.1.3 Multi-View Correlation Distillation for Incremental Object Detection

This paper [56] tries to solve the issue of a too constrained distillation loss, it uses a distillation loss on features similar to [26]. Moreover it adds three different distillation losses:

- **Channel-wise**: Distillation loss that acts on the aggregation of the channels of the feature map leaving the RPN. It uses an SE attention module (squeeze-and-excitation), the "important" channels are therefore chosen by means of a threshold. This information is then used to distill. The distillation is not done directly on the features but on the correlation matrix between the features of the channels deemed important.

- **Point-wise**:it attempts to anchor the more activated points of the feature map output from the RPN to the previous model. In this way this loss tries to keep the knowledge only on specific parts of the feature map, leaving more freedom to the new model to act on the other parts of the image. This is done on the high or low points response in the activation based spatial attention map, which is a sum of the feature map across all channels.

- **Instance-wise**: This loss tries to explore the correlations within the single instance and keep them with respect to the previous model. The

feature map after RoI pooling is divided into parts and the correlation is calculated (within it image) between the various parts. Knowledge about this is maintained by introducing a term in the distillation which is the distance between the correlation matrix on t

he old model and on the new model.

This paper is very recent (summer 2021) and interestingly faces the problem of too much constrictions due to distillation loss, trying to tailor an approach to distill in a smarter way the model.

5.1.4 Lifelong object detection

This paper [29] uses an approach similar to [26], using the same set of losses for both RPN and RoI heads. Moreover it uses a method that deals specifically with background shift. One issue that leads the freshly updated model to forget past class knowledge is that any old object classes in the newly provided photos are considered as "background." This is because the only bounding boxes in the new data that are labeled are those that correspond to the new classes. In order to prevent this problem, the authors propose a method to be aware of pseudo-positive boxes while collecting negative anchors and RoIs for training the student detector on new classes. Pseudo positive boxes are defined as output bounding boxes with probability score greater than 0.5 for any class other than background. Particularly, it prevents pseudo-positive anchors from being used as negative anchors, and prevents pseudo-positive RoIs from being used as negative RoI samples.

5.1.5 Methods with rehearsal

ILOD-Meta

This method uses the ILOD [25] framework, empowering it with the use of exemplars and a meta learning strategy to adjust gradients during training.

RILOD

This paper [59] uses the same set of losses of [26], distilling the knowledge on the three parts of the Faster-rcnn architecture. It uses exemplars, particularly it studies the effect of different sampling strategies. Moreover it uses data downloaded using search engines to increase the dataset size.

OWOD

This work [64] is focused on open world object detection, therefore to recognize unknown classes. This method uses a normal rehearsal strategy and interestingly, although it is not the main task it achieve good results on incremental object detection.

5.2 Incremental Instance Segmentation

Very few works exist on incremental instance segmentation, particularly, to the best of my knowledge the only one is: "Class-incremental instance segmentation via multi-teacher networks" [65]. This paper is based on YOLACT architecture [66] (1-stage) and uses two teachers to perform knowledge distillation. Results are promising, on the other hand it is not fair to compare our work with the latter since the architecture itself is deeply different and is tailored on real time instance segmentation. This paper proposes an approach already used in incremental learning tasks: using two different teachers to distill the knowledge. The first one is a normal teacher network trained on old classes (**F**ormer **T**eacher **N**etwork), the second one is a network that is trained just on the new classes (**C**urrent **T**eacher **N**etwork). In this way the knowledge is constrained by a model that knows just old classes but also from a model that is able to recognize just new classes; in this way this paper correctly balance the effect of distillation loss, avoiding to constraining too much the network to old classes.

Chapter 6

Method

6.1 Notation and Preliminaries

The goal of object detection is to train a model able to detect objects, i.e. localize and classify them by producing a rectangular box and a class label. In this work, we focus on detection model in the R-CNN [12, 13, 9] family. A detection model, denoted $\mathcal{F}_\theta$ with parameters θ, is composed by three components: a feature extractor $\mathcal{F}_{FE}$, a region proposal network (RPN) $\mathcal{F}_\theta^{RPN}$, and a classification head $\mathcal{F}_\theta^{RCN}$. Denoting with x an image, the feature extractor outputs a dense feature map. The map is forwarded to the RPN that is devoted to produce a set of N rectangular regions of interest (RoIs), and associates to it a binary objecteness score. The classification head then applies the N RoIs to the feature map and classifies them, generating class probabilities for each RoI ($p \in \mathbb{R}^{|\mathcal{C}|+1}$), indicating with $\mathcal{C}$ the number of classes, and the final rectangular boxes $r \in \mathbb{R}^{4|\mathcal{C}|}$, one for each class. Notably, the classifier also generates a class score for the background, suggesting that there are no objects within the relative RoI. The training process in incremental learning for object detection (ILOD) is divided into many *learning phases*, each of which introduces a new collection of classes to be detected. Formally, in the t-th training step, a detection model $\mathcal{F}_{\theta^t}$ is updated to learn a set of classes $\mathcal{Y}^t$ using a training set $\mathcal{D}^t$. We note that, while an image in the training set $\mathcal{D}^t$ can contain multiple objects of different classes, *only* annotations for classes in $\mathcal{Y}^t$ are provided. Moreover, at training step t the old training sets $\{\mathcal{D}^0, \dots, \mathcal{D}^{t-1}\}$ are not available. After the t-th step, the model $\mathcal{F}_{\theta^t}$ should produce prediction for all the classes seen so far, i.e. its output should consider the classes in $\mathcal{C}^t = \cup_{t'=1}^{t} \mathcal{Y}^{t'}$. We note that $\mathcal{Y}^i \cap \mathcal{Y}^j = \emptyset$ for any $i, j \leq t$ and $i \neq j$.

6.2 MMA: Modeling the Missing Annotations

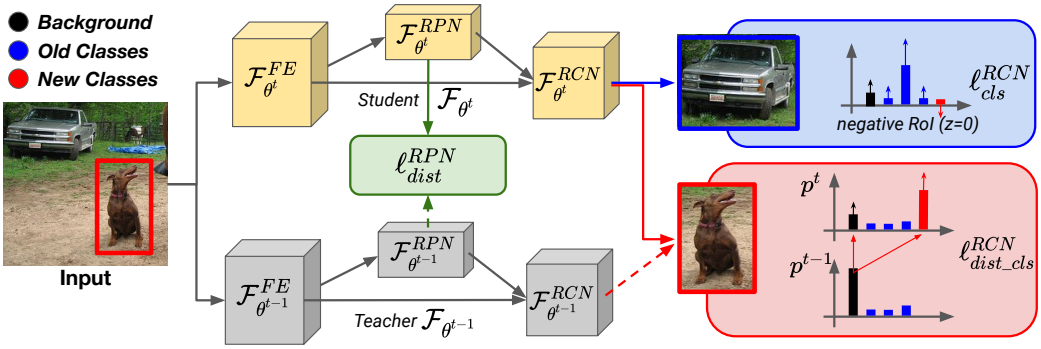

Figure 6.1. The blue box illustrates how unbiased cross entropy loss behaves when the RoI is negative (i.e. when the RoI is empty): the model maximizes the likelihood of having either the background or an old class. We demonstrate the effect of the unbiased distillation loss on the classification output for a new class region in the red box: it associates the teacher's background with either the student's background or a new class. Finally, in green, the RPN distillation loss is indicated.

