Universidad de Buenos Aires

Facultad de Ingeniería

Departamento de Electrónica

2021

Tesis de Ingeniería Electrónica

Detección de objetos en imágenes usando aprendizaje Semi-supervisado

---

por

Gonzalo Iñaki Quintana

Director: Dr. Ing. Leonardo Rey Vega

Agradecimientos

A mis padres, por haberme inculcado la importancia del trabajo duro y el amor por el conocimiento.

A mi familia, por su contención y apoyo incondicional.

A mi amor, Pauline, por haber estado en los momentos más duros.

A mis amigos, por haber sido mi cable a tierra durante todos estos años.

A la Universidad de Buenos Aires, y a todos sus excelentes y vocacionales profesores, en especial a Leonardo.

Resumen

El Aprendizaje Profundo o *Deep Learning* (DL) es el actual estado del arte en detección y segmentación de objetos en imágenes. Sin embargo, el entrenamiento de detectores basados en *Deep Learning* con un Aprendizaje Totalmente Supervisado (*Fully Supervised Learning*) representa un gran desafío en términos de adquisición de datos, ya que requiere de grandes *sets* de datos anotados. Este problema puede alivianarse mediante el uso de técnicas de Aprendizaje Débilmente Supervisado (*Weakly Supervised Learning*), que permiten que un modelo de Aprendizaje Profundo aprenda con datos parcial o completamente no anotados. En particular, nos concentraremos en un enfoque de Aprendizaje Débilmente Supervisado: el Aprendizaje Semi-supervisado (*Semi-supervised Learning*). Este tipo de aprendizaje permite incorporar datos no anotados al entrenamiento y explotar mejor las imágenes anotadas. Esto hace posible entrenar modelos con menos datos anotados, e incluso aumentar su desempeño sin necesidad de aumentar la cantidad de datos anotados.

En este trabajo, presentamos un detector basado en Aprendizaje Profundo que detecta dos clases de objetos en un tipo particular de imágenes. Luego de haber realizado un estudio bibliográfico sobre detección de objetos con *Deep Learning*, decidimos basar nuestro detector en *Faster R-CNN*. El detector fue primero entrenado de una manera totalmente supervisada utilizando un *set* de datos de 1000 imágenes anotadas. Luego, se utilizaron técnicas de Aprendizaje Semi-supervisado, en particular el Aprendizaje por Consistencia (*Consistency Learning*), para mejorar su desempeño. Se incorporó un *dataset* de 8500 imágenes no anotadas al *subset* de entrenamiento, lo que mejoró la capacidad de generalización y la robustez del detector, aumentando su Precisión Media (*Average Precision*) en 3,7 puntos para las dos clases detectadas. Además, usando un modelo Maestro Estudiante Promedio (Mean Teacher-student), logramos llevar el desempeño del detector a 23 y 12,2 puntos de Precisión Media para las clases 1 y 2, respectivamente. Esto implica un remarcable aumento de 6,7 y 8,4 puntos de Precisión Media, con respecto al detector entrenado de una manera totalmente supervisada.

Palabras clave: detección de objetos, Visión por Ordenador, Aprendizaje Profundo, Aprendizaje Débilmente Supervisado, Aprendizaje Semi-supervisado, Aprendizaje por Consistencia, Redes Neuronales Convolucionales.

Abstract

Deep Learning (DL) is the current state-of-the-art in object detection in images. However, training DL-based detectors with classical Fully Supervised learning represents a huge challenge in terms of data acquisition, as it requires large, annotated datasets. This issue can be alleviated by using Weakly Supervised Learning (WSL) techniques, which enable a Deep Learning model to learn with partially or completely unlabeled data. In particular, we are concerned by one WSL approach: Semi-supervised Learning (SSL). SSL permits to incorporate non-annotated data to training and to better exploit annotated images. This ultimately makes possible to train models with less annotated data, or to increase their performance without increasing the size of the annotated dataset.

In this work, we present a DL-based detector that detects two types of objects in a particular kind of images. After conducting a literature review in object detection with Deep Learning, we decided to base our detector on Faster R-CNN, a high-performing two stages detector. The detector was first trained in a Fully Supervised way using a dataset of 1k annotated images. After that, Semi-supervised learning techniques, in particular Consistency Learning, were used to improve its performance. A dataset of 8.5k non-annotated images was incorporated to the train set, which enhanced the generalization power and the robustness of the detector and increased its Average Precision (AP) by 3.7 AP points for the two detected classes. Moreover, by using a Mean Teacher-student model, we managed to rise performances to 23 and 12.2 AP points for Class 1 and Class 2 objects, respectively. This implies a remarkable increase of 6.7 and 8.4 AP points, with respect to the detector trained using FSL.

Keywords: object detection, Computer Vision, Deep Learning, Weakly Supervised Learning, Semi-supervised Learning, Consistency Learning, Convolutional Neural Networks.

Contents

1 Introduction

Object Detection in images is today one of the main problems of Computer Vision (CV), as well as an active research field with tons of industrial applications.

1.1 Computer Vision & Object Detection

Computer Vision is the field of Computer Science that focuses on enabling computers to identify and process objects in images and videos in the same way that humans do. One can identify mainly four tasks in Computer Vision: classification, semantic segmentation, object detection and instance segmentation. These tasks are illustrated in Figure 1 and detailed below.

- **Classification** is the first and simplest Computer Vision task. It consists of assigning a predefined class or category to the whole image, based on the objects that it contains. For example, in Figure 1a, we would classify the image into the "balloon" category. For developing image classifiers and assessing their performances, one needs to have a database of images labeled with different classes. That is, we only need image level-annotations.

- **Semantic segmentation** looks to identify similar objects in an image and decide of a category at a pixel level. That is, rather than predicting a class for the whole image as in the classification task, we want to predict a category label for each pixel of the input image (see Figure 1b). Semantic segmentation does not differentiate instances of the same class, it only cares about pixels. For instance, in Figure 1b all the pixels of the balloons are classified in the class "balloon", but no distinction is made between individual balloon instances. For developing such systems, one needs a labeled database of images with pixel-level annotations. Annotating images at a pixel level is much more costly and time-consuming than labeling at an image-level.

- **Object detection** is the ability to detect or identify all objects in an image along with their spatial position in that image, in the form of rectangular boxes (known as bounding boxes) which bound the object within it. That is, the object detection problem is concretely defined by two tasks that must be performed simultaneously: object localization (to determine where semantic objects are) and object classification (to determine the predefined category to which each object belongs to). In Figure 1c we can see that all the objects have been located and correctly classified as being balloons, which achieves the detection task. In object detection problems, we usually do not know the number of instances that each image contains. Developing an object detector needs object-level annotations, as it needs a database of images where all the objects have been bounded with the bounding boxes. This makes object detection more costly than classification, from a data point of view, but less costly than semantic segmentation.

- **Instance segmentation** attempts to predict the locations and the identities of all the objects in an image but, rather than just predicting bounding boxes as object detection, it seeks to precisely define those objects by predicting a whole segmentation mask for each one of them. Instance segmentation can be seen as a combination of semantic segmentation and object detection and is the most complex of all the Computer Vision tasks. The example of Figure 1d illustrates this well: the pixels corresponding to different balloons are painted with different colors, to indicate that they belong to different objects. The instance segmentation task, as well as semantic segmentation, needs pixel-level annotations, which makes them the costliest CV tasks from a data point of view.

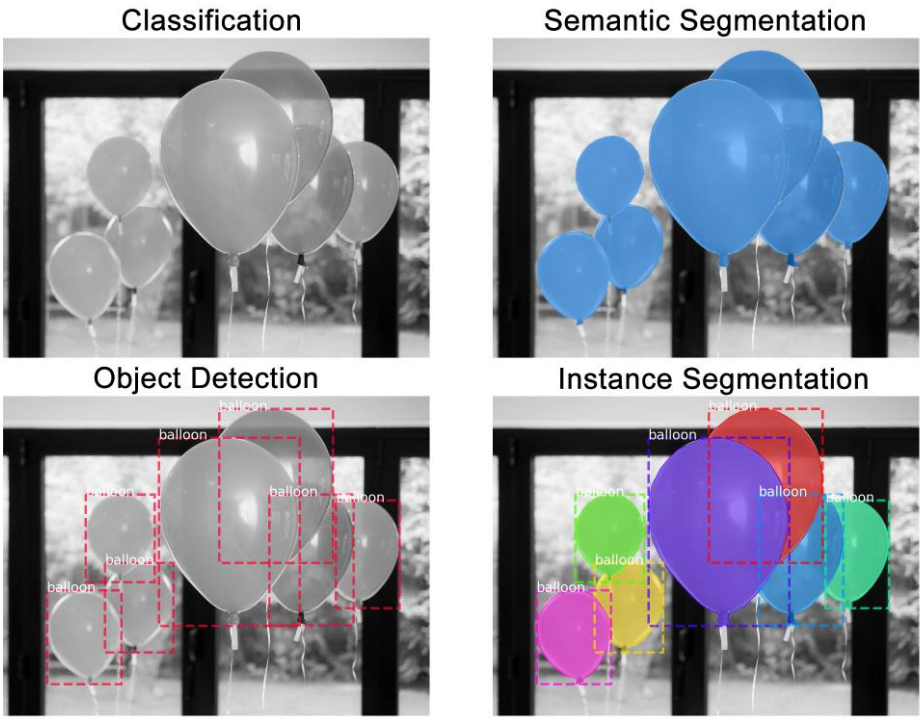

Figure 1: task of Computer Vision: classification (upper left corner), semantic segmentation (upper right corner), object detection (lower left corner) and instance segmentation (lower right corner). Source: https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46

In this work, we will focus on the object detection task. As one of the fundamental Computer Vision problems, object detection provides information for semantic understanding of images and videos, and is related to many applications in different domains, such as autonomous vehicles, surveillance, industry & manufacturing, medicine, human behavior analysis, etc.

Self-driving cars are one of the best examples of the importance of object detection. Before making the decision of whether accelerating, turning or applying the brakes, the autonomous vehicle needs to know which objects are around it (pedestrians, bikers, zebra-crossing, other cars, etc.) and exactly where they are. Object detection is also highly used in tracking systems for surveillance of cities and houses, traffic monitoring and video communication (Abdul Vahab, 2019). In manufacturing and industry, object detection is used for identifying products, materials, and components, and allow to automate lots of processes or tasks, like inventory management and product or component sorting. It is of extreme importance in automated assembly lines and distribution centers. In all these applications, we are interested in identifying all the objects in an image, video, or scene, but we do not need a segmentation mask that precisely defines the found objects. As

instance segmentation systems are more costly from a data point of view, more resource-consuming and harder to develop, we usually prefer solutions based on object detection algorithms.

Traditional object detection methods are built on handcrafted features and shallow trainable architectures. They are said to be *human-driven*, as the Computer Vision engineer actively participates in the feature extraction process. These approaches date back to the past 10 or 20 years and are characterized by extracting human-engineered features (edges, corners, colors) using classical algorithms like SIFT, SURF and BRIEF. The pipeline of traditional object detectors can be divided into three stages: informative region selection, feature extraction and classification (see Figure 2).

- **Informative region selection**: the entire image is scanned with a multi-scale sliding window to find all the objects, which can have different sizes and aspect ratios. Even though this strategy can find all the objects in an image, it is computationally expensive and produces too many redundant windows.

- **Feature extraction:** to recognize different objects, we extract visual features that provide a semantic and robust representation. This feature extraction process is human-engineered and uses classical feature detection algorithms like SIFT (Lowe, 2004), SURF (Herbert Bay, 2006) or BRIEF (Michael Calonder, 2010). However, due to the diversity of appearances, illumination conditions and backgrounds, it is difficult to manually design a robust feature descriptor to reasonably describe all kinds of objects (Zhong-Qiu Zhao P. Z.-t., 2019). When the number of classes and objects increase, the feature extraction process becomes more cumbersome, and performance decreases (Niall O'Mahony) (Zhong-Qiu Zhao P. Z.-t., 2019). Moreover, these types of detectors contain lots of hyperparameters which must be fine-tuned by the Computer Vision engineer in a tedious trail-and-error process (Niall O'Mahony).

- **Classification: t**he extracted features of each selected region are then passed to a shallow classifier that distinguishes the object contained in the region from all other categories. Examples of these classifiers are Support Vector Machines (Bernhard E

Boser, 1996), AdaBoost (Yoav Freund, 1996) and Deformable Part-based Models (Pedro Felzenszwalb, 2013).

These methods enabled to achieve state-of-the-art performances in the PASCAL VOC competition (Zhong-Qiu Zhao P. Z.-t., 2019). Some small gains in performance were obtained during 2010-2012 by building ensemble systems and employing small variants and modifications of these traditional CV approaches (Zhong-Qiu Zhao P. Z.-t., 2019). However, performance easily stagnates by mainly two reasons: 1) the generation of candidate bounding boxes with a sliding window strategy is redundant, inefficient, and inaccurate and 2) the semantic gap cannot be bridged by the combination of manually engineered low-level descriptors and discriminatively trained shallow models (Zhong-Qiu Zhao P. Z.-t., 2019).

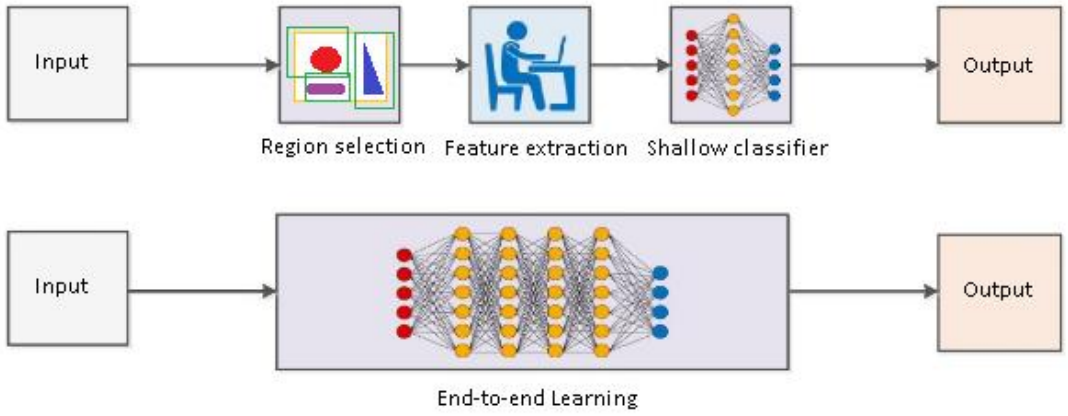

Figure 2: traditional CV detectors pipeline (above) vs. DL-based detectors pipeline (below). (Niall O'Mahony)

The emergence of Deep Learning (DL) introduced more powerful tools which are able to learn semantic, high-level, deeper features, and enabled to overcome the existing problems in traditional detector architectures, substantially increasing performances. Deep Learning introduces the concept of *end-to-end learning*, where the machine is just given a dataset of annotated images and is expected to produce the detection results, without an explicit human intervention. In contrast to traditional *human-driven* detection techniques, DL-based detectors are said to be *data-driven*. Throughout the training process, the DL model manages to discover the underlying patterns and works out the most descriptive and salient features for each class. This can be seen in Figure 2, which contrasts the *end-to-end* pipeline of DL-based detectors with the three-stages pipeline of traditional detection algorithms. Besides, DL detectors have deeper architectures with the capacity to learn more complex

features than shallow ones. Using this approach requires often less expert analysis and fine-tuning, while exploiting the tremendous amount of video and image data available today. DL also provides superior flexibility, as DL-based detectors can be re-trained using a custom dataset for any use case, contrary to traditional CV algorithms, which tend to be more domain-specific.

1.2 Data challenge & motivation

Deep Learning (DL) based detection and segmentation models emerged as today's state-of-the-art. They achieve higher performance and are more robust and faster than their classical counterparts. DL models are typically trained using Fully Supervised Learning (FSL), which consists of learning a function or mapping with labeled data. However, developing such detectors represents a huge challenge in terms of data acquisition, as it requires huge, annotated datasets. Moreover, those datasets should also be highly variated and not contain biases, as any bias in the data will translate to the model. For most practical applications, collecting those high volume and variated annotated datasets is expensive and incredibly time-consuming. Some concerns also rise in terms of data privacy. All these reasons hinder the development of DL-based detectors.

Weakly Supervised Learning (WSL) is a learning paradigm where noisy, unlabeled, or imprecisely labeled data is used for training Machine Learning (ML) algorithms. This approach appears as an alternative to FSL, where only labeled data is used for training, and alleviates the burden of obtaining big hand-labeled data sets. Instead, inexpensive weak labels are employed with the understanding that they are imperfect but can nonetheless be used to create a strong predictive model or to improve an existing trained model.

Semi-supervised Learning (SSL) is a WSL approach that combines the classical paradigms of Fully Supervised Learning (FSL) and Unsupervised Learning (UL). By incorporating non-annotated data to training, it increases the generalization capacity and robustness of ML and DL models. Moreover, it also enables the model to extract more semantic information from annotated data than classical FSL. This ultimately makes possible to train models with less annotated data, or to increase performance without increasing the size of the annotated dataset.

The objective of this work is to exploit Weakly Supervised Learning techniques, in particular Semi-supervised Learning, to reduce the annotated data needed for training object detectors. A Deep Learning-based object detector will be developed and first trained using FSL. Then, the same detector will be trained using SSL, which will enable to incorporate a non-annotated dataset to the training. This will allow to compare and assess the benefits of a Semi-supervised training.

1.3 Organization of this work

The rest of this work is organized as follows. The next section (section number 2) contains a bibliographic and literature review, which is divided into two main parts: object detection with Deep Learning and Weakly Supervised Learning. In the first part, we present the two main paradigms found in the literature, which are single stage and two stages detectors. We also explain in detail the best performing detectors of each group (Faster R-CNN and RetinaNet). Additionally, we detail the metrics that are commonly used in the Computer Vision community for assessing detectors performances. In the second part, we discuss Weakly Supervised Learning and analyze the two WSL approaches that are commonly of interest in the incomplete supervision case: Semi-supervised Learning and Active Learning (AL).

The third section contains all the methodological aspects. We present the architecture of our detector (Faster R-CNN), and justify the election of the hyperparameters. We also explain the pipeline used for Semi-supervised Learning, and detail all the modifications carried out on the base detector for enabling training with non-annotated data. In the fourth section we present and analyze the datasets used in this work, and show the results obtained with the two types of training: Fully Supervised and Semi-supervised. We detail all the experiments that were carried out and compare the results. The last section contains the conclusions that are extracted from this work, as well as its limitations and possible future directions.

2 Literature review

We conduct a bibliographic research divided into two great parts: object detection with Deep Learning and Weakly Supervised Learning. The current state-of-the-art approaches for detecting objects in images are all DL-based and can be divided into two groups: single stage detectors and two stages detectors. Weakly Supervised Learning stands for the techniques that allow us to train a DL model with partially or totally unlabeled data. It consists of mainly two approaches: Semi-supervised Learning and Active Learning.

2.1 Object detection with Deep Learning

Deep Learning-based detectors achieve the best performances and the lowest computational complexity in object detection. There are two different paradigms in DL-based object detection: single stage detectors and two stages detectors.

2.1.1 Single stage detectors vs two stages detectors

Single stage detectors typically consist of a Convolutional Neural Network (CNN) Backbone to which we attach two neural networks, a classifier and a regressor. The regressor predicts bounding boxes (BBs) coordinates that contain the detected objects in the image, and the classifier a class probability for each object contained in the predicted bounding boxes. Both sub-networks (classifier and regressor) take as input the feature maps generated by the CNN Backbone, and forwards them through convolutions or fully connected layers. Classical single stage detectors are YOLO (Joseph Redmon, 2016) and SSD (Wei Liu, 2016). Those detectors often have less trainable parameters, are quicker and are more adapted to real time detection, but achieve lower performance than two stages

detectors. RetinaNet (Tsung-Yi Lin P. G., 2018) significantly increases the performance by introducing a novel Focal Loss, achieving those of two stages detectors in some datasets.

On the other hand, two stages detectors usually outperform single stage ones, but are slower and harder to train. They consist of a first stage that proposes Regions of Interest (RoI), i.e., regions in an image where there is likely to be an object, and a second stage that conducts the final detection. DL-based two stages detectors were first introduced in the Regions with CNN features (R-CNN) detector (Ross Girshick, 2013). The first stage finds the Regions of Interest using Selective Search, a non-learning algorithm that proposes around 2000 regions in the image, and then forwards them through a CNN ended in a Support Vector Machine (SVM) that conducts the final detection. However, this method is computationally expensive as the 2000 RoI are forwarded independently through the CNN. Fast R-CNN (Girshick, 2015) significantly lowers the computational complexity by forwarding the entire image through the CNN at once, and then projecting the proposals to the convolutional feature maps. In Fast R-CNN, the bottleneck in computation time is the Selective Search algorithm of the first stage. Faster R-CNN (Ren S, 2016) solves this issue by using a CNN-based network, the Region Proposal Network (RPN), for calculating the RoI. The RPN and R-CNN (first and second stages) share the CNN's weights, decreasing the computational complexity. These shared CNN weights are commonly known as CNN Backbone.

Faster R-CNN (two stages detector) and RetinaNet (single stage detector) are currently the state-of-the-art in DL-based object detection.

2.1.2 Faster R-CNN

Faster R-CNN is a two stages detector that combines the previous Fast R-CNN with a Region Proposal Network (RPN) to enhance its performance and lower the computational complexity. It can be thought of like a CNN-based classifier and localizer with an attention mechanism (the RPN) that tells it where to look for objects, as shown in Figure 3.

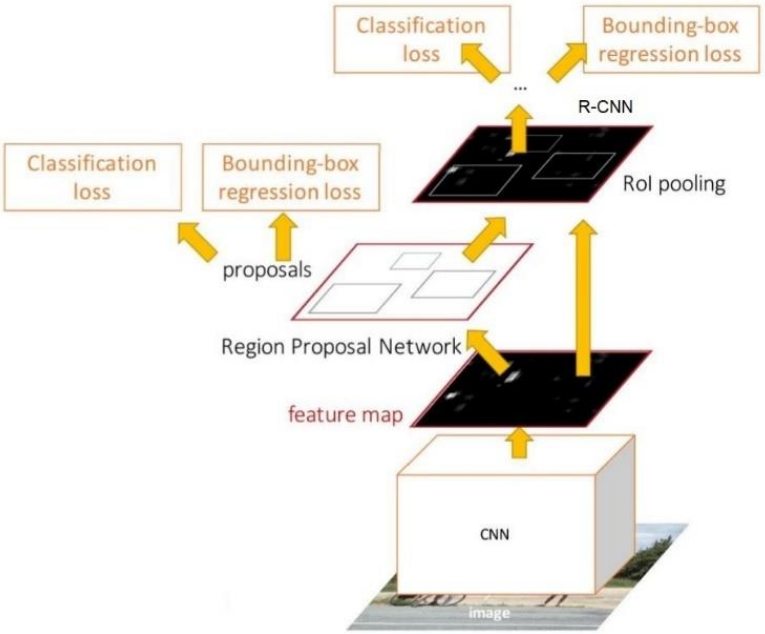

Figure 3: Faster R-CNN basic pipeline. (Ren S, 2016)

The input image is first forwarded by a CNN Backbone, that is shared between the RPN and the R-CNN. The output feature maps are taken by the RPN, which predicts Regions of Interest of different sizes. These RoI are then forwarded through the R-CNN detector, which classifies each RoI (between all the possible object classes and the background) and predicts four bounding boxes coordinates with respect to the RoI's borders (and not to the original image), in order to correct the first stage's localization. As shown in Figure 3, Faster R-CNN implements a multi-task learning with four losses: a classification loss and a regression loss for the RPN, and another two for the R-CNN. These four losses are combined in a single final loss, that enables us to train the two networks simultaneously (detector trained in an end-to-end fashion).

2.1.2.1 Region Proposal Network (RPN)

The RPN is based on the concept of anchors boxes. Anchor boxes are bounding boxes of fixed shape that are placed in every pixel of the feature maps generated by the CNN Backbone (evenly spaced in the input image) and are used as a base to construct the RoI predicted by the RPN. The anchor boxes are defined by their size and their height-width relationship. For instance, in the original Faster R-CNN paper, 3 sizes (128^2, 256^2 and 512^2) and 3 height-width relationships (0.5, 1 and 2) are used. This gives 9 different anchor boxes

(Figure 4). For each one of the anchor boxes, the RPN decides if they possibly contain an object or not; it also adjusts the anchor boxes coordinates by a regressor network. This gives the final RoI predictions.

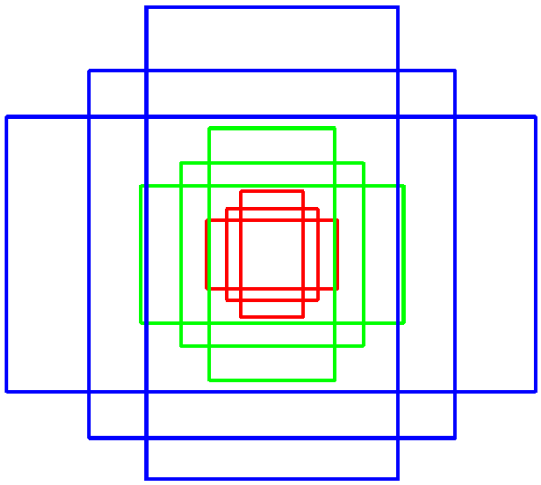

Figure 4: anchor boxes of three different sizes (in colors) and three aspect ratios. Source: https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

The RPN first applies a 3x3 strided convolution on the feature maps produced by the CNN Backbone (of size H×W×C), generating a H×W×256 feature maps (a feature vector of 256 elements for each pixel in the input feature map). Then, two 1×1 convolutions are applied to the feature maps (the classifier and regressor layers). The classifier layer outputs object/non object probabilities for each of the anchor boxes centered at each of the pixels of the feature map; the regressor layer outputs 4 bounding boxes coordinates (relative to the anchor box) for each of the anchor boxes, for each pixel of the feature map. This is illustrated in Figure 5.

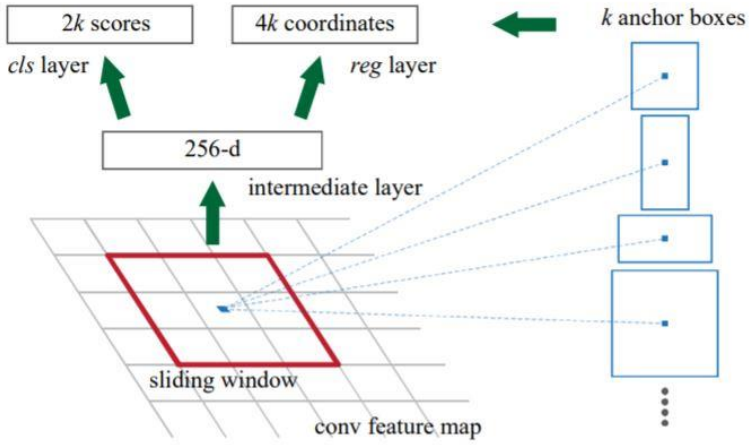

Figure 5: Region Proposal Network. (Ren S, 2016)

The RPN is trained with the following compound loss function, that is the weighted sum of the classification and regression losses:

$$L_{RPN} = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Equation 1: RPN loss function.

where p_i is the predicted probability of the anchor i containing an object, p_i^* is the ground-truth probability of anchor box i containing an object (positive/negative anchors), t_i are the coordinates of the predicted bounding box i with respect to the anchor box and t_i^* those of the ground-truth box associated to that anchor (with respect to that anchor box). L_{cls} and L_{reg} are the classification and regression losses, respectively. The classification loss is the classical Cross Entropy Loss (or negative log-likelihood), and the regression loss is the Smooth L_1 loss, given by Equation 2. This loss behaves like the L_2 loss for small values and like the L_1 loss for big ones and allows us to prevent exploding gradients, that may happen when using L_2 loss for unbounded regression targets. The classification and regression terms are normalized by N_{cls} and N_{reg}, which are the mini-batch size (256 in the original paper) and the number of anchor locations (around 2400), respectively. The weight λ is set to 10 in order to have those two terms roughly equally weighted (Ren S, 2016).

$$smooth_{L1}(x) = \begin{cases} 0.5\,x^2\ if\ |x| < 1 \\ |x| - 0.5\ otherwise \end{cases}$$

Equation 2: smooth L1 loss.

In order to calculate this loss, we must define the concept of positive and negative anchor boxes. A positive anchor box is an anchor box that is considered to contain or to be associated to an object in an image. A negative anchor box is considered not to contain an object. Positive anchor boxes will have $p_i^* = 1$ in Equation 1 and negative ones will have $p_i^* = 0$.

To determine whether an anchor box is negative or positive, we relate it to the bounding boxes ground-truths (which are available as we are in training mode). An anchor box i is positive if it has an Intersection over Union (IoU) of more than 0.7 with some ground-truth box or if it has the highest IoU of all the other anchor boxes with a given ground-truth box (being all of them smaller than 0.7). An anchor box is set to negative if it has an IoU of less of 0.3 with all the ground-truth boxes. Anchors that are neither positive nor negative do not contribute to the loss.

During training, in the original paper they follow an "image centric" sampling strategy, which consists in forming each mini batch with example anchors from the same image. As negative examples are dominant (objects in an image are "rare events"), each mini batch (of size 256) is formed with a ratio of up to 1:1 of negative and positive examples. If there are fewer than 128 positives examples, the mini batch is padded with negative ones (Ren S, 2016).

2.1.2.2 RoI Pooling vs. RoI Aling vs. RoI Warping

As explained in the las subsection, the RPN outputs offsets for each of the anchors, that correct them and give the final proposed region coordinates. These coordinates are referred to the original image size and the RoI that they define are of arbitrary size (as objects in an image do not have a fixed size). The second stage of the detector, the R-CNN, takes as input feature maps of fixed size, that should contain the features of the RoI proposed by the RPN. Fast R-CNN implements a mechanism that connects these two stages: the RoI Pooling block. This block is in charge of projecting the RoI coordinates to the features space, and of fixing the size of the input to the R-CNN.

RoI Pooling

RoI Pooling is the mechanism that appeared in the Faster-RCNN original paper. It is in charge of projecting the RoI coordinates to the feature maps and cropping the area of the feature maps delimited by those coordinates. As the Backbone is a CNN, it implements a series of convolutions that decrease the size of the output feature maps by a ratio of k (where $k=16$ in the original paper), with respect to the input image size. RoI Pooling proposes to divide each coordinate of the proposed region by k, and then to take the integer part, that is,

$$x' = [x/k]$$

Equation 3: coordinate translation to the features space.