Despite its strength, Faster R-CNN is not well-suited to updating its weights in order to learn new classes. In particular, fine-tuning the model on $\mathcal{D}^t$ drive the model to completely forget what it has learnt, suffering catastrophic forgetting [17]. To alleviate it, previous works [25, 26, 29, 27, 28] introduced knowledge distillation [67, 20]: at the training step t, the *student* model $\mathcal{F}_{\theta^t}$ is forced to mimic the output of the *teacher* model $\mathcal{F}_{\theta^{t-1}}$, i.e. the model frozen after the previous training step.

Despite alleviating the forgetting, previous works did not consider the missing annotation issue. Despite their efforts to alleviate forgetting, prior works did not address the issue of missing annotations. At time step t, the dataset $\mathcal{D}^t$ annotates just the classes in $\mathcal{Y}^t$; other objects in the image, whether they belong to previous or future classes, are not marked. Any RoI that does not match a ground truth annotation is assigned to the background using the normal detection procedure. This raises two points: i) if the RoI contains an object from a previous class, the model learns to predict it as background, exacerbating forgetting; (ii) if the RoI contains an object from a future class, the model learns to consider it as background, making it more difficult to learn new classes when presented. The missing annotation issue is similar to the background shift presented in [30] in the context of incremental learning for semantic segmentation. In the following, we show how to adapt

the equations proposed by [30] in incremental learning for object detection.

6.2.1 Unbiased Classification Loss.

The classification loss ℓ_{cls}^{RCN} in the Faster R-CNN has the goal to push the network to predict the correct class label for the RoIs. In depth, given a sampled set of N RoIs, generated by the RPN, each one is matched with a ground truth label (positive RoI) or with the background (negative RoI). It is computed as:

$$\ell_{cls}^{RCN} = \frac{1}{N}\sum_{i=1}^{N} z_i(\sum_{c\in\mathcal{C}^{\sqcup}} \bar{y}_i^c \log(p_i^c)) + (1 - z_i)\log(p_i^b), \tag{6.1}$$

where z_i is 1 for a positive proposal and 0 otherwise, $\bar{y}_i$ is the one-hot class label (1 for the ground truth class, 0 otherwise), and p_i^b indicates the probability for the background class for the i-th RoI.

Since it was built for offline object recognition, the 6.1 ignores the fact that the ground truth contains only information about novel classes. The difficulty is that all other objects in the image are not associated with any ground truth and are thus deemed to have a negative RoI, for which the model learns to predict the background class. This issue is especially critical for objects of older classes, since it causes the model to forget the object's actual class and replace it with the background class, resulting in a severe catastrophic forgetting. To address this concern, we amend 6.1 as follows: To avoid this issue, we modify 6.1 as follows:

$$\begin{aligned}\ell_{cls}^{RCN} = \frac{1}{N}\sum_{i=1}^{N} z_i(\sum_{c\in\mathcal{Y}^t} \bar{y}_i^c \log(p_i^c)) + \\ (1 - z_i)\log(p_i^b + \sum_{o\in\mathcal{C}^{t-1}} p_i^o),\end{aligned} \tag{6.2}$$

where $\mathcal{Y}^t$ are the new classes at t and $\mathcal{C}^{t-1}$ are all the classes seen before t. Using 6.2 the model learns new classes on the positive RoIs ($z_i = 1$) while preventing the background to prevail on the old classes: instead of pushing the background class on every negative RoI ($z_i = 0$), as in 6.1, it forces the model to predict either the background or any old class by maximizing the sum of their probabilities.

6.2.2 Unbiased Knowledge Distillation.

A common solution [26, 29, 27, 28] to avoid forgetting is to add two knowledge distillation loss terms to the training objective:

$$\ell = \ell_{faster} + \lambda^1 \ell_{dist}^{RCN} + \lambda^2 \ell_{dist}^{RPN}, \tag{6.3}$$

where λ^1, λ^2 are hyper-parameters.

The goal of ℓ_{dist}^{RCN} is to preserve the knowledge about old classes on the classification head. Previous works [25, 26] force the student model to output classification scores and box coordinates for old classes close to the teacher using an L2 loss. However, they ignore the missing annotations, i.e. the new classes have been seen in previous steps but, since they had been observed without annotations, they have been matched to the background class. The teacher would predict an high background score for new classes RoIs, and forcing the student to mimic its behavior would make more difficult to learn new classes, contrasting the classification loss. In order to model the missing annotations, we formulate the distillation loss as:

$$\ell_{dist}^{RCN} = \frac{1}{N} \sum_{i=1}^{N} \ell_{dist\_cls}^{RCN}(i) + \ell_{smooth\_l1}(r_i^t, r_i^{t-1}), \tag{6.4}$$

$$\begin{aligned} \ell_{dist\_cls}^{RCN}(i) = \frac{1}{|\mathcal{C}^{t-1}| + 1} (p_i^{b,t-1} \log(p_i^{b,t} + \sum_{j \in \mathcal{Y}^t} p_i^{j,t}) \\ + \sum_{c \in \mathcal{C}^{t-1}} p_i^{c,t-1} \log(p_i^{c,t})), \end{aligned} \tag{6.5}$$

where $p_i^{k,t-1}$, r_i^{t-1} and $p_i^{k,t}$, r_i^t indicates, respectively, the classification and regression output for the proposal i and class k of the teacher and student model, and b is the background class. While the second term of 6.4 has been used in previous works [25, 26] and considers the box coordinates, we propose to modify the first term that is crucial to handle the classification scores.

To model the missing annotations, 6.5 employs all of the student model's class probabilities to match the teacher model's: the old classes $\mathcal{C}^{t-1}$ are preserved between the student and teacher models, while the teacher's background $p_i^{b,t-1}$ is associated with either a new class or the student's background. When the teacher predicts a high background probability for a RoI belonging to a new class, the student is not required to copy the teacher's conduct; instead, the student can consolidate its new knowledge and correctly predict the class.

6.2.3 Distillation on Region proposal network

ℓ_{dist}^{RPN} goal is to prevent forgetting on the RPN output. Since annotations for old classes are not usable, the RPN learns to predict an high objectness score only on RoIs belonging to new classes. To force the RPN to maintain an high objectness score for regions belonging to old classes, we use the loss proposed by [26]. The student is compelled to imitate the teacher only in locations associated with previous classes., i.e. where the teacher score is greater than the student one. Considering A regions, we compute ℓ_{dist}^{RPN} as:

$$\begin{aligned}\ell_{dist}^{RPN} = \frac{1}{A}\sum_{i=1}^{A} 1_{[s_i^t \geq s_i^{t-1}]}||s_i^t - s_i^{t-1}||+ \\ 1_{[s_i^t \geq s_i^{t-1}+\tau]}||\omega_i^t - \omega_i^{t-1}||,\end{aligned} \tag{6.6}$$

where s_i^t is the objectness score and ω_i^t the coordinates of $\mathcal{F}_{\theta^t}^{RPN}$ on the i-th proposal, $||\cdot||$ is the euclidean distance, τ is an hyperparameter, and 1 is the indicator function equal to 1 when the condition on the brackets is verified and 0 otherwise. Note that when $s_i^t > s_i^{t-1}$, the teacher produces an objectness score greater then the student and the proposal is probably containing an old class. In contrast, when $s_i^t \geq s_i^{t-1}$, the proposal is most likely to belong to a new class, and requiring the student to replicate the teacher score may bring errors that impair performance in new classes.