Where x is a coordinate of the proposed region with respect to the original image size, and x' is the coordinate associated to that proposed region, in the features space. With these new coordinates, we can crop the features that correspond to the proposed region from the feature maps generated by the Backbone.

However, the features extracted with these new coordinates will have an arbitrary size. To fix its output size, RoI Pooling divides the h × w cropped feature maps into a grid of H × W bins (in the original paper, H=W=7), of approximate size h/H and w/W (however, if h or w are not a multiple of H or W, some bins will be bigger than others, as in Figure 6). The value of each bin is set by max-pooling from the pixels that compose it (we could also take the average). Pooling is applied independently to each channel in the feature maps. This whole process is illustrated in Figure 6.

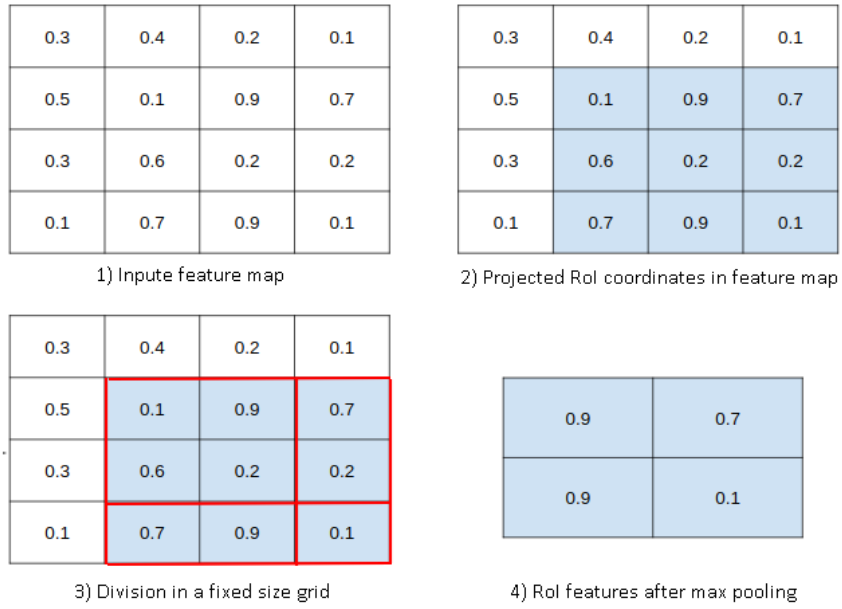

Figure 6: visualization of RoI Pooling with a 2x2 output feature map size.Source: https://firiuza.medium.com/roi-pooling-vs-roi-align-65293ab741db

We remark that this method carries an information loss during the two quantization processes. The first quantization is performed when rounding the result of the division by *k* in Equation 3, and the second one when dividing the cropped feature maps in bins of integer size. This is does not have much negative effects for classification tasks but can be problematic when doing segmentation or object detection.

RoI Align

RoI Align was first presented in the Mask R-CNN paper (Kaiming He G. G., 2018) and has the same goal as RoI Pooling. It also divides the input coordinates by *k* but, unlike RoI Pooling, it does not take the integer part in Equation 3. This means that the new coordinates are float values. The cropped feature maps are also divided into a grid of H × W bins, except that know every bin has exactly the same size (h/H × w/W). The full process is illustrated in Figure 7. For calculating the value in each bin, RoI Align defines four regularly sampled locations in each bin (see the step three in Figure 7) and uses bilinear interpolation to compute the values at each location. Finally, as in RoI Pooling, the four locations in each bin are pooled using max or average pooling.

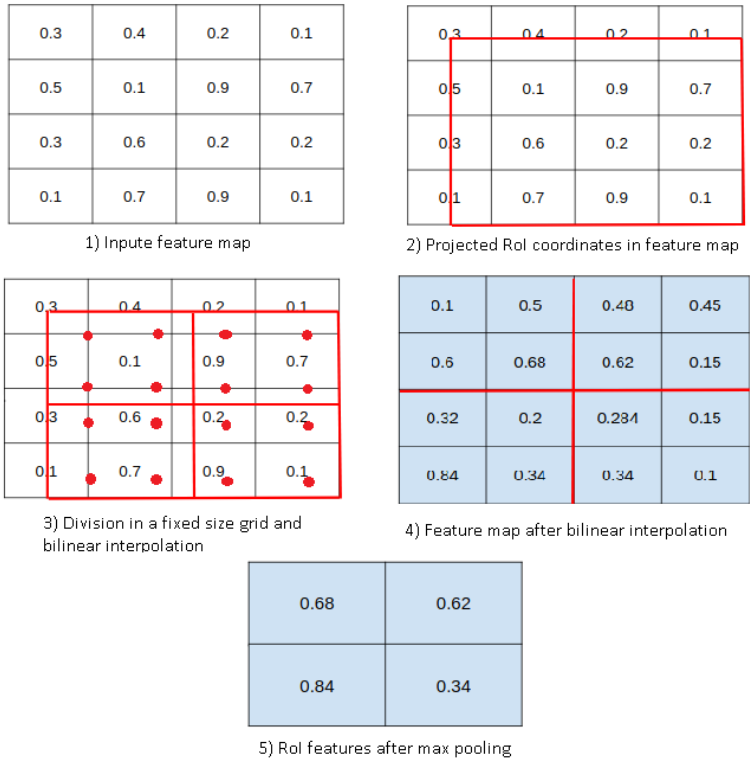

Figure 7: visualization of RoI Align with a 2x2 output feature map size. Source: https://firiuza.medium.com/roi-pooling-vs-roi-align-65293ab741db

The advantage of RoI Align is that the two quantizations present in RoI Pooling are avoided. These quantizations introduce misalignments between the RoI and the extracted features. While this does not impact classification, which is robust to small translations, it has a large negative effect on predicting pixel-accurate masks (Kaiming He G. G., 2018).

RoI Warping

RoI Warping (Jifeng Dai, 2015) is the third of the mechanisms that can be used for solving this problem and is situated between RoI Pooling and RoI Align. RoI Warping performs quantizations when projecting coordinates on the image to coordinates on the feature maps, but it does not perform quantization when fixing the size of the RoI feature maps. The three presented methods are summarized in Table 1. In the literature, the best results for detection and segmentation are obtained with RoI Align.

	RoI Pooling	RoI Warping	RoI Align
Projecting coordinates on image to coordinates on the feature map.	Quantization	Quantization	No quantization
Fixing the size of the RoI features	Quantization	No quantization	No quantization

Table 1: comparison of the quantizations performed by RoI Pooling, RoI Warping and RoI Align.

2.1.2.3 R-CNN (Regions with CNN features)

The R-CNN is the second and last stage of the Faster R-CNN detector, and performs the final object detection. It takes as input the feature maps of each of the proposed RoI (output of the RoI Pooling, RoI Align or RoI Warping block) and forwards them through a shared Fully Connected Network (FCN) and then through two separated Fully Connected Layers. These two networks are the final classifier and regressor. The classifier ends in a Softmax layer, that generates a class-probability vector of length *k+1* (*k* being the number of foreground classes). The regressor ends in a linear layer, and the output is a vector of length *4k* that contains the bounding box coordinates (with respect to the RoI boundary) for each class.

We recall that, as the RoI have an arbitrary shape and the network requires a fixed-size input (it has several Fully Connected Layers), Fast R-CNN implements a Pooling mechanism (described in the above subsection) that transforms feature maps of arbitrary size and shape to a fixed size tensor, that can then be forwarded through the network. In order to reduce redundancy in the proposed RoI, Non-Maximum Suppression (NMS) is used. This consists of eliminating the proposed regions that have an IoU higher than 0.7 with another RoI.

The R-CNN is trained with the following compound loss function, that is the weighted sum of the classification and regression losses.

$$L_{R-CNN} = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v)$$

Equation 4: R-CNN loss.

Each training RoI is labeled with a ground-truth class u and a ground-truth bounding box v. In Equation 4 t^u is the predicted bounding box for class u, the ground-truth class of that

region and p is the predicted probability vector for that region. L_{cls} is a classical log loss and L_{loc} the smooth L_1 loss. $[x]$ is the indicator function, that takes the value 1 when the condition x is met, and 0 when it is not. It indicates that the localization term is only calculated for positive RoIs.

2.1.3 RetinaNet

RetinaNet is a single stage detector proposed by the Facebook AI Research team in 2018 (Tsung-Yi Lin P. G., 2018). Single stage detectors are outperformed by two stages ones mainly because of extreme foreground-background disbalance (objects of interest are considered as "rare" events in the image). Easy examples, when abundant, can sum up and dominate the gradient, making networks converge to the trivial solution of not detecting anything. The idea is to eliminate this accuracy gap and construct high performance single stage detectors that can run faster that its two stages counterparts.

2.1.3.1 Focal Loss

The key concept behind RetinaNet is the loss function. In the context of classification we can define the variable p_t, the probability of ground-truth class, as

$$p_t = \begin{cases} p & if\ y = 1 \\ 1 - p & otherwise \end{cases}$$

Equation 5: probability of ground-truth class.

With this, the classical Cross Entropy (CE) loss can be rewritten as

$$CE(p, y) = CE(p_t) = -\log(p_t)$$

Equation 6: Cross Entropy loss.

We can also define the class-balanced Cross Entropy loss

$$CE(p_t) = -\alpha_t \log(p_t)$$

Equation 7: class-balanced Cross Entropy loss.

Where α_t is usually the inverse of the class frequencies, or a hyperparameter tunable with cross validation. This parameter balances the importance of different classes, but it does not differentiate between easy and hard examples.

The Focal Loss is defined as

$$FL(p_t) = -\alpha_t(1 - p_t)^{\gamma}\log(p_t)$$

Equation 8: Focal Loss.

where γ reduces the contribution of easy examples and extends the range in which an example receives low loss. When p_t goes to zero and the example is misclassified, the loss does not change much with respect to the Cross-Entropy loss. However, when p_t goes to one, the multiplicative factor goes to zero and the loss for well classified examples is down weighted. Misclassified examples contribute more to the loss and, therefore, the network can focus on difficult examples. By increasing γ, the focal loss changes as shown in Figure 8. For γ= 0, the FL is equivalent to the CE loss. Experimentally, the paper's authors obtained that α should be decreased slightly as γ is increased (best value α = 0.25 and γ=2).

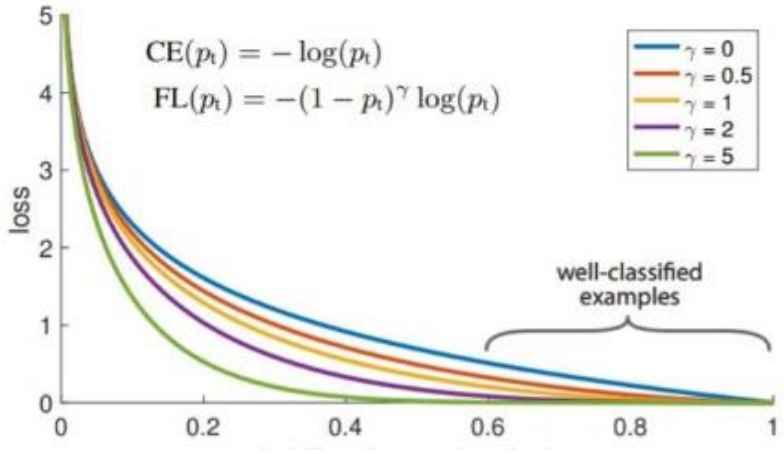

Figure 8: Focal loss and Cross Entropy loss. (Tsung-Yi Lin P. G., 2018)

2.1.3.2 Network Architecture

RetinaNet uses Feature Pyramid Networks (FPN) as Backbone for improving multi-scale detection (Tsung-Yi Lin P. D., 2017). FPN (Figure 9b) are composed of a bottom-up and bottom-down pathways that adds semantic information to low-level feature maps, which are usually rich in resolution but have a poor semantic value. This enables to use multiple

feature maps of different levels to conduct detection, which improves detection of objects of different sizes. In each scale, detection is performed by a classification and regression sub-networks (Figure 9c). As these two sub-networks are purely convolutional, they can be applied to different size feature maps. The classification network predicts a class probability vector of size *k* (number of classes) for each anchor box (*A* anchor boxes), for each pixel in the feature maps; the regression network predicts 4 boxes coordinates for each anchor box, for each pixel in the feature maps (Figure 9d). In each scale, there are three sizes of anchor boxes as well as three aspect ratios. The classification and regression sub-networks do not have shared weights.

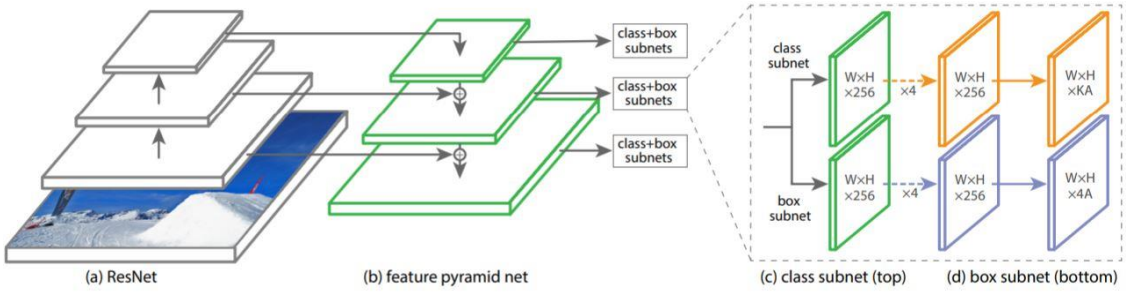

Figure 9: RetinaNet pipeline. (Tsung-Yi Lin P. G., 2018)

RetinaNet uses Focal Loss as the classification loss and Smooth L_1 loss for regression. For the loss calculation, each anchor is assigned the ground-truth box with which it has the highest IoU (the IoU must be of at least 0.5). If the highest IoU is smaller than 0.4, the anchor is assigned to the background class. Anchors with IoU between 0.4 and 0.5 do not contribute to the loss.

2.2 Evaluation metrics: Average Precision and COCO evaluation

Average Precision (AP) is the most common metric for object detection, used to compare results between different papers. Its definition is based on two important concepts: precision and recall.

The precision of a detection system is the percentage of detected instances that are truly real instances, that is, the number of true positives (or true detections) divided by the sum

of the numbers of true positives and false positives (all the detections). The recall is defined as the percentage of instances that were detected, that is, the number of true positives over the sum of the numbers of true positives and false negatives. These definitions are summarized in Equation 9.

$$p = {TP}/{TP + FP}$$

$$r = {TP}/{TP + FN}$$

Equation 9: precision and recall definitions.

The output of a typical detector is a class score or probability, and four bounding box coordinates. The TP, FP and FN and, consequently, the precision and the recall, depend on two parameters: a class score threshold and an IoU threshold, both lying between 0 and 1. The IoU threshold determines the minimum intersection over union score that two bounding boxes need to have to be considered as localizing the same object. The class score is the confidence level that the network has when it detects some object in the image and assigns it to a class (and not to background). All detection results with class score smaller than this threshold are not considered as positive detections.

By fixing the IoU threshold to a certain value and varying the class score threshold from 0 to 1, we construct the precision-recall curve (Figure 10). When the class score threshold is near to 0, we can detect all the real instances (recall →1), but we accept almost every detection result as positive (precision →0). On the other hand, when the class score reaches 1, we are extremely exigent and we do not make false detections (precision →1), but we fail to detect all the true instances (recall→0).