6.2.4 Feature Attention Distillation

In [26] the authors use a feature distillation on the output of the Resnet backbone, this loss aims to maintain unaltered the feature space from the old to the new model. Since one of the main problems in Object detection ICL is the too constrictive distillation, then this loss increases that constraint, blindly distillate all the feature map.
On the other hand since objects are not present in the whole image it could be useful to distill just some parts of the feature map. Therefore in our method we use the activation map of the previous model to weight the feature distillation.
The old model should be activated in the portion of the feature map where the object is, hence calculating the following matrices (respectively channel and spatial attention map), we can have a measure of pixel importances in

$\mathcal{F}$, inspiring by [68] [69]

$$\begin{aligned} S^{att}(t) &= HW \cdot softmax[\frac{1}{C} \cdot \sum_{i}^{H} \sum_{j}^{W} |\mathcal{F}_{ij}^{t}|)/T] \\ C^{att}(t) &= C \cdot softmax[(\frac{1}{HW} \cdot \sum_{c}^{C} \mathcal{F}_{c}^{t})/T] \end{aligned} \tag{6.7}$$

where $\mathcal{F}^t$ is the feature map of the model trained at step t and T is a temperature to avoid high activations to deviate the softmax normalization and H,W,C are respectively the feature map height, width and channels. These two matrices represents the weights for the distillation loss that can be written as follows:

$$L_{act} = \sum_{i}^{H} \sum_{j}^{W} \sum_{c}^{C} S^{att}(t-1)_{ij} C^{att}(t-1)_{c} \cdot (\mathcal{F}_{ijc}^{t} - \mathcal{F}_{ijc}^{t-1}) \tag{6.8}$$

Moreover, since the activation map used is calculated on the old model, there is the need to add a constraint to avoid that the new model diverges too much in feature activation. Therefore another term to the distillation loss is added:

$$L_{act}^{dist} = |S^{att}(t) - S^{att}(t-1)| + |C^{att}(t) - C^{att}(t-1)| \tag{6.9}$$

Therefore the total attentive loss can be written as:

$$L_{att} = L_{act} + L_{act}^{dist} \tag{6.10}$$

In order to give a practical example of the work made by this loss the following images represent the activation maps on some images of VOC2007 dataset [11]. In these images the colored parts are the most activated by the old model attention map. Particularly the "hottest" zones will be weighted more in the feature distillation than the coldest one, allowing the model to distill more the correct parts of the image.

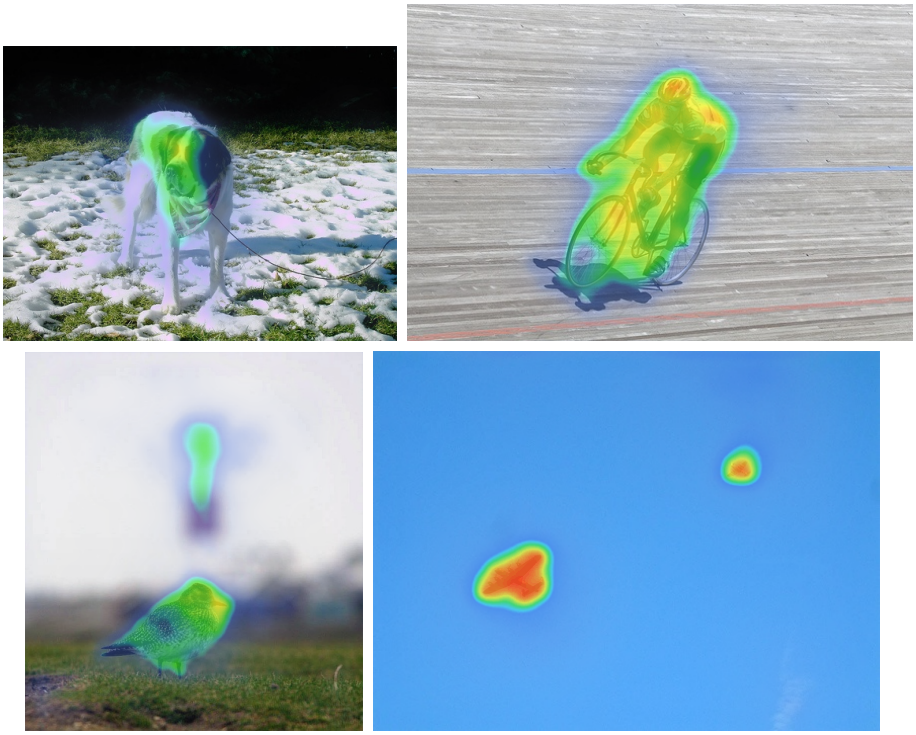

Figure 6.2. Example images taken from VOC dataset representing the activation maps of the old model

6.3 Extension to Instance Segmentation

The goal of instance segmentation is to produce an accurate pixel-wise mask for each object in the image instead of a coarse bounding box. To produce masks we rely on Mask R-CNN [9], that extends the Faster R-CNN introducing a mask head $\mathcal{F}_\theta^{MASK}$. It generates an additional binary segmentation mask with shape for each RoI $\mathcal{C} \times h \times w$, where $\mathcal{C}$ is the number of classes and h,w is the mask resolution. To train the mask head, [9] introduces an additional loss term that is summed to the total multi-task loss. Formally, Mask R-CNN objective is:

$$\ell_{mask} = \ell_{faster} + \ell_{cls}^{MASK}, \tag{6.11}$$

where ℓ_{cls}^{MASK} is a average per-pixel binary cross-entropy loss between the $\mathcal{F}_\theta^{MASK}$ output and the binary mask of the ground truth class.

Despite the method presented in 6.2 already accounts for forgetting on the detection head, by applying 6.11 we incur the risk to forget how to segment past objects while learning the new ones. For this reason, we further extend

6.3 to add a knowledge distillation term on the mask head. Formally, in instance segmentation we employ the following training objective:

$$\ell = \ell_{mask} + \lambda_1 \ell_{dist}^{RCN} + \lambda_2 \ell_{dist}^{RPN} + \lambda_3 \ell_{dist}^{MASK}, \tag{6.12}$$

where λ_1, λ_2, λ_3 are hyper-parameters.

ℓ_{dist}^{MASK} has the goal of keeping the segmentation mask for old classes close to the output of the teacher model. In particular, we employ a per-pixel binary cross-entropy loss between the teacher model masks and the student ones. Formally, denoting as $m_{c,i}^t$ the segmentation mask produced by $\mathcal{F}_{\theta^t}^{MASK}$ for the class c at pixel i, we compute

$$\begin{aligned} \ell_{dist}^{MASK} = \frac{1}{|I||\mathcal{C}^{t-1}|} \sum_{i \in I} \sum_{c \in \mathcal{C}^{t-1}} & m_{c,i}^{t-1} \log(m_{c,i}^t) + \\ & (1 - m_{c,i}^{t-1}) \log(1 - m_{c,i}^t), \end{aligned} \tag{6.13}$$

where I is the set of pixels and $|I| = h \times w$. We note that 6.13 is computed only on the segmentation masks belonging to old classes in $\mathcal{C}^{t-1}$, while the masks of the new ones are not distilled.

Chapter 7

Experiments

7.1 Experimental Protocol

We evaluate MMA on the Pascal dataset. In particular, following previous works, we employ PASCAL-VOC 2007 [11] for object detection. It is a widely used benchmark that includes 20 foreground object classes and consists in 5K images for training and 5K for testing. For instance segmentation, we employed Pascal SBD 2012 [70], that contains the same set of 20 classes but also reports the instance segmentation annotations. We used the standard split of Pascal SBD 2012, using 8498 images for training and 2857 for evaluation. Following [25], for both object detection and instance segmentation the experimental protocol is as follows: each training step contains all the images that have at least one bounding box of a novel class. We remark that at each training step we have only labels for bounding boxes of novel classes, while all the other objects that appear in the image, either belonging to past or future classes, are not annotated. This is a very realistic configuration because it makes no assumptions about the items in the photographs and significantly minimizes the amount of annotation necessary at each incremental step.

7.2 Implementation Details

For object detection, we followed previous works [26, 29, 56, 71, 72, 60] and we use the Faster R-CNN architecture with a ResNet-50 backbone. Similarly, for instance segmentation, we employ the Mask R-CNN [9] architecture with ResNet-50 backbone. Both backbones are initialized using the ImageNet pretrained model [73]. We used the same training protocol of [25, 26] but

we increased the batch size from 1 to 4 to reduce the time required for training, scaling accordingly the learning rate and number of iterations. In particular, for object detection we train the network with SGD, weight decay 10^{-4} and momentum 0.9. We use an initial learning rate of $4 \cdot 10^{-3}$ for the first learning step and $4 \cdot 10^{-4}$ in the followings. We performed 10K iterations when adding 5 or 10 classes, while we trained for 2.5K when learning only one or two classes. We apply the same data augmentation of [25, 26]. We set λ_2 equal to 0.1, 0.5, and 1 when adding 10 classes, 5, and 1 or 2 classes, respectively. λ_1, λ_3 are set to 1.

7.3 Evaluation Metrics

AP (**A**verage **P**recision) is a widely used metric in object detection to evaluate how good the model is at both locating and classifying a given object category. **mAP** is the average of the single APs computed for each class. Before giving a formal definition of AP, it is necessary to step back and first define the concepts of precision, recall and precision-recall curve.

Definition 7.3.1 (Precision-Recall-F1) *Let c the target class, TP the number of true positive (i.e. number of correct predictions of the model for class c), FP the number of false positive (i.e. number of wrong predictions of the model for class c), FN the number of false negative (i.e. number of wrong predictions of the model for class different from c), then:*

$$Precision = \frac{TP}{TP + FP} \tag{7.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{7.2}$$

$$F1 = 2 \cdot \frac{Precision * Recall}{Precision + Recall} \tag{7.3}$$

In other words, *precision* measures how accurate are the predictions of the model, i.e. the percentage of predictions are correct; the *recall* measures how good the model finds all the positives. So the more the model tends to predict each sample as positive, the higher the recall will be, but the lower the precision will be as the number of false positive tends to increase. For this reason, the metric *F1* is introduced, which summarizes both *precision* and *recall* in a single metric and it is maximized if and only if both *precision*

and *recall* are maximized (the perfect model).
Due to the importance of both precision and recall, *precision-recall curve* shows the tradeoff between the precision and recall values for different thresholds. Let's create a dummy example to better understand how to *precision-recall curve* is plotted.

Table 7.1. Precision-Recall values [1]

Correct?	Precision	Recall
True	1	0.2
True	1	0.4
False	0.67	0.4
False	0.5	0.4
False	0.4	0.2
False	0.5	0.6
True	0.57	0.8

In table 7.3 we report the predictions of the model, sorted by confidence score in a descending order. Note that recall values increase as we go down the predictions, on the other hand precision has a zigzag pattern: it goes down with false positives and goes up again with true positives. Let's plot the precision against the recall value to see this zig-zag pattern (7.3).

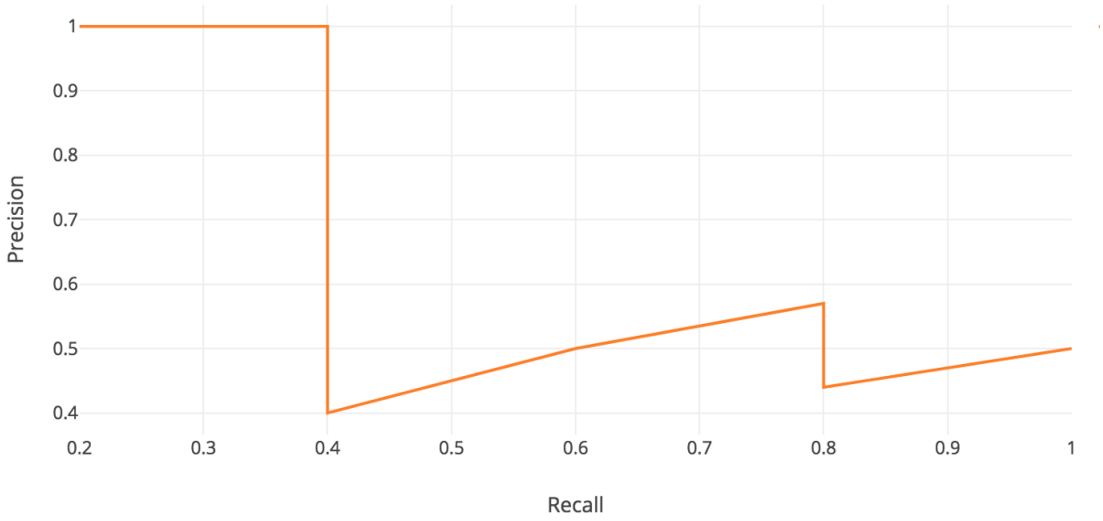

Figure 7.1. Precision-Recall curve [1]

Now we can define the average precision as:

Definition 7.3.2 (Average Precision) *Let $p : \mathbb{R} \to \mathbb{R}$ a function that calculates the precision value at a specific recall level, then:*

$$AP = \int_0^1 p(r)dr \tag{7.4}$$

In other words, the average precision (**AP**) is finding the area under the precision-recall curve defined above (7.3), so it is a way to summarize the precision-recall curve into a single value representing the average of all precisions. Note that as precision and recall values are in between 0 and 1, also the average precision will be in the range [0,1].

Finally, we can define the mAP as:

Definition 7.3.3 *Let $\mathcal{C}$ the set of target classes, then:*

$$mAP = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} AP^c \tag{7.5}$$

Each prediction of the model is considered correct when $IoU \geq IoU_{tresh} = 0.5$.

To fairly compare our method with others for VOC 2007 [11] we use $mAP_{0.5}$, meaning that the Iou_{tresh} is set to 0.5. For the same reason on Pascal SBD [74] we use $mAP_{0.5...0.95}$. It is an extension of the mAP and it is calculated as follows:

$$mAP_{0.5...0.95} = \frac{1}{|\mathcal{T}|} \sum_{th} mAP_{th} \tag{7.6}$$

where $\mathcal{T}$ is the set of thresholds. Particularly thresholds goes from 0.5 to 0.95 with steps of 0.05. Since in this case the mAP is calculated for semantic segmentation, the definition of IoU must be selected accordingly as in **??**.

7.4 Object Detection Results

As done by previous works [56, 29, 60, 25, 26], for incremental object detection we evaluate our method considering experimental settings adding a different number of classes in one or multiple training steps. We report adding 10 (*10-10*), 5 (*15-5*) or 1 (*19-1*) class in a single incremental step and performing two incremental steps adding 5 classes (*10-5*), five steps adding two classes (*10-2*) and either ten (*10-1*) or five (*15-1*) steps adding one class. As in previous works, we split the classes following the alphabetical order.

7.4.1 Single-step incremental settings (10-10, 15-5, 19-1)

Results are reported in the tables below. The *Avg* metric equally weights new and old classes averaging their aggregated mAP. We benchmark MMA against previous works reporting the results on the same settings. We compare either with methods using rehearsal [72, 71, 60] or not using them [56, 29, 25, 26].
We emphasize that the previous approaches cannot be accurately compared to MMA, as we do not use any replay memory to store previous samples. Furthermore, for a fair comparison we report ILOD [25] and Faster ILOD [26] using our same architecture and training protocol. Finally, we report two baselines: the joint training upper bound, where the architecture is trained using the whole dataset and all the annotations, and the fine-tuning, where the architecture is trained on the new data using the standard Faster-RCNN pipeline without using any regularization strategy.