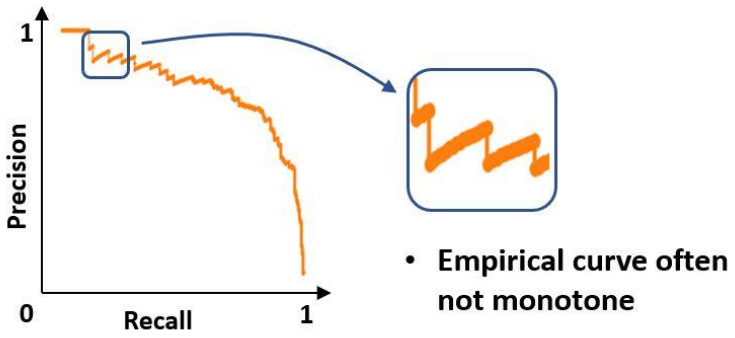

Figure 10: precision-recall curve.

For a fixed IoU threshold, the area under the precision-recall curve or Average Precision, is a measure that allows us to evaluate and compare different detectors.

$$AP^{IoU} = \int_0^1 p(r, IoU)dr$$

Equation 10: Average Precision (AP).

The COCO dataset (Lin TY, 2015) defines a series of standardized AP-based metrics, that are commonly used in the Computer Vision community.

$$AP = mAP^{IoU=[0.5:0.95]}$$

$$AP50 = AP^{IoU=0.5}$$

$$AP75 = AP^{IoU=0.75}$$

Equation 11: AP metrics defined in COCO.

where $mAP^{IoU=[0.5:0.95]}$ is the AP averaged from IoU=0.5 to 0.95 at 10 scales, and is commonly known as AP. COCO also defines a per class AP and an AP evaluated at different object scales: APs, APm, APl, which stands for small, medium, and large, respectively. However, in this work we will not deal with object-scale AP.

2.3 Weakly Supervised Learning

Weakly Supervised Learning is an umbrella term that considers learning where data is not fully labelled (Zhou, 2018). This includes three possible scenarios: incomplete supervision (some subset of the data is labeled, and the rest is unlabeled), inexact supervision (only coarse-grained labels are given) and inaccurate supervision (some given labels are wrong or are not ground truth) (Zhou, 2018). In this project we will mainly focus in the first scenario: we have access to annotated and non-annotated databases of images, and some annotations may be missing in the annotated images (that is, an image can contain annotated and non-annotated objects). However, all the annotations available are correct and precise. For solving this issue, two main approaches are typically considered: Semi-supervised Learning and Active Learning.

2.3.1 Semi-supervised Learning

Semi-supervised Learning is defined as the set of techniques and approaches that enable us to incorporate non labeled data into training, and to exploit more information from labeled data. In the literature, we identify two approaches for achieving this in the case of object detection: Consistency Learning (CL) and Self-supervised Learning (SL).

2.3.1.1 Consistency Learning

Many of the current state-of-the-art semi-supervised techniques consist in evaluating consistency between the detection results (classification + bounding box regression) of different versions of the feature maps of an input image. This is illustrated in Figure 11. An input image is forwarded through the first layers of the detector (typically convolutional), generating a feature map. This feature map is then perturbated or transformed N times (this can be achieved by geometric transforms, like flips, or by adding noise to it) and these N new feature maps are forwarded through the upper layers of the network, producing N different detection results. The key concept here is that, as all the feature maps are only perturbated versions of an original one and belong to the same image, the detection result should be (approximately) the same for all of them. That is, Consistency Learning is based on the hypothesis that the features space is smooth. To make the network learn this, we calculate and minimize a consistency loss, that increases when the N detection results differ much between each other and is minimal when they are all the same. We note that for calculating this loss function we do not need the ground-truth labels, which makes it an unsupervised loss. If the input comes from the labeled dataset, we also calculate a classical supervised loss (which can be cross entropy for classification and Smooth L_1 loss for regression) and combine the supervised and unsupervised losses by a weighted sum.

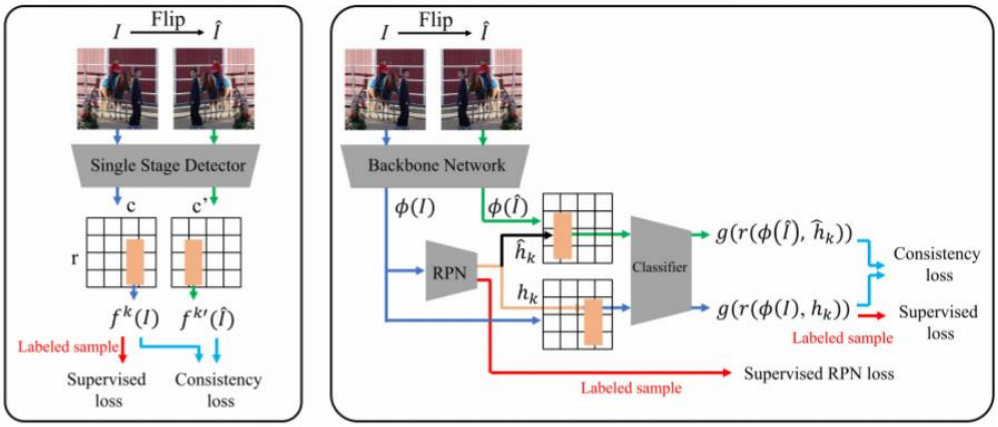

Figure 11: consistency loss in single stage and two stages detectors. (Jisoo Jeong, 2019)

In all consistency-based techniques this schema is maintained and mainly two things vary between different implementations: how are the input images transformed and the type of consistency loss used. For instance, Tang et al. use Spatial and Block Dropout for randomly transforming the feature maps and the Kullback-Leibler divergence (classical metric for stating the difference between two probability distributions) and the Smooth L_1 loss as the classification and regression consistency losses, respectively (Peng Tang, 2020). Jeon et al. transform the feature maps by only flipping them and use the Jensen-Shannon and the L_2 loss as consistency losses (Jisoo Jeong, 2019).

2.3.1.2 Consistency, Self-ensembling and Temporal Ensembling

Another way of understanding consistency-based learning is by the concept of Self-ensembling (Samuli Laine, 2017). The Ensembling technique consists of training several detectors in different datasets and then combining all their predictions to get a final prediction. This makes each detector focus on different aspects of the data and learn different characteristics. The final ensemble prediction is usually better that any individual's prediction. Laine et al. explain that this can also be achieved implicitly by training a single neural network with dropout and under different data augmentations (or input transforms), which makes the output of the neural network a random variable (Samuli Laine, 2017). As each forward pass of the same input through the network produces a different output, this neural network is equivalent to an ensemble of N different networks.

They call this technique Self-ensembling and implement two variations of it: the π-model and the Temporal Ensembling model.

The π-model is equivalent to the classical consistency model presented above: they forward an input image two times, under different stochastic augmentations (dropout and additive gaussian noise) and they use the L_2 loss between the final feature vectors as consistency loss (they only do classification).

The Temporal Ensembling model consists of combining the predicted feature vectors during different epochs with a moving average of themselves and calculating consistency between the feature vectors of the current epoch and the averaged version. This significantly improves execution time (2x speedup during training over the π-model) and slightly improves the performance of the π-model (Samuli Laine, 2017), as the averaged feature vectors (which can be considered as "targets") are more stable and have a lower probability of containing incorrect results. However, Temporal Ensemling cannot be used for large datasets, as a copy of the prediction for every image must be stored in memory, and it is unclear how it could be used for on-line learning (Antti Tarvainen, 2018). The two Self-ensembling models outperform Fully Supervised models and are also more robust and tolerant to incorrect labels.

2.3.1.3 Mean Teacher-student model

Tarvainen et al. relate consistency-based learning to pseudo-labelling. When calculating consistency loss, one of the two predictions can be thought as target or pseudo-label, proposed by the model itself. However, as the model under training generates its own targets for consistency calculation, they may be incorrect, which will cause the model to suffer from confirmation bias (Antti Tarvainen, 2018).

In order to mitigate this effect, they propose to improve the quality of the targets by using a Mean Teacher-student (MTS) model. This MTS model consists of having two twin networks that share the same architecture but have different weights. The Student's weights are updated by minimizing the final loss (supervised loss + consistency loss) via backpropagation. The Teacher's weights are the result of an Exponential Moving Average (EMA) of the Student's weights, which can be easily calculated in an iterative way (Antti Tarvainen, 2018). The supervised loss is calculated between the Student's predictions and

the ground-truths, and the consistency loss between the Student's and the Teacher's predictions (after applying transforms to the feature maps). Actually, it is shown that averaging model weights over training steps tends to produce a more accurate model and more reliable predictions than using the final weights directly (Antti Tarvainen, 2018). The Mean Teacher-student model is conceptually similar to the Temporal Ensembling model, but it has two practical advantages over it. Firstly, in the MTS model the learned information is incorporated into the training process at a quicker pace: the Teacher model is updated once per iteration and not once per epoch, as in the Temporal Ensembling model. This results in a better test accuracy (Antti Tarvainen, 2018). Secondly, the MTS model scales to large datasets and on-line training (Antti Tarvainen, 2018).

2.3.1.4 Self-supervised Learning

Self-supervised Learning is the other common approach for improving performances by using unlabeled data. It consists of using the network for solving an auxiliary or pretext task, for which we can automatically obtain the targets or labels. Solving this pretext task, which is different from the original task, would allow the network to learn more contextual information from the data, and therefore to perform better it the original task. The most common pretext-tasks that we can find in the literature are predicting a patch position in the original image, solving jigsaw puzzles, denoising, color restoration, etc.

Self-supervised Learning is often used for pre-training. Chen et al. use the context restoration task for pre-training neural networks for classification, detection, and segmentation (Liang Chen, 2019). They randomly select a number N of isolated patches in an image and swap their content. They use an autoencoder to reconstruct the original image and calculate the L_2 loss between the original and reconstructed images. Taleb et al. solve jigsaw puzzles as pretext task for pre-training a segmentation model (Aiham Taleb, 2019). They cut an image into pieces and randomly reorder them to generate the jigsaw puzzle that they then solve with the network. In Figure 12, the network predicts a permutation matrix S which solves the puzzle when applied to the disordered image P. Mean Squared error (MSE) is calculated between the original image $P*$ and the solution of the puzzle S^TP.

However, Self-supervised Learning can also be used during the main training loop for increasing the performance. For instance, Tang et al. introduce Self-learning in the pipeline

of Faster R-CNN (Peng Tang, 2020). The pretext task they use is predicting the RoI position in the original image (the output coordinates of the RPN).

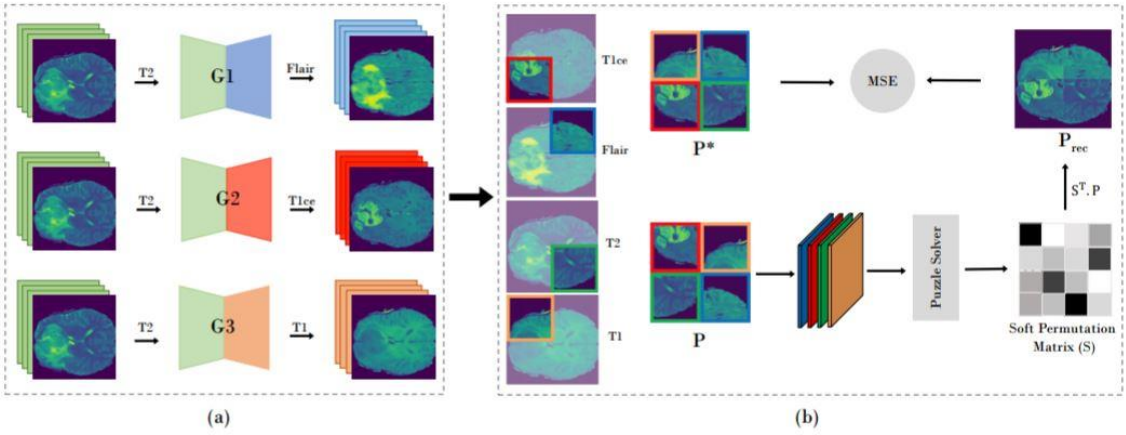

Figure 12: example Self-supervised Learning with solving jigsaw puzzle as pretext task (Aiham Taleb, 2019).

2.3.2 Active Learning

Active Learning (AL) is a learning algorithm in which the model takes a less passive role during training, as it interacts with the human specialist in the labeling procedure. Figure 13 shows the pipeline of an active training. Given a small set of labeled data and abundant unlabeled data, the model is first trained with the labeled instances in a fully supervised way over several iterations. It is then used on inference mode for selecting the most valuable unlabeled instances for human labelling. This selection is based on two criterions: informativeness (measures which unlabeled instances contain more unknown information) and representativeness (which unlabeled instances are more representative of all the data). After these selected instances are annotated, they are incorporated to the labeled dataset and used for re-training the model. By repeating this process during *N* active iterations, the AL-trained model achieves almost the same performances with less annotated data than that same model trained in a fully supervised way.

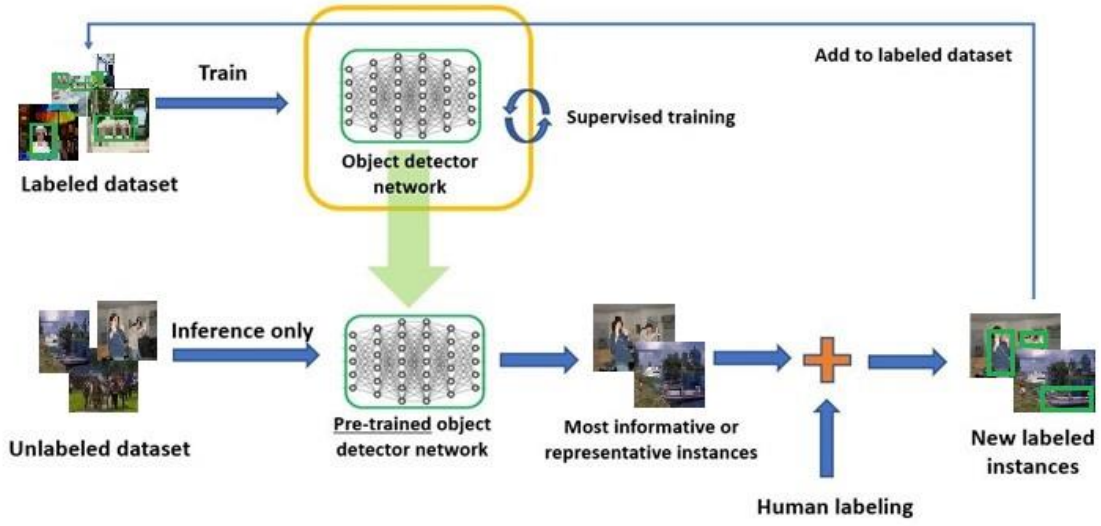

Figure 13: Active Learning pipeline.

In most recent Active Learning works, pseudo-labelling is also incorporated. It consists of taking the network's high confidence prediction as labels and using those pseudo-labeled instances (that come from the unlabeled dataset) for training the model. These samples are selected and incorporated to the labeled dataset at each active iteration. However, the pseudo-labeled samples are generally removed from the labeled train set and sent to the unlabeled set after each active iteration.

The pipeline of Figure 13 is generally maintained in almost every Active Learning paper. What generally changes between different works and implementations are the criterions used when selecting data for human annotation and pseudo-labelling. Wang et al. select the most uncertain (or least confident) samples for human labelling and the most confident for pseudo-labelling (Keze Wang D. Z., 2017). The most classical criterions for assessing model confidence in classification are the Least Confidence, Margin Sampling and Entropy criterions (Keze Wang D. Z., 2017). The Least Confidence criterion establishes the confidence of a sample with the probability of the most probable class; the Margin Sampling with the difference between the two most probable classes (the bigger the margin, the higher the confidence) and the Entropy criterion with the entropy of the predicted distribution (a bigger entropy is associated with high uncertainty and a smaller entropy with high confidence). Sener and Savarese define Active Learning as a Core Set Selection Problem. This problem considers a fully labeled dataset and tries to choose a

subset of it such that the model trained on the selected subset will perform as closely as possible to the model trained on the entire dataset (Ozan Sener, 2017). That is, they consider representativeness when selecting instances.

Kao et al. present two metrics for measuring informativeness in object detection: Localization Tightness and Localization Stability (Chieh-Chi Kao, 2018). The former consists of measuring the overlapping ratio between the Region of Interest (in a two stages detector) and the final prediction, to estimate how tight the detected bounding boxes can enclose true objects. The tighter the bounding box, the more certain the localization (Chieh-Chi Kao, 2018). Localization Stability is based on whether the detected bounding boxes are sensitive to changes in the input image. Different amounts of gaussian noise (different variances) are added to the input image, and change in the predicted boxes is measured. A high variation on predicted boxes is related to a high uncertainty (Chieh-Chi Kao, 2018). We remark that they do not use pseudo-labelling. Wang et al. expand their previous work (Keze Wang D. Z., 2017) to the object detection task by incorporating Cross Image Validation as a measure of consistency for selecting pseudo-labels (Keze Wang X. Y., 2018). A region from an unlabeled image that has been predicted to contain class *j* is pasted into a labeled image that does not contain that class. This newly generated image is forwarded through the network. For this prediction to be considered consistent and reliable, that same class *j* must be detected in the new image, with an IoU higher than a certain threshold (typically set to 0.5) between the predicted bounding boxes in the original and new images.

3 Methodology

We aim to design and develop an object detector that detects three classes. The detector will first be trained using classical Fully Supervised Learning and a fully annotated dataset, with object-level annotations. That is, we have the ground-truth bounding boxes for each object in the images.

After that, the same base detector will be trained using Semi-supervised Learning. This will enable the incorporation of non-annotated data to the training process and should, at least theoretically, increase its performance. For this, we incorporate the two main SSL approaches presented in the last section: Consistency Learning and Self-supervised Learning.

In this work, we decided not to use Active Learning. We dispose of a non-annotated database that cannot be annotated (at least in the time required by the AL-pipeline) and can only be used as it is, as we do not have access to the labelling procedure. This makes it impossible to apply the iterative-training pipeline of Figure 13. However, in section 2.3.2 we described the principle of Active Learning for the sake of completeness.

3.1 Detector architecture (Faster R-CNN)

After several performance tests with Faster R-CNN and RetinaNet detectors we decided to use the former one as our base detector, as it performs better and has a lower training time. We recall that Faster R-CNN is a two stages detector, composed by a first stage that generates the Regions of Interest (RoI), and a second stage that performs the final detection (localization + classification).

3.1.1 Backbone: ResNet-34

We use a ResNet-34-based backbone. ResNets (Kaiming He X. Z., 2015) are deep CNNs that avoid the vanishing gradient problem by including skip connections between layers. When networks go too deep (more than 16 layers), the gradient of the initial layers (propagated from the last layers via Backpropagation) gets too small, which causes the initial layer's weights (often the most crucial ones) to be too hard to train and, therefore, degrades the performance. The basic building block of ResNets (Figure 14), called residual block, overcomes this by shortcutting the output of a layer with the output of the previous layer. This skip connection or identity mapping, which just adds the output of the previous layer, assures that the propagated gradient will not shrink and that initial layers can be trained properly.

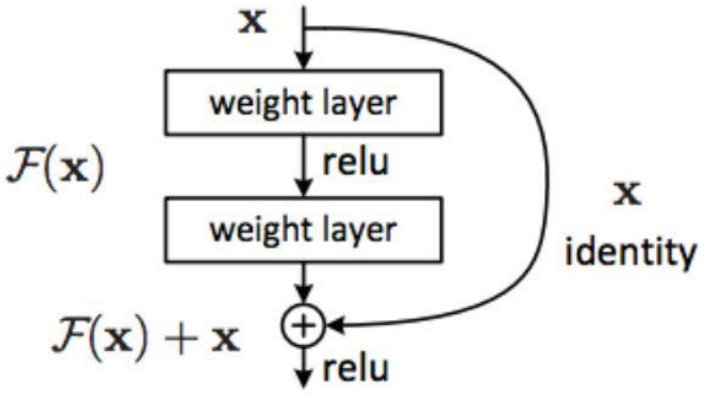

Figure 14: ResNet residual block. (Kaiming He X. Z., 2015)

ResNets enable to train deep models without degrading performances and are used in most of the recent object detection papers. The ResNet depth is set to 34 mainly for two reasons: (1) training deeper models as ResNet-50 or ResNet-101 is unfeasible given the size of our dataset, (2) we have a Resnet-34-based classifier trained on similar images to do transfer learning. Table 2 shows the architecture of ResNet-34.

Stem	7x7, 64, stride 2 3x3, max pool, stride 2
Res2	$\begin{bmatrix} 3x3, 64 \\ 3x3, 64 \end{bmatrix}$ x2
Res3	$\begin{bmatrix} 3x3, 128 \\ 3x3, 128 \end{bmatrix}$ x4

Res4	$\begin{bmatrix} 3x3, 256 \\ 3x3, 256 \end{bmatrix}$ x6
Res5	$\begin{bmatrix} 3x3, 512 \\ 3x3, 512 \end{bmatrix}$ x3

Table 2: ResNet-34 architecture. Brackets identify basic blocks.

3.1.1.1 CNNs and Receptive Field

The receptive field of a CNN is defined as the size of the region in the input that produces a feature in the output feature maps. It depends principally on the network depth and the type of the layers stacked, and changes if the network has multiple paths or skipped connections. Araujo et al. give a complete explanation on how to calculate receptive fields, which was used in this project (André Araujo, 2019).

In the case of Faster R-CNN, the receptive field of the RPN (Backbone + RPN head) defines the maximum object size that can be detected, as the detection of objects by the R-CNN block is limited to the RoI proposed by the RPN. By looking at the object size histograms of the images in the dataset (Figure 21 and Figure 24, presented in section 4.1), we can see that for detecting all objects (excluding some outliers) we should have a receptive field of around 1000 × 1000 pixels. By using the architecture detailed in Table 3, with ResNet-34 as backbone, we attain an RPN-receptive field of 1029 × 1029 pixels (André Araujo, 2019), which is enough for the size of the objects that we aim to detect. This is the architecture used in all the project.

Backbone	ResNet-34
RPN	Conv layer (channels=512, kernel_size=3×3, stride=1, padding=1) + 2 Conv layers (one for object/background prediction and another for bbox regression)
R-CNN (final detector)	RoI Align + 2 Conv layers (channels=512, kernel_size=3×3, stride=1) + 2 FC layers (one for class score - 4 channels - and another one for bounding box regression -12 channels -)

Backbone Receptive Field	965×965
RPN Receptive Field	1029×1029

Table 3: Faster-RCNN implemented architecture.

3.1.2 Faster R-CNN Hyperparameters setting

In this subsection, we explain and justify the election of some hyperparameters of the model and of the training loop. We detail the setting of the learning rate (LR), the optimizer, the RPN's anchor boxes (size, aspect ratio), the number of RoI considered in the RPN, and the post-processing stage.

3.1.2.1 Optimizer and learning rate

We do not use Adam or other adaptative optimizers, as they have recently been shown to find solutions with less generalization capabilities and lower performances in the test set (Ashia C. Wilson, 2018). Adaptative methods often display faster initial progress in the train set but are more susceptible to end up converging to local minima (Ashia C. Wilson, 2018). Furthermore, even though it is usually suggested that Adam does not require tuning, Wilson et al. also found that its performance changes significantly when modifying the initial learning rate and decay. Therefore, we use Stochastic Gradient Descent (SGD) as optimizer.

For the learning rate, we use a Warm Up Multi Step scheduler, as it is useful for training attention mechanisms (like Region Proposal Networks). This consists of a first stage of Warm Up, in which the learning rate starts in a low value (typically a 1/10 of the base learning rate) and linearly increases until reaching its base value. After several iterations, the learning is divided by a parametrizable value γ. An example is shown in Figure 15, with warm up period of 1k iterations, $\gamma = 10$ and base LR of 0.02. The learning rate is divided by γ in iterations 40k and 60k.

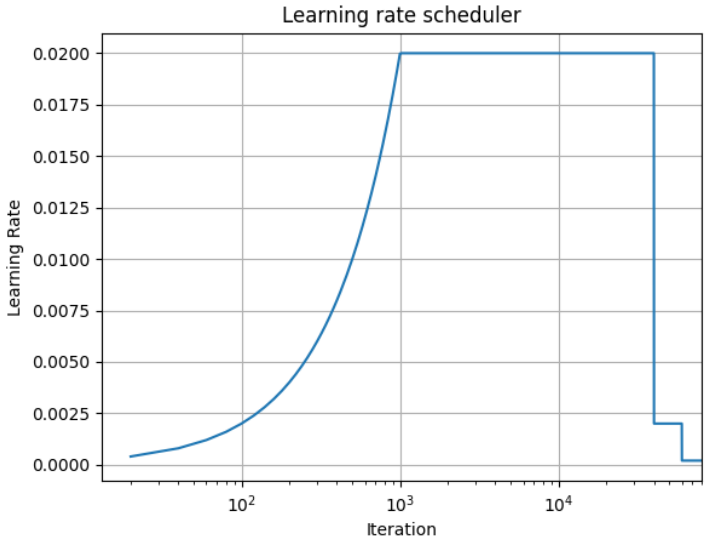

Figure 15: learning rate scheduler

3.1.2.2 RPN and anchor boxes

The number of objects per image in our dataset ranges from 1 to 3 and is typically of 1 object per image. This can lead to the class imbalance problem (negative or background regions dominate the loss) when training the RPN. In order to balance positive and negative examples, we fix a maximum number of 20 RoI per image in each mini-batch, and a ratio of positive examples of 0.5 (which means that half of the proposed regions have been classified by the RPN as “foreground”). If an image has less than 10 positive RoI, the mini-batch is completed with negative examples. These values were obtained experimentally after observing that there are, in average, 7 positive RoI per image. This same configuration (number of examples per mini-batch and positive ratio) was also used for training the R-CNN.

Finally, we only use squared anchor boxes (aspect ratio 1), as there are no rectangular ground-truths. The anchor boxes have the following sizes, in pixels: 32, 64, 128, 256 and 512.

3.1.2.3 Post-processing

Output bounding boxes are first filtered by score, in order to reduce false positives. The score of a bounding box comes from the output of the classification layer and is the

confidence that the detector has in associating that region to a certain class. Bounding boxes that have a score smaller than a threshold (fixed to 0.01) are eliminated.

Non-maximum Suppression (NMS) is then applied to the remaining bounding boxes in order to reduce redundancy. NMS consists in calculating the IoU between all the predicted boxes and removing all the overlapping boxes, keeping the ones with the highest scores. The IoU threshold is set to 0.3.

3.2 Semi-supervised Learning: pipeline and losses

We incorporate unsupervised images to the training loop by implementing the Semi-supervised pipeline of Figure 16 (Peng Tang, 2020). Here, the two main approaches in Semi-supervised Learning described in section 2.3.1 are incorporated: Consistency Learning and Self-supervised Learning. For images coming from the labeled dataset, all five losses are calculated (supervised losses in blue and unsupervised losses in yellow); for images coming from the unlabeled dataset, only unsupervised losses are calculated (in yellow).

Noise is added to the feature maps (at the output of the RoI Align block), generating *N* noisy feature maps per patch. These *N* versions are forwarded, as the original non-noisy feature maps, through the R-CNN; we then obtain *N+1* class-probability vectors and bounding boxes for each patch. With these outputa, we calculate consistency, contrastive and location losses. We remark that this pipeline is focused on modifying the R-CNN because, as will be shown in the section 4.2.1, the second stage of the detector has lower a performance than the first stage and dominates the loss.

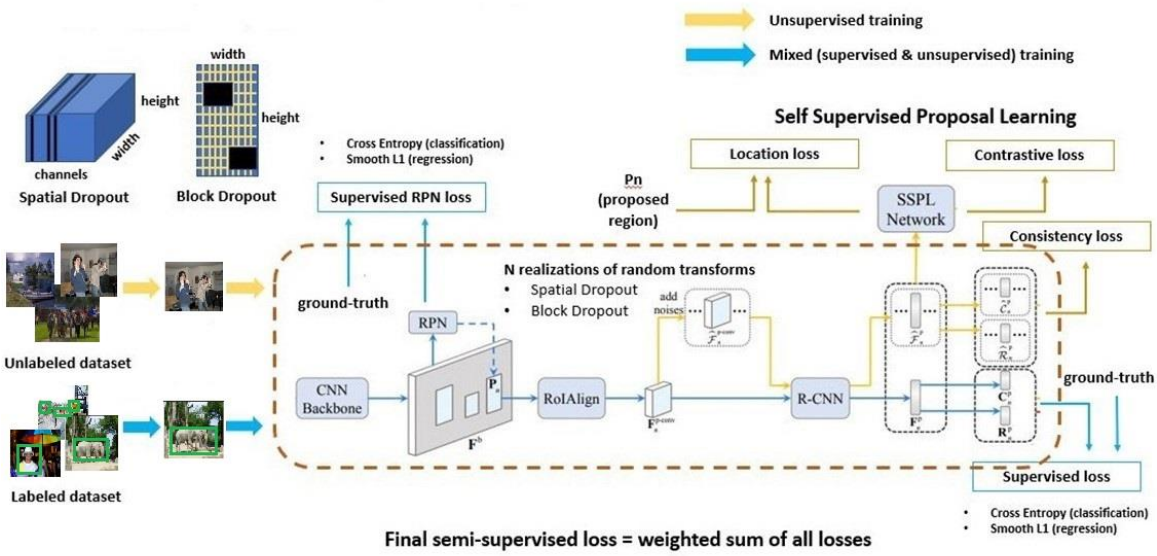

Figure 16: Semi-supervised Learning pipeline.

3.2.1 Consistency Learning

The consistency loss enforces class and bounding box predictions from noisy feature maps to be similar to the noiseless predictions, following the assumption that the features space is smooth. The total consistency loss is the weighted sum of the classification and regression consistency losses, defined by

$$L^{cons-class} = \frac{1}{K}\sum_{k=1}^{K} KL(c||\hat{c}_k)$$

$$L^{cons-reg} = min_k\left(smooth_{l1}(r - \hat{r}_k)\right)$$

Equation 12: consistency losses: classification (above) and localization (below)

where c and r are the noiseless class-probability vector and the bounding boxes predictions, $\hat{c}_k$ and $\hat{r}_k$ are the k[th] noisy versions, and $KL(p||q)$is the Kullback-Leibler divergence, a measure of distance between two probability distributions.

3.2.2 Self-supervised Learning

The Self Supervised Proposal Learning (SSPL) model uses a proposal location loss and a contrastive loss to learn context-aware and noise-robust proposal features, respectively (Peng Tang, 2020). In order to do that, it solves two pretext tasks: proposal location prediction and instance discrimination. For this, it incorporates a SSPL network.