Table 7.2. mAP@0.5 results on single incremental step (10-10 scenario) on Pascal-VOC 2007. Methods with † come from reimplementation. Methods with * use exemplars.

	10-10			
Method	**1-10**	**11-20**	**1-20**	**Avg**
Joint Training	74.7	75.7	75.2	75.2
Fine-tuning	9.5	62.5	36.0	36.0
ILOD (Fast R-CNN) [25]	63.2	63.1	63.2	63.2
ILOD (Faster R-CNN) [25] †	65.3	69.2	53.0	61.1
61.1				
Faster ILOD [26]	69.8	54.5	62.1	62.1
Faster ILOD [26] †	71.1	52.3	61.7	61.7
PPAS [75]	63.5	60.0	61.8	61.8
MVC [56]	66.2	66.0	66.1	66.1
OREO* [72]	60.4	68.8	64.6	64.6
OW-DETR* [71]	63.5	67.9	65.7	65.7
ILOD-Meta* [60]	68.4	64.3	66.3	66.3
MMA	69.3	63.9	**66.6**	**66.6**

As can be noted in 7.2 7.3 7.4, fine-tuning results in a significant decline in performance when compared to previous classes, showing that catastrophic forgetting is a problem that needs to be addressed. While earlier research has improved performance by addressing the forgetting problem, MMA surpasses all previous methods, including those that employ exemplars to prevent forgetting, confirming the validity of our methodology.
In particular, when comparing with ILOD [25] and Faster ILOD [26], we note that our method achieve comparable performance on old classes but outperforms them on the new classes, outperforming them of 1% on both 19-1 and 15-5, and even by 10% on the 10-10 setting. We believe that the improvement is mostly attributable to the unbiased distillation loss, which eliminates incoherent training targets, hence improving performance.
Comparing MMA to previous state-of-the-art, we note that it outperforms the competitive rehearsal strategies in every setting. On the 19-1 setting, MMA outperforms the ILOD-Meta by 0.5% considering equally every class (*1-20*) and by 1.1% OW-DETR when considering equally old and new classes (*Avg*). Similarly, in the 15-5 and 10-10 settings, MMA outperforms the best rehearsal method by 0.9% and 0.3% on all the classes 0.7% and by 0.3% on

Table 7.3. mAP@0.5 results on single incremental step on (15-5 scenario) Pascal-VOC 2007. Methods with † come from reimplementation. Methods with * use exemplars.

	15-5			
Method	**1-15**	**16-20**	**1-20**	**Avg**
Joint Training	75.3	73.6	75.2	74.4
Fine-tuning	14.2	59.2	25.4	36.7
ILOD (Fast R-CNN) [25]	68.3	58.4	65.9	63.4
ILOD (Faster R-CNN) [25] †	72.5	58.0	68.9	65.3
Faster ILOD [26]	71.6	56.9	67.9	64.3
Faster ILOD [26] †	73.5	55.6	69.1	64.6
MVC [56]	69.4	57.9	66.5	63.7
OREO* [72]	71.8	58.7	68.5	65.2
OW-DETR* [71]	72.2	59.8	69.1	66.0
ILOD-Meta* [60]	71.7	55.9	67.8	63.8
MMA	73.0	60.5	**69.9**	**66.7**

Table 7.4. mAP@0.5 results on single incremental step (19-1 scenario) on Pascal-VOC 2007. Methods with † come from reimplementation. Methods with * use exemplars.

	19-1			
Method	**1-19**	**20**	**1-20**	**Avg**
Joint Training	75.3	73.6	75.2	74.4
Fine-tuning	12.0	62.8	14.5	37.4
ILOD (Fast R-CNN) [25]	68.5	62.7	68.3	65.6
ILOD (Faster R-CNN) [25] †	70.3	65.2	70.0	**67.8**
Faster ILOD [26]	68.9	61.1	68.5	65.0
Faster ILOD [26] †	70.9	64.3	70.6	67.6
PPAS [75]	70.5	53.0	69.2	61.8
MVC [56]	70.2	60.6	69.7	65.4
OREO* [72]	69.4	60.1	68.9	64.7
OW-DETR* [71]	70.2	62.0	69.8	66.1
ILOD-Meta* [60]	70.9	57.6	70.2	64.2
MMA	71.1	63.4	**70.7**	67.2

the *Avg* metric, respectively.

7.4.2 Multi-step incremental settings (10-5, 10-2, 15-1, 10-1)

While doing a single training step is beneficial for assessing the ability to prevent catastrophic forgetting, a more realistic scenario involves executing numerous incremental steps to add additional classes. In this section, we analyze the behavior of MMA against three baselines: fine-tuning, ILOD [25], Faster ILOD [26], all implemented following our experimental protocol. We report the results for the four considered settings in 7.8, showing the mAP% over multiple incremental steps and 7.5, where the results after the last incremental step are reported. 7.8 further reports the average performance across multiple steps *Avg-S*.

We can observe that doing numerous incremental steps is difficult and that current solutions perform poorly in comparison to single step instances. In particular, fine-tuning the network on fresh data without applying any approach to avoid forgetting causes the network to totally forget the old classes, resulting in performance close to 0% on the old classes in the final step.
ILOD [25] and Faster ILOD [26] substantially alleviate catastrophic forgetting, leading to better results both on old and new classes. However, when comparing with MMA, we see that both ILOD and Faster ILOD achieve worse results. In particular, after the last step, evidently MMA obtain better performances on novel classes: +2.4% o 10-5, +3.3% on 10-2, +6.3% on 15-1, and 6.8% on 10-1 wrt. the best among the baselines. Furthermore, MMA also obtains comparable or greater performance than previous methods on the old classes.
Overall, MMA outperforms the best among ILOD and Faster ILOD by 0.9% on 10-5, 2.7% on 10-2, 2.8% on 15-1, and 6.5% on the 10-1 setting. We observe that the improvement is greater as more classes are added, indicating that our strategy is more suited to executing several incremental stages. Considering the trend over multiple training steps in 7.5, we note that MMA is always comparable or better than previous methods. In particular, it is remarkable that MMA largely outperforms the other methods when increasing the number of training steps, as shown in the 10-1 setting.

Table 7.5. mAP@0.5% results on multi incremental steps on Pascal-VOC 2007. Methods with † come from reimplementation.

	10-5			
Method	**1-10**	**11-20**	**1-20**	**Avg-S**
Joint Training	74.7	75.7	75.2	75.2
Fine-tuning	6.6	28.3	17.4	21.8
ILOD (Faster R-CNN) [25] †	67.2	59.4	63.3	65.2
Faster ILOD [26] †	68.3	57.9	63.1	65.5
MMA	66.7	61.8	**64.2**	**67.3**

Table 7.6. mAP@0.5% results on multi incremental steps on Pascal-VOC 2007. Methods with † come from reimplementation.

	10-2			
Method	**1-10**	**11-20**	**1-20**	**Avg-S**
Joint Training	74.7	75.7	75.2	75.2
Fine-tuning	5.2	12.3	8.8	16.7
ILOD (Faster R-CNN) [25] †	62.1	49.8	55.9	62.2
Faster ILOD [26] †	64.2	48.6	56.4	62.8
MMA	65.0	53.1	**59.1**	**63.8**

Table 7.7. mAP@0.5% results on multi incremental steps on Pascal-VOC 2007. Methods with † come from reimplementation.