For solving the proposal location prediction task, the feature vectors are forwarded through two convolutional layers (output size 1024 and 4, respectively) and a sigmoid layer to compute proposal locations. That is, we use the feature vectors to predict the location of the patch in the original image. The proposal location loss is defined as

$$L^{self-loc} = \frac{1}{K+1}\left(||L_n - \tilde{P}_n||_2^2 + \sum_{k=1}^{K} ||\hat{L}_{nk} - \tilde{P}_n||_2^2\right)$$

Equation 13: Self-learning localization loss.

where $\tilde{P}_n$ are the normalized coordinates of the nth proposed region (output of the RPN) i.e., divided by the image dimensions, L_n is the proposal location prediction using the noiseless feature maps, and $\{\hat{L}_{nk}\}_{k=1}^{K}$ the set of predictions using the noisy feature maps. By optimizing this loss, the network learns global information of the image (Peng Tang, 2020).

For solving the instance discrimination task, the feature vectors are projected to an embedded space of proposal features (of length 128) by a Fully Connected layer and an L_2 normalization layer. The contrastive loss is written as

$$L^{self-cont} = \frac{-1}{K}\left(\sum_{k=1}^{K} log \frac{\exp\left(\frac{\hat{F}_{nk}^{e\ T} F_n^e}{\tau}\right)}{\sum_{n'} \exp\left(\frac{\hat{F}_{nk}^{e\ T} F_{n'}^e}{\tau}\right)}\right)$$

Equation 14: contrastive loss.

where F_n^e is the original embedded proposal feature vector for patch *n*, $\{\hat{F}_{nk}^e\}_{k=1}^{K}$ the set of noisy proposal feature vectors for patch *n*, and τ a temperature parameter that is set to 0.1. By optimizing this loss, noisy proposal features are enforced to be closer to their original

proposal features than to other proposal features (in other patches of the same image). The network then learns noise-robust features, which increase the model's sturdiness towards noise in the detection task (Peng Tang, 2020).

3.2.3 Feature maps random transforms

Two types of noise specially well adapted to CNNs were initially used: Spatial Dropout and Block Dropout. When working with images and feature maps, randomly dropping out activations as in classical dropout is not effective in removing semantic information (Golnaz Ghiasi, 2018). In CNNs, features are correlated spatially and dropping one activation does not prevent its information to remain in the feature maps. Spatial Dropout and Block Dropout are two alternatives to classical Dropout that enable to avoid that problem. In tensors of dimension C × W × H, Spatial Dropout (Jonathan Tompson, 2015) consists of randomly dropping an entire channel with a probability *p*. On the other hand, Block Dropout drops blocks of pixels of a given size with a probability *p*, independently for each channel (Figure 17). In Figure 17, we can see that dropping contiguous regions can remove certain semantic information (e.g., head or feet) that regular Dropout cannot (Golnaz Ghiasi, 2018).

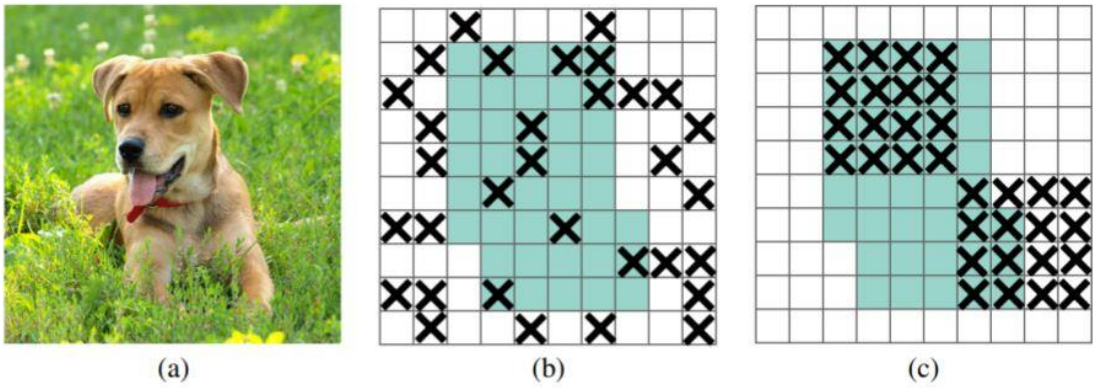

Figure 17: (a) input image to Convolutional Neural Network. The green regions in (b) and (c) show the activation units which contain semantic information in the input image. (b) shows the activation dropped with regular Dropout and (c) with Block Dropout. (Golnaz Ghiasi, 2018)

3.2.4 Ramp Up weight for unsupervised terms

To combine the supervised and unsupervised terms (consistency and self-supervised) into a final loss, we multiply the unsupervised terms by a varying factor *w(t)*, which follows a

Ramp Up regime (Samuli Laine, 2017). This factor starts in 0 and reaches 1 following a Gaussian curve during the first *M* iterations, which makes that in the beginning the total loss and its gradients are dominated by the supervised loss component. After the Ramp Up period *M*, the factor is stabilized in 1 and all terms contribute equally to the final loss. This is important to prevent the network from getting stuck in degenerate solutions or local minima (Samuli Laine, 2017).

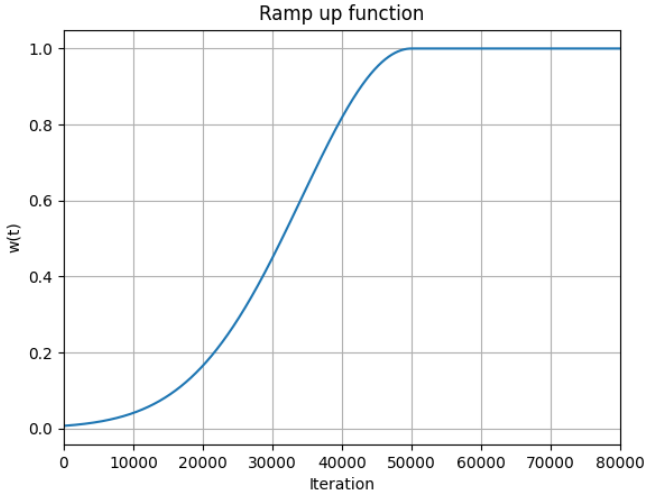

Figure 18: Ramp Up function for unsupervised loss weight with period 50k.

3.2.5 Final loss function

The final loss function is the weighted sum of all the aforementioned terms:

$$L = L^{sup-class} + L^{sup-reg} + w(t)(\lambda^{cons-class} L^{cons-class} + \lambda^{cons-reg} L^{cons-reg} + \lambda^{self-loc} L^{self-loc} + \lambda^{self-cont} L^{self-cont})$$

Where we set $\lambda^{cons-class} = 1$, $\lambda^{cons-reg} = 1$, $\lambda^{self-loc} = 0.25$ and $\lambda^{self-cont} = 1$ as in the original paper (Peng Tang, 2020). We note that the unsupervised terms should not modify the RPN. In the first place, the RPN should not adapt itself to provide RoI that make the pretext tasks easier. Secondly, in consistency calculation the noise is added to the RoI and not to the input image: the RPN should not adapt itself to providing RoI for which the final detection result is noise robust. The RPN gradient with respect to the unsupervised terms was deactivated.

3.3 Dynamic batching

To manage the different image sizes when stacking them in a batch for training, we implement a data loader that dynamically adapts the shape of the input tensor in the height and width dimensions (dynamic batching). That is, the input tensor shape is not fixed and depends on the images in the batch, which are randomly selected. Three different resizing strategies were tested: match with max, match with random and pad up to max. Match with max consists of resizing all the images to the maximal height and width in the batch; match with random of resizing all the images to a height and width of a randomly chosen image from the batch (which introduces some random variations and could improve performances); and pad up to max consists of zero padding all the images up to the maximal height and width. We tested the three approaches, and obtained that they lead to similar performances if the aspect ratio is kept when resizing images (which is done by changing the image resolution and applying zero padding). Performances decrease when the aspect ratio is not maintained, as deforming some objects negatively affects their detection.

4 Experiments & results

4.1 Datasets

Two datasets of 16-bytes-precision images were used in this project: a labeled dataset that contains images with object-level annotations (bounding boxes around objects), and an unlabeled dataset that contains the images without annotations.

4.1.1 Labeled dataset

We have access to a database of 1145 annotated images. We distinguish three classes of objects, each containing several subclasses. The labels are given in the form of four bounding box coordinates that contain the annotated objects in the image, together with their corresponding class and subclass. Figure 19 and Figure 20 show the histograms of the image height and width. We can see that the height of the images is mostly constant, but there is a big variation in the width dimension. Figure 21 shows the histogram of the bounding box size, where we can see that most bounding boxes have a width and height smaller than 600 pixels (we recall that all the ground-truth bounding boxes are squares). Figure 24, Figure 25 and Figure 26 show the per-class bounding box size histograms, where we can see that class 2 objects are usually smaller than class 1 ones, even though there are a few examples of big class 2 objects. The class 3 objects histogram is not very representative, as there are not enough instances of that class in the labeled database.

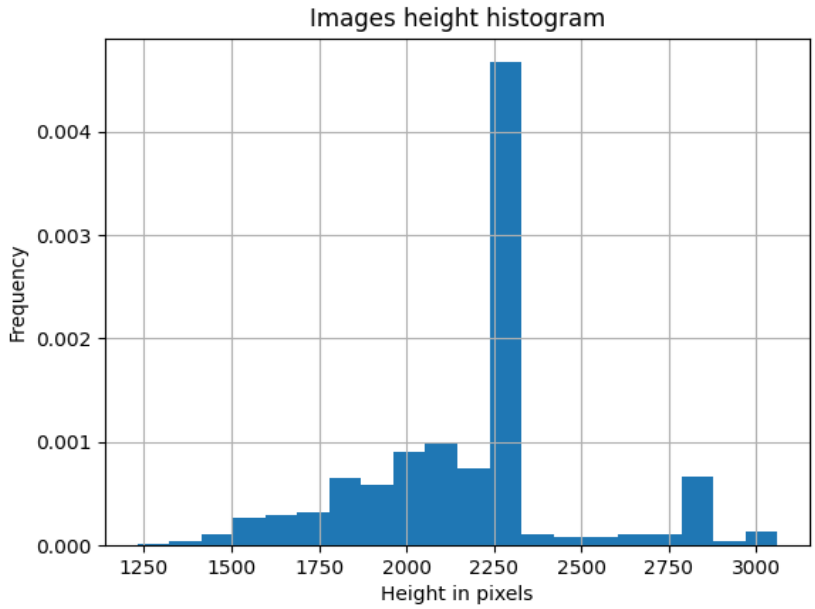

Figure 19: image height histogram.

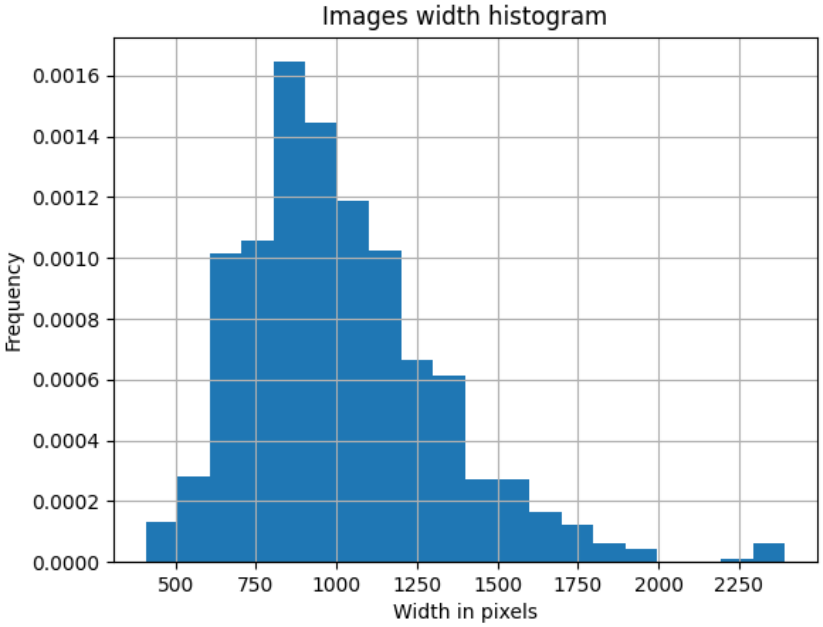

Figure 20: image width histogram.

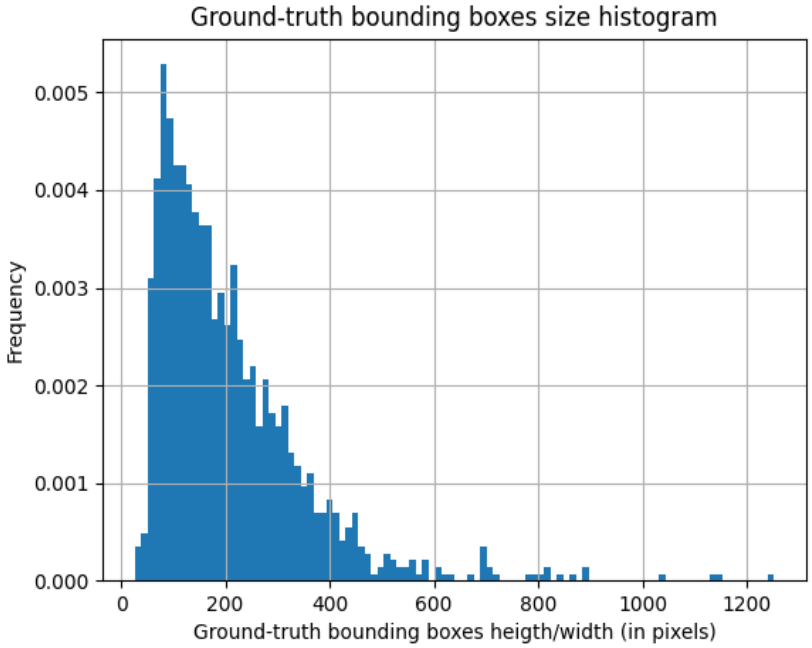

Figure 21: Histogram of the ground-truth bounding box size, in pixels (all ground-truths are squared)

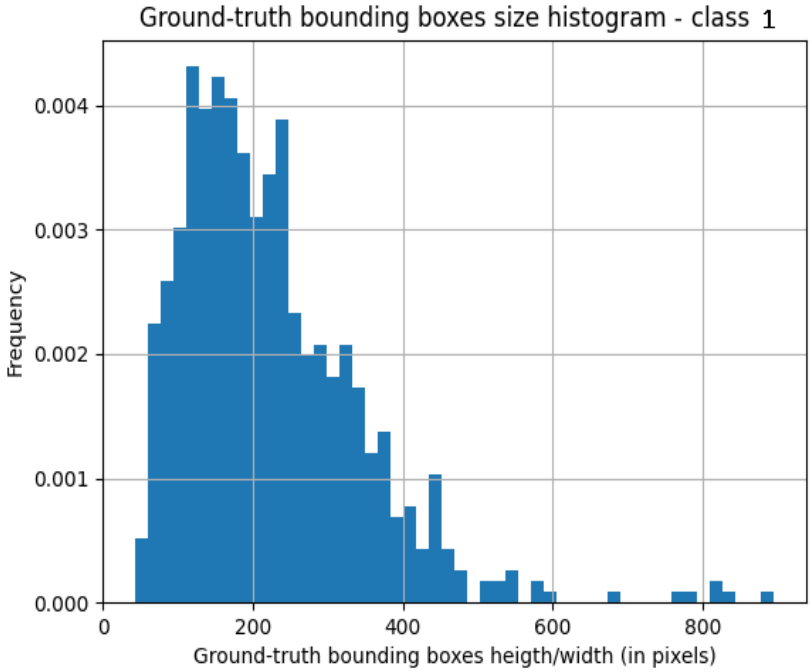

Figure 22: ground-truth bounding box size histogram, class 1.

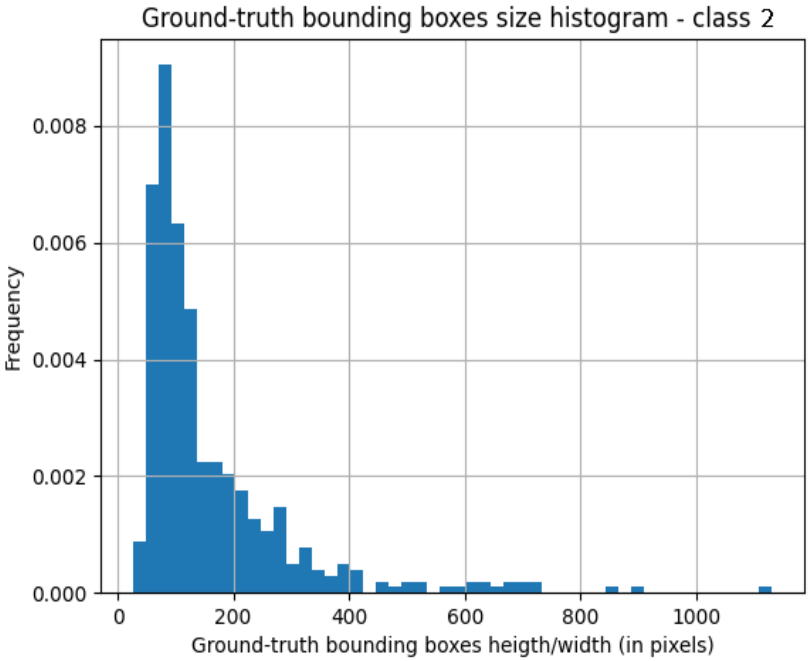

Figure 23: ground-truth bounding box size histogram, class 2.

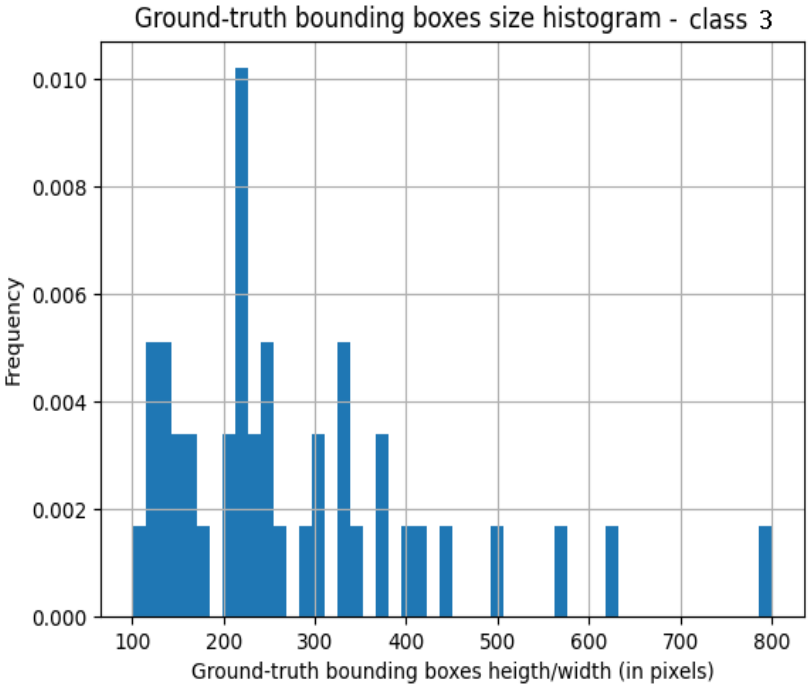

Figure 24: ground-truth bounding box size histogram, class 3.

4.1.2 Unlabeled dataset

We also have access to an unlabeled dataset of 8470 images, 6294 from which do not have any objects. The remaining 2176 images of this subset probably have objects in them, even though they are not annotated.

4.1.3 Stratified train-validation-test split

As in every Machine Learning project, we split the labeled dataset in train, validation, and test sets. However, probability distributions across different classes and subclasses may not be maintained when doing a regular data train-validation-test split (randomly assigning images to each subset), which can be especially harmful given the small number of images that we have. To prevent this, we introduce the stratified train-validation-test split. This technique, which intends to preserve in the validation and test sets the probability distribution that has been learnt in the train set, assigns images to each subset depending on the classes and subclasses that they contain.

Each image is assigned to one and only one class-subclass pair, based on the number of objects of that class and subclass that are found in their annotations (if two class-subclass pairs appear the same number of times, the image will be assigned to the “rarest” class-subclass, that is, the one that is the least represented in the entire dataset). Once all images have been assigned to a class-subclass pair, we can construct a histogram that shows the total number of images of each class-subclass. We then split them in train, validation and test sets by making sure that the class-subclass distribution is maintained. If some class-subclass pair is associated to too few images, it is impossible to maintain the distribution between the subsets and all its images are assigned to the train set. Table 4 shows the number of images in each split. Figure 25, Figure 26 and Figure 27 show the obtained histograms for the three subsets where the split proportions are, approximately, 0.8 for the train set, and 0.1 for the validation and test sets. We remark that some subclasses that are in the train set could not be included in the validation and test sets.

	Total	Class 1	Class 2	Class 3
Train	**930**	492	406	32
Val	**100**	54	42	4
Test	**115**	61	48	6

Table 4: number of images in train, validation, and test set, by representative class.

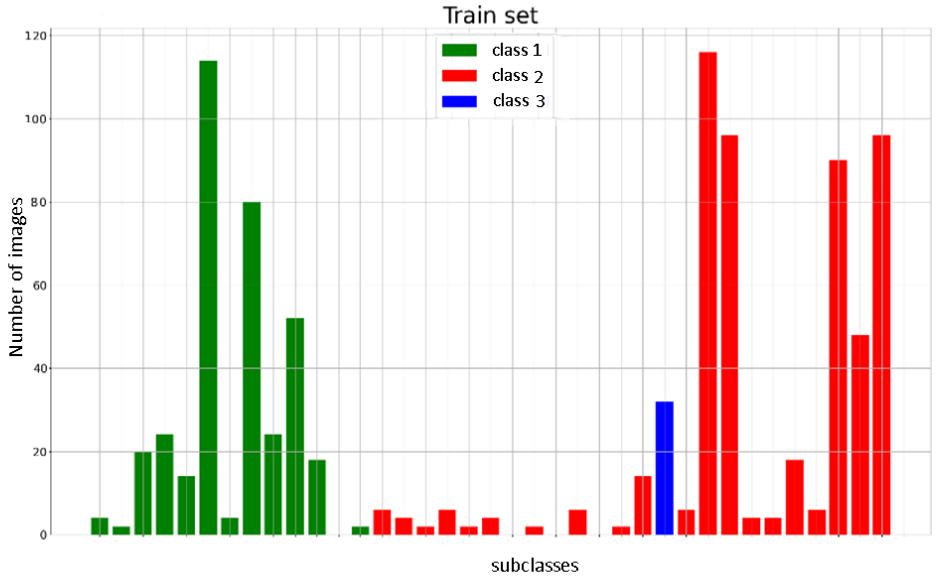

Figure 25: class-subclass distribution in the train set. Classes are shown in different colors and subclasses across the abscise axis.

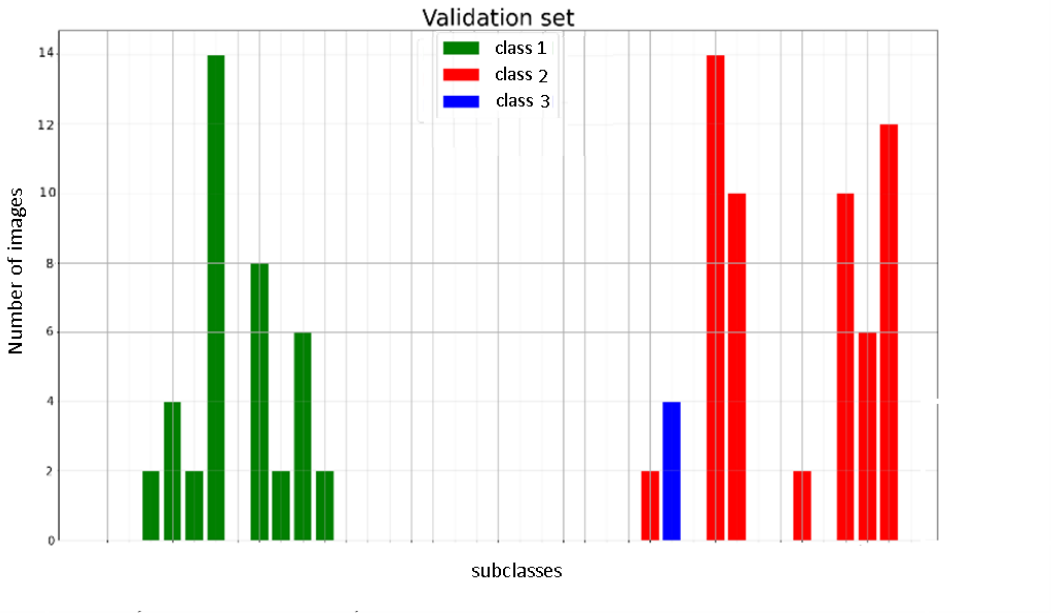

Figure 26: class-subclass distribution in the validation set. Classes are shown in different colors and subclasses across the abscise axis.

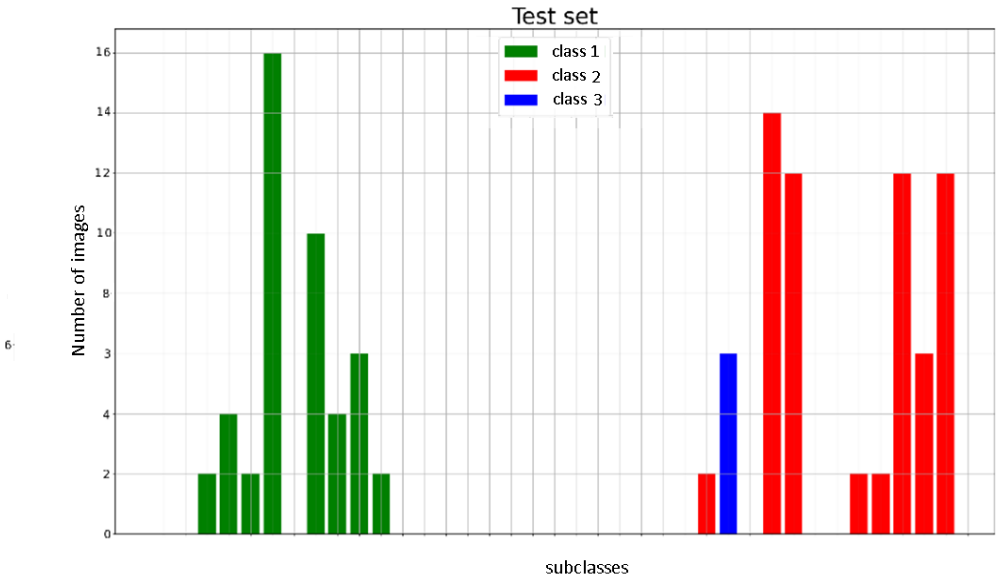

Figure 27: class-subclass distribution in the test set. Classes are shown in different colors and subclasses across the abscise axis.

4.2 Results & analysis

The detector was first trained using Fully Supervised Learning in the labeled dataset. In order to improve its performance, the Semi-supervised Learning techniques described in the section 3.2 were incorporated to the training loop. The Fully Supervised-trained model is used as baseline to assess the improvements of SSL.

We also remove the class 3 from the possible object classification results, since the labeled dataset does not contain enough examples of this kind of objects. This prevents us from having statistically significant results for this class. We thus consider the class 3 objects as being of class 1, as those two types of objects have lots of shared characteristics.

4.2.1 Fully Supervised training

The detector was trained in a Fully Supervised way during 90k iterations, using the Warm Up Scheduler of Figure 15 ($\gamma = 10$, base LR of 0.02, warm up period of 1k, and dividing steps of 40k and 60k). All the images used for the training contained objects.

Figure 28 and Figure 29 show the evolution of all the losses of Faster R-CNN during training. We remark that the R-CNN losses are higher than those of the RPN and that they dominate the total loss. This is the reason why we concentrated our efforts in improving the R-CNN to increase the global performance.

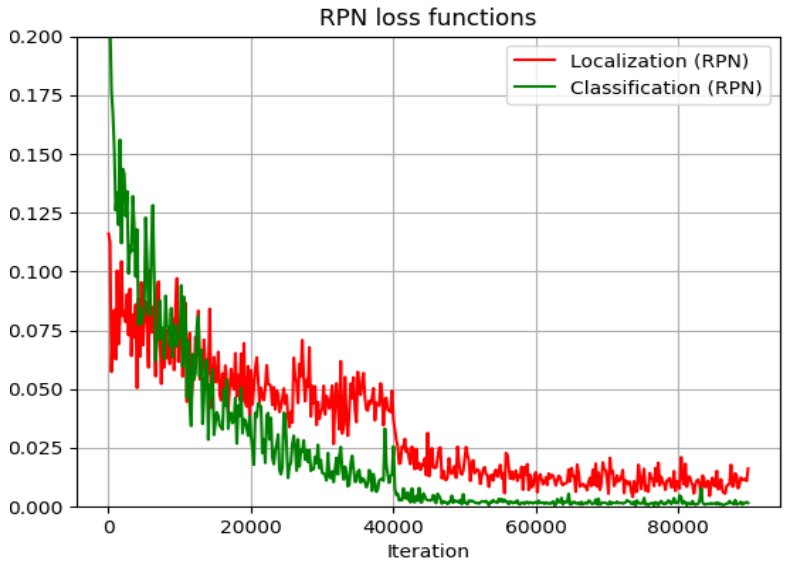

Figure 28: RPN loss functions.

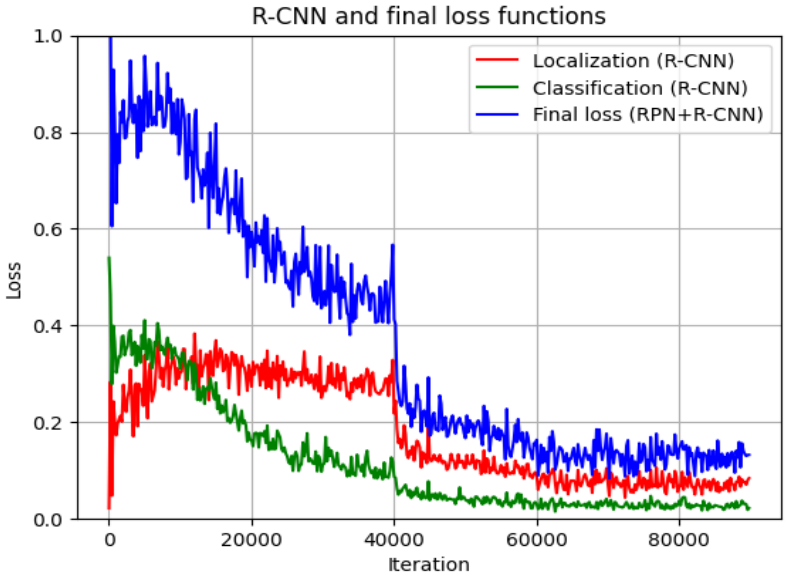

Figure 29: R-CNN and total loss functions.

4.2.1.1 Batch size and normalization technique (BN/GN) optimization

Two hyperparameters were optimized experimentally: the batch size and the normalization technique.

Batch size

In each training iteration, a fixed number of randomly picked images are forwarded through the network, and are used to estimate the loss function and its gradient (Stochastic Gradient Descent optimizer). This fixed number is known as batch size. For small batches, the estimation of the loss and specially of its gradient are very poor, leading to an unstable and noisy learning and to poor results. On the other hand, big batch sizes are computationally prohibitive and lead to the network having less generalization capabilities: small batches introduce noise in the training, which regularizes the learnt weights.

Batch Normalization (BN) vs. Group Normalization (GN)

The input of a convolutional layer consists of several feature maps. If the mean value of one of the feature maps is significantly higher than those of the others, the layer's output will tend to depend only on the feature map with the biggest mean. This can be easily avoided by including a normalization layer after each layer in the neural network. The most common normalization technique used is the Batch Normalization Layer (BN), which normalizes each channel (or feature map) independently and across all inputs in the batch. It also includes two learnable parameters ß and γ (Equation 15), which represent a scale and a shift. In Equation 15, μ_B and σ_B^2 are the mean and variance of the batch, and ε is a small constant for preventing division by zero.

$$y_i = BN_{\gamma,ß}(x_i) = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + ß$$

Equation 15: batch normalization.

Batch Normalization gives very good results when the batch size is big enough, but it fails for smaller batches, when the estimation of the mean and standard deviation across the inputs are not good. An alternative to BN is Group Normalization (GN), which groups neighboring channels and calculates the mean and variance for each group of channels and for each input (batch component) independently (Yuxin Wu, 2018). This eliminates the negative effects of small batch sizes. Figure 30 conceptually compares BN and GN. The only difference that GN introduces in Equation 15 is that the mean and variance are calculated across groups and not across the entire batch. The number of groups *G* is an hyperparameter, which is commonly set to 32. GN usually outperforms BN for small batches but fails to match the performance of BN for big batches.

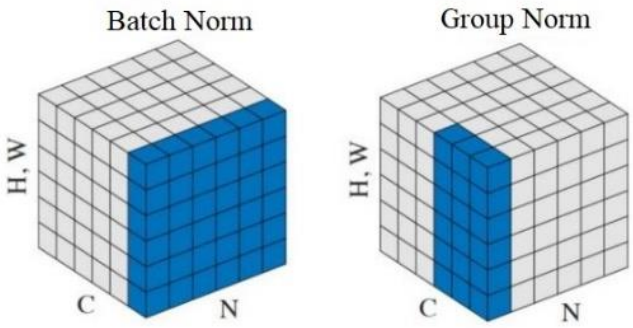

Figure 30: Batch and Group Normalization. Source: https://towardsdatascience.com/an-alternative-to-batch-normalization-2cee9051e8bc

Optimization

We train the model and vary the batch size and the normalization technique. We set the number of groups parameter in GN to its default value in the paper by Wu et al. (G=32). Given the size of the images, the RAM memory of the GPU (Graphics Processing Unit) limited the maximum batch size to 4 images, which is a pretty small batch. In order to choose the most optimal values, we look at the parameters that maximize the Average Precision in the validation set (Figure 31). We note that using Group Normalization leads to better performance than using Batch Normalization, which was expected as this is generally the case for small batches. We also remark that a batch size of 4 clearly outperforms a batch size of 2. Batch sizes of 1 and 3, which are not represented in this plot, lead to poorer performances. We keep a batch size of 4 with Group Normalization, as this is the configuration that maximizes the Average Precision.

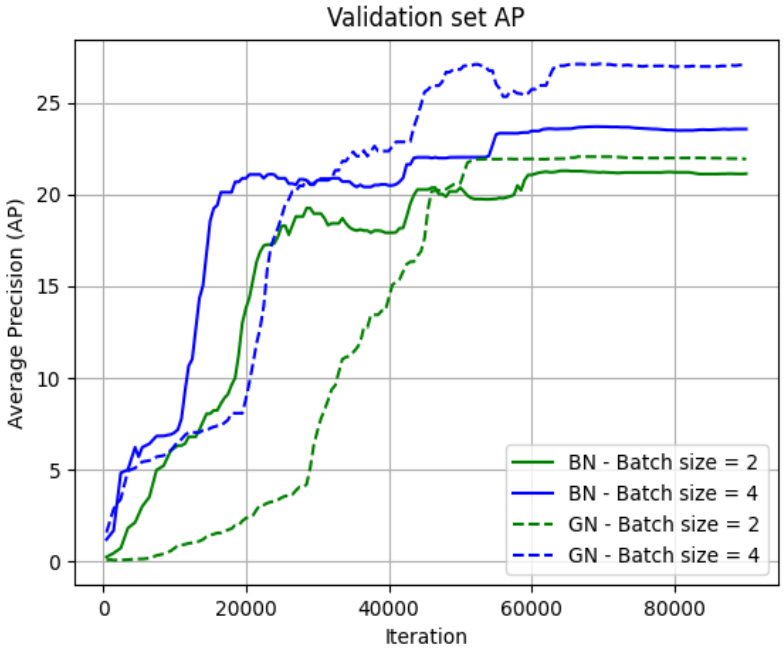

Figure 31: AP in the validation set for different batch sizes and normalization techniques.

4.2.2 Semi-supervised training

We train the same model of the last subsection under the Semi-supervised scheme of Figure 16 and we evaluate the improvements of SSL using the Fully Supervised model as baseline.

4.2.2.1 First experiments

The first experiments that we conduct use Block and Spatial Dropout (drop probability of 0.01) as random transformations of the feature maps. They consist of a data ablation study (using only the labeled dataset), and of experiences using the labeled and the unlabeled datasets.

Labeled dataset ablation study

In order to test the performance of the proposed Semi-supervised method, we perform a data ablation study. This consists of removing the annotations of a fraction of the labeled dataset and training the model with this reduced dataset. The images that were removed from the labeled dataset are incorporated to the unlabeled dataset and are only available for

Semi-supervised training. We remove the 25%, 50% and 75% of the dataset, train the model in both Fully Supervised and Semi-supervised ways and calculate the Average Precision in the validation set. In these experiments, we only use the consistency loss during the Semi-supervised training. Training was performed using the same learning rate scheduler than in the last section (Figure 15) and a Ramp Up of period 50k for the unsupervised terms in the loss (Figure 18).

We can see from Figure 32 (in which we plot the AP in the validation set) that, in this ablation study, the Semi-supervised training does not improve the performance of the Fully supervised training. This can have different origins. One possibility is that for experiencing the benefits of Semi-supervised Learning the unlabeled dataset must be much larger than the labeled dataset.

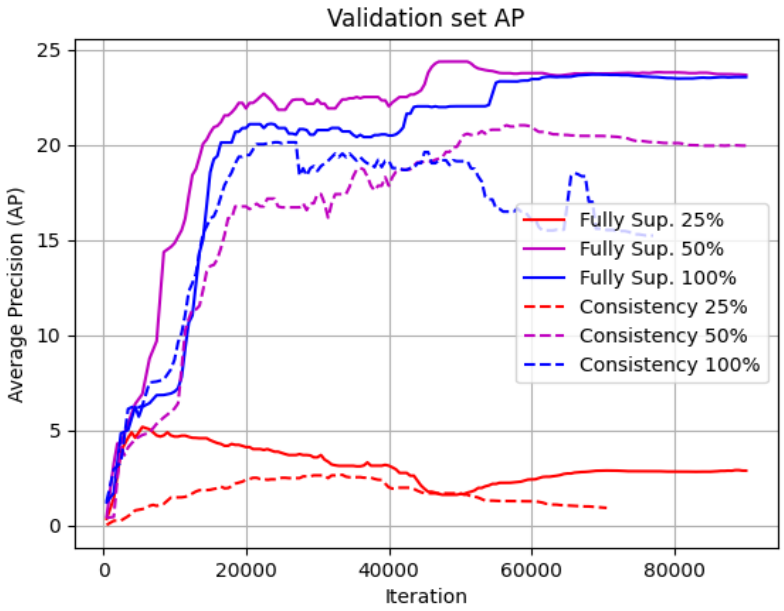

Figure 32: AP in the validation set for the data ablation study.

Complete dataset experiments

In order to overcome this problem, we include the unlabeled database of 8470 images for Semi-supervised training. We compare the Fully Supervised model with two Semi-supervised models: one using only the consistency loss and another using the consistency and self-supervised losses (complete Semi-supervised pipeline of Figure 16). We train the Semi-supervised models during 120k iterations (as they use much more data for training)

and use a Ramp Up of period 70k. Figure 33 shows the AP in the validation set for these configurations. We can see the effect of incorporating the unlabeled dataset, which increases the AP of the model implementing the consistency loss from 20 to 21.5. The overall AP with respect to the fully supervised-trained model is still not improved when using Consistency Learning or Self-supervised Learning.

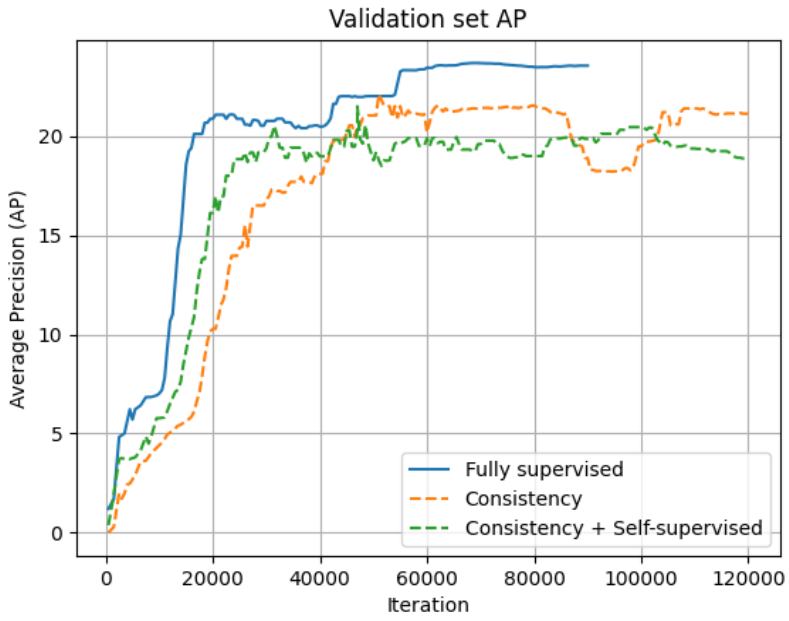

Figure 33: Average Precision (AP) in the validation set.

In Figure 34 and Figure 35 we show the AP per-class. We remark the different effect that the Semi-supervised training has in the two classes: it enhances the performance for class 1, but it lowers it for class 2 objects. In order to understand this, we must analyze the effects of the noise used for forcing consistency during training. The Block Dropout noise works by setting to zero an area of continuous pixels in the feature maps. If the object contained in that part of the image has not much spatial redundancy, applying this noise would make it very difficult, if not impossible, to recognize it. By looking at the images, which could not be included in this manuscript, we learn that Class 1 objects have a lot spatial redundancies, which makes them robust to spatial noises; Class 2 objects, on the contrary, seem not to be well suited to that kind of noise. The conclusion that we extract from here is that we need to modify the transformations applied to the feature maps when forcing consistency.

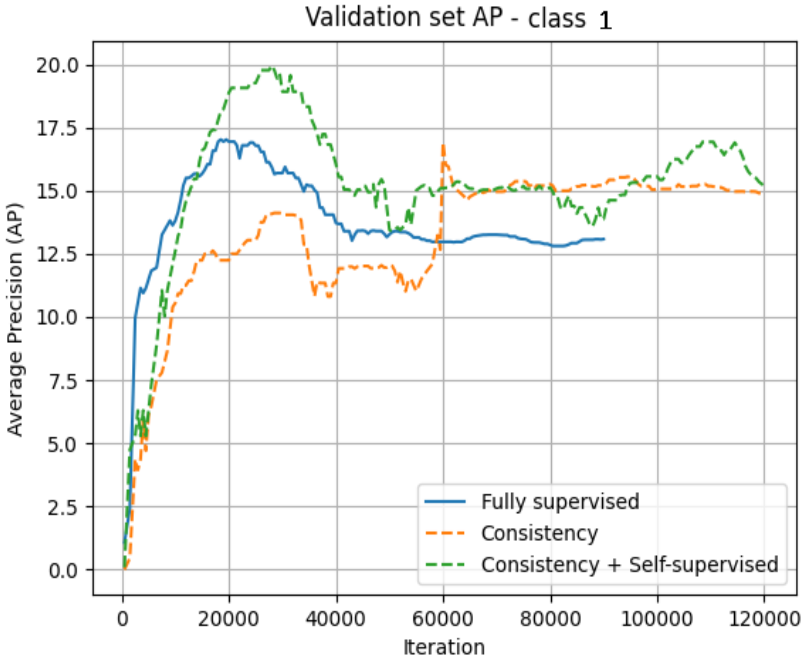

Figure 34: AP in the validation set, class 1.

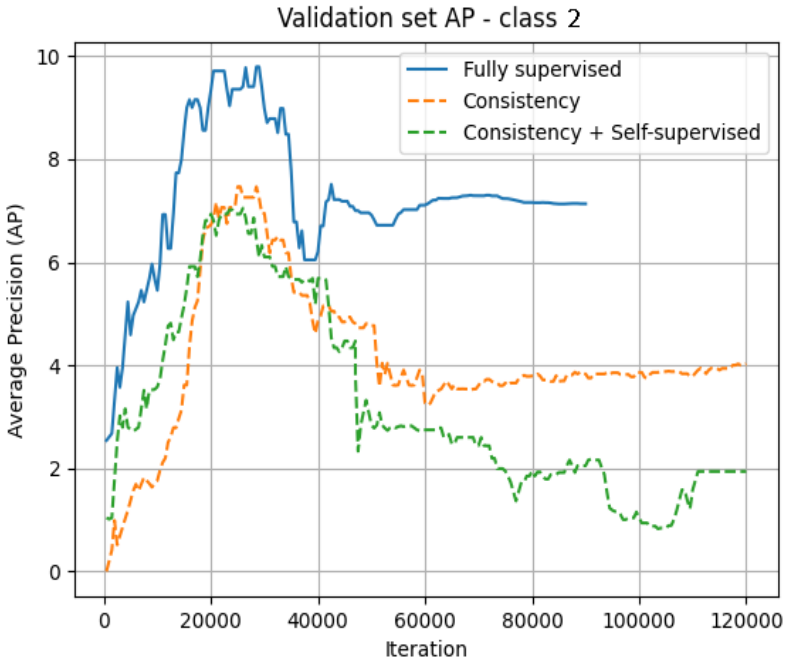

Figure 35: AP in the validation set, class 2.

4.2.2.2 Augmentation of the validation set

After performing the previous experiments, we were able to retrieve 292 additional annotated images. We incorporate