	15-1			
Method	**1-15**	**16-20**	**1-20**	**Avg-S**
Joint Training	76.8	70.4	75.2	73.5
Fine-tuning	16.7	0.0	8.0	2.4
ILOD (Faster R-CNN) [25] †	65.6	47.6	60.2	65.8
Faster ILOD [26] †	66.9	44.5	61.3	67.1
MMA	68.3	54.3	**64.1**	**67.5**

Table 7.8. mAP@0.5% results on multi incremental steps on Pascal-VOC 2007. Methods with † come from reimplementation.

	10-1			
Method	**1-10**	**11-20**	**1-20**	**Avg-S**
Joint Training	74.7	75.7	75.2	75.2
Fine-tuning	0.0	4.6	2.3	8.6
ILOD (Faster R-CNN) [25] †	52.9	41.5	47.2	59.1
Faster ILOD [26] †	53.5	41.0	47.3	60.4
MMA	59.2	48.3	**53.8**	**62.4**

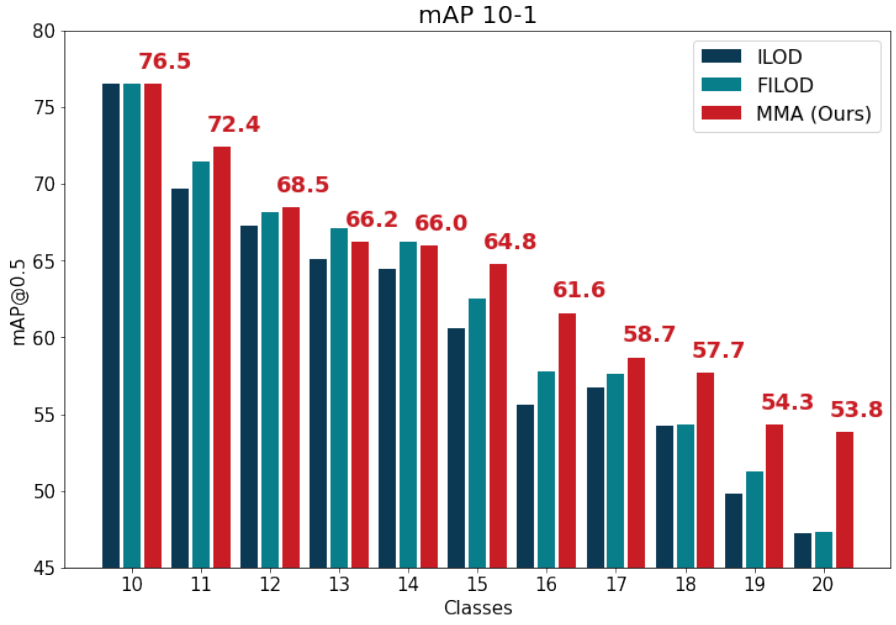

Figure 7.2. mAP% results on multiple incremental steps (10-1 scenario) on Pascal-VOC 2007.

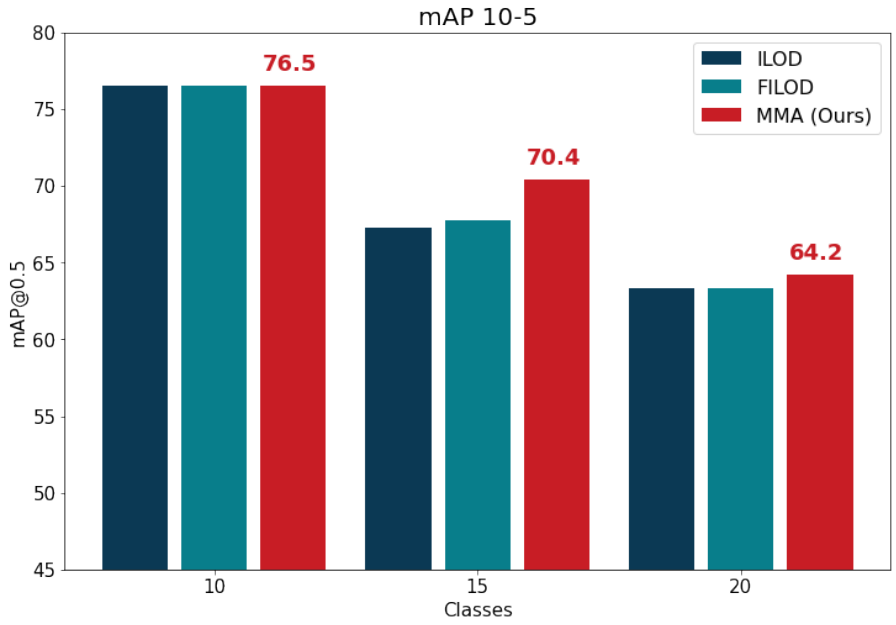

Figure 7.3. mAP% results on multiple incremental steps (10-2 scenario) on Pascal-VOC 2007.

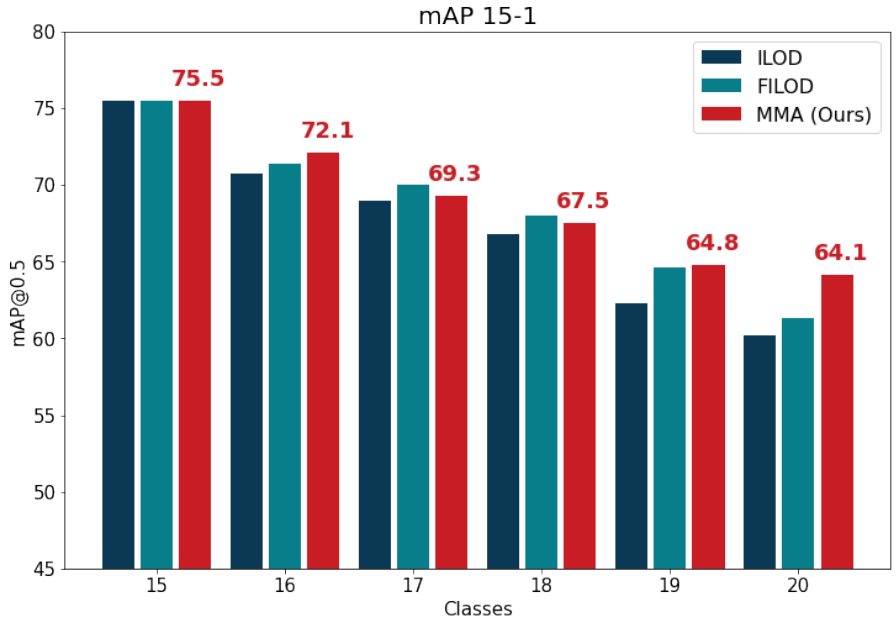

Figure 7.4. mAP% results on multiple incremental steps (15-1 scenario) on Pascal-VOC 2007.

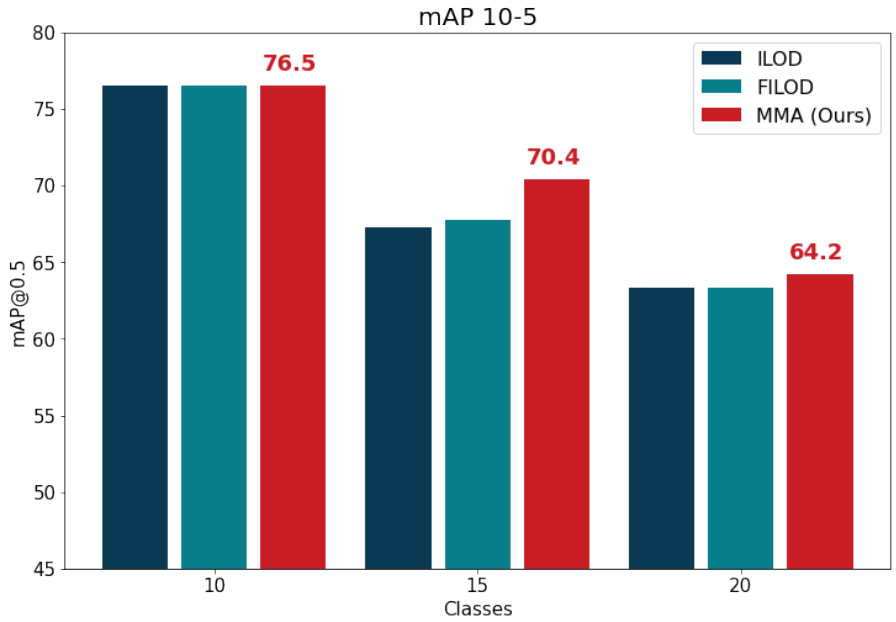

Figure 7.5. mAP% results on multiple incremental steps (10-5 scenario) on Pascal-VOC 2007.

7.5 Instance Segmentation Results

Following the protocol used in incremental object detection, we evaluate our method considering two experimental settings: adding one (*19-1*) and five (*15-5*) classes in a single training step. As with object detection, we use the dataset's alphabetical order. We report the mAP averaged across 11 IoU criteria ranging from 0.5 to 0.95, with a 0.05 step, in accordance with normal practice for instance segmentation.
We compare MMA with fine-tuning, fine-tuning using the unbiased classification loss (6.2), ILOD [25] and Faster ILOD [26]. For all the methods we employ the same architecture and hyper-parameters. 7.9 shows the results for the 19-1 and 15-5 settings, reporting the average mAP of new and old classes separately, the average over all classes, and the average of new and old classes (*Avg*), weighting them equally.
We can see that fine-tuning shows a substantial forgetting on old classes, both on the 19-1 and 15-5 settings. Introducing the unbiased classification loss (6.2 helps with forgetting, the results are still poor for older classes, indicating that a way to prevent forgetting is required.
ILOD and FasterILOD, in fact, improve the performances on old classes. However, forgetting is prevented at the cost of a decrease in performance on novel classes: they both loses nearly 8% on the 19-1 and 5% on the 15-5 with respect to fine-tuning. Differently, employing our proposed MMA we clearly improve the performance, preventing forgetting while showing good performance on novel classes. In particular, ILOD and Faster ILOD, MMA obtains, on new classes, nearly +5% and +3%, respectively on 19-1 and 15-5, while showing comparable performance on old classes.
Considering the extended version of MMA (MMA + ℓ_{dist}^{MASK}), it slightly improves the performance on old classes MMA, while obtaning comparable results on the new ones. Overall, it obtains 41.1% and 38.2% on the 19-1 and 15-5, respectively, 0.3% and 0.9% better than MMA. Interestingly, we observe that even without any mask head regularization (MMA), we may still get acceptable segmentation performance. This is because classes on the mask head are not competitive, as the mask head just regresses a binary segmentation mask, while the classification head predicts the class, like in regular Faster R-CNN.
By and large, MMA and its extension beat other baselines in instance segmentation, demonstrating a favorable trade-off between learning new classes and avoiding forgetting existing ones.

Table 7.9. mAP@(0.5,0.95)% results of incremental instance segmentation on Pascal-VOC 2012.

	19-1				15-5			
Method	**1-19**	**20**	**1-20**	**Avg**	**1-15**	**16-20**	**1-20**	**Avg**
Joint Training	40.4	54.1	41.1	47.2	41.0	41.2	41.1	41.1
Fine-tuning	6.7	46.3	8.7	26.5	1.9	35.3	10.2	18.6
Fine-tuning w/ 6.2	12.5	47.5	14.3	30.0	13.0	35.5	18.6	24.2
ILOD [25]	40.1	38.3	40.0	39.2	39.2	30.8	37.1	35.0
Faster ILOD [26]	40.6	38.1	40.4	39.3	39.4	30.3	37.1	34.8
MMA	40.6	43.0	40.8	41.8	38.2	33.7	37.1	35.9
MMA + ℓ_{dist}^{MASK}	41.0	42.8	**41.1**	**41.9**	40.2	32.2	**38.2**	**36.2**

7.6 Ablation Study on the importance of each component

In Table 7.10 we report a detailed analysis of our contributions, considering 15-5 setting in incremental object detection. We ablate each proposed component: the unbiased classification loss (6.2), the classification head knowledge distillation loss (ℓ_{dist}^{RCN}), the use of the RPN distillation loss (ℓ_{dist}^{RPN}), and finally, the use of a feature distillation loss, as proposed in [26] compared to the attentive distillation loss described in 6.2.4. The first row indicates fine-tuning the network on the new data, without applying any regularization. It is worth noting that the performances are subpar in older classes, but are excellent in new ones. Adding the unbiased classification, the performance on the old classes substantially improves: from 14.2% to 40.0%. This is due to the handling of missing annotation that alleviates forgetting. Introducing the unbiased distillation loss in 6.5 (UKD), the performances improves significantly, both on old classes, reaching 67.3%, and new classes, going from 57.8% to 60.3%. We believe that the distillation loss increases performance on new classes because it teaches the model to distinguish between old and new classes, thus increasing overall precision. We then include the RPN distillation loss to achieve the final MMA model. We see that the performance further improves on old classes, achieving 73.0%, while the performance on the new classes is comparable.

Moreover, we compare the unbiased knowledge distillation in MMA with other possible choices. Inspired by previous works we employ the L2 loss

Table 7.10. Ablation study of the contribution of MMA components in the 15-5 setting. Results are mAP@0.5%. MMA is in green.

6.2	ℓ_{dist}^{RCN}	ℓ_{dist}^{RPN}	ℓ_{dist}^{att}	**1-15**	**16-20**	**1-20**	**Avg**
-	-	-	-	14.2	59.2	25.4	36.7
✓	-	-	-	40.0	57.8	44.4	48.9
✓	UKD	-	-	67.3	60.3	65.6	63.8
✓	l2	✓	-	**73.7**	56.8	69.5	65.3
✓	CE	✓	-	72.8	59.4	69.5	66.1
✓	UKD	✓	-	72.9	59.9	69.6	66.4
✓	UKD	✓	✓	73.0	**60.5**	**69.9**	**66.7**

on the normalized classification scores [25, 26] and the cross-entropy (CE) loss between the probability of old classes [20]. Finally, by inserting the ℓ_{dist}^{att} we note that the performances on old classes remain unvaried, while the one on the new classes continue growing leading to the best result in the ablation. We see that MMA distillation outperforms them, especially on the new classes, clearly demonstrating that modeling the missing annotations is essential to properly learn them. Overall, MMA achieves on the average of old and new class performance 66.7%, 1.4% and 0.6% more than using the L2 loss or the cross-entropy loss.

Chapter 8

Conclusions and future works

We studied the incremental learning problem in object detection considering an issue mostly overlooked by previous works. In particular, in each training step only the annotation for the classes to learn is provided, while the other objects are not considered, leading to many missing annotations that mislead the model to predict background on them, exacerbating catastrophic forgetting. We address the missing annotations by revisiting the standard knowledge distillation framework to consider non annotated regions as possibly containing past objects. We show that our approach outperforms all the previous works without using any data from previous training steps on the Pascal-VOC 2007 dataset, considering multiple class-incremental settings. Moreover one interesting insight coming from our results is to over-perform methods that use rehearsal memory, demonstrating that incremental learning can be achieved also without using old classes examples.
Finally, we provide a simple extension of our method in the instance segmentation task, showing that it outperforms all the baselines.
Incremental learning is unfortunately not ready to be used in a real practical scenario, but with this thesis and with the current efforts in this field it will be surely improved and optimized. We hope that our work will set a new knowledge distillation formulation for incremental object detection methods. Moreover we expect that the problem of background in incremental learning will be further analyzed in order to find new strategies to completely overcome it, in fact we think it is the main issue harming performances in incremental object detection. We leave extending our formulation to one-stage detectors and to think to a more fine strategy to extend the pipeline

to instance segmentation as a future work.

Bibliography

[1] J. Hui, "map (mean average precision) for object detection," 2018.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[3] Y. Jing and Y. Guanci, "Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer," *Algorithms*, vol. 11, p. 28, 03 2018.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[5] P. Ganesh, "Object detection : Simplified," 2019.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[7] S. K, "Inon-maximum suppression (nms)," 2019.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[9] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[10] J. Cao, R. M. Anwer, H. Cholakkal, F. S. Khan, Y. Pang, and L. Shao, "Sipmask: Spatial information preservation for fast image and video instance segmentation," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV*, (Berlin, Heidelberg), p. 1–18, Springer-Verlag, 2020.

[11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[12] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[14] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[16] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*, pp. 213–229, Springer, 2020.

[17] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[18] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

[19] A. Douillard, M. Cord, C. Ollion, T. Robert, and E. Valle, "Podnet: Pooled outputs distillation for small-tasks incremental learning," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pp. 86–102, Springer, 2020.

[20] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[21] E. Fini, S. Lathuiliere, E. Sangineto, M. Nabi, and E. Ricci, "Online continual learning under extreme memory constraints," in *European Conference on Computer Vision*, pp. 720–735, Springer, 2020.

[22] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *ECCV*, 2018.

[23] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016.

[24] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526,

2017.

[25] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," in *ICCV*, 2017.

[26] C. Peng, K. Zhao, and B. C. Lovell, "Faster ilod: Incremental learning for object detectors based on faster rcnn," 2020.

[27] Y. Hao, Y. Fu, Y.-G. Jiang, and Q. Tian, "An end-to-end architecture for class-incremental object detection with knowledge distillation," in *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2019.

[28] L. Chen, C. Yu, and L. Chen, "A new knowledge distillation for incremental object detection," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, 2019.

[29] W. Zhou, S. Chang, N. Sosa, H. Hamann, and D. Cox, "Lifelong object detection," *arXiv preprint arXiv:2009.01129*, 2020.

[30] F. Cermelli, M. Mancini, S. R. Bulo, E. Ricci, and B. Caputo, "Modeling the background for incremental learning in semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9233–9242, 2020.

[31] U. Michieli and P. Zanuttigh, "Incremental learning techniques for semantic segmentation," in *ICCV-WS*, pp. 0–0, 2019.

[32] U. Michieli and P. Zanuttigh, "Continual semantic segmentation via repulsion-attraction of sparse and disentangled latent representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1114–1124, 2021.

[33] A. Douillard, Y. Chen, A. Dapogny, and M. Cord, "Plop: Learning without forgetting for continual semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4040–4050, 2021.

[34] F. Cermelli, M. Mancini, Y. Xian, Z. Akata, and B. Caputo, "A few guidelines for incremental few-shot segmentation," *arXiv preprint arXiv:2012.01415*, 2020.

[35] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[36] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[37] Z. Yanling, D. Bimin, and W. Zhanrong, "Analysis and study of perceptron to solve xor problem," in *The 2nd International Workshop on Autonomous Decentralized System, 2002.*, pp. 168–173, 2002.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[40] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[41] R. Girshick, "Fast r-cnn," 2015.

[42] T. Wang, R. M. Anwer, H. Cholakkal, F. S. Khan, Y. Pang, and L. Shao, "Learning rich features at high-speed for single-shot object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1971–1980, 2019.

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[44] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[45] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[46] J. Cao, Y. Pang, J. Han, and X. Li, "Hierarchical shot detector," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9705–9714, 2019.

[47] R. Vaillant, C. Monrocq, and Y. Le Cun, "Original approach for the localization of objects in images," *IEE Proceedings: Vision, Image and Signal Processing*, vol. 141, pp. 245–250, Aug. 1994.

[48] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23–38, 1998.

[49] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2014.

[50] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in Neural Information Processing Systems* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[51] C. Gu, J. J. Lim, P. Arbelaez, and J. Malik, "Recognition using regions," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1030–1037, 2009.

[52] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 09 2013.

[53] M. Masana, X. Liu, B. Twardowski, M. Menta, A. D. Bagdanov, and J. van de Weijer, "Class-incremental learning: survey and performance evaluation on image classification," 2021.

[54] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, 1999.

[55] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

[56] D. Yang, Y. Zhou, and W. Wang, "Multi-view correlation distillation for incremental object detection," 2021.

[57] X. Zhao, X. Liu, Y. Shen, Y. Qiao, Y. Ma, and D. Wang, "Revisiting open world object detection," 2022.

[58] M. Acharya, T. L. Hayes, and C. Kanan, "Rodeo: Replay for online object detection," in *BMVC*, 2020.

[59] D. Li, S. Tasci, S. Ghosh, J. Zhu, J. Zhang, and L. Heck, "Rilod: Near real-time incremental learning for object detection at the edge," 2019.

[60] J. Kj, J. Rajasegaran, S. Khan, F. S. Khan, and V. N Balasubramanian, "Incremental object detection via meta-learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1–1, 2021.

[61] N. Dong, Y. Zhang, M. Ding, and G. H. Lee, "Bridging non co-occurrence with unlabeled in-the-wild data for incremental object detection," 2021.

[62] L. Liu, Z. Kuang, Y. Chen, J.-H. Xue, W. Yang, and W. Zhang, "Incdet: In defense of elastic weight consolidation for incremental object detection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2306–2319, 2021.

[63] C. Peng, K. Zhao, S. Maksoud, T. Wang, and B. C. Lovell, "Diode: Dilatable incremental object detection," 2021.

[64] K. J. Joseph, S. Khan, F. S. Khan, and V. N. Balasubramanian, "Towards open world object detection," 2021.

[65] Y. Gu, C. Deng, and K. Wei, "Class-incremental instance segmentation via multi-teacher networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 1478–1486, May 2021.

[66] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "Yolact: Real-time instance segmentation," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9157–9166, 2019.

[67] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[68] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," 2018.

[69] P. Zhao, J. Zhang, W. Fang, and S. Deng, "Scau-net: Spatial-channel attention u-net for gland segmentation," *Frontiers in Bioengineering and Biotechnology*, vol. 8, 2020.

[70] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *International Conference on Computer Vision (ICCV)*, 2011.

[71] A. Gupta, S. Narayan, K. Joseph, S. Khan, F. S. Khan, and M. Shah, "Ow-detr: Open-world detection transformer," *arXiv preprint arXiv:2112.01513*, 2021.

[72] K. Joseph, S. Khan, F. S. Khan, and V. N. Balasubramanian, "Towards open world object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5830–5840, 2021.

[73] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[74] B. Hariharan, P. Arbelaez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *International Conference on Computer Vision (ICCV)*, 2011.

[75] W. Zhou, S. Chang, N. Sosa, H. Hamann, and D. Cox, "Lifelong object detection," *ArXiv*, vol. abs/2009.01129, 2020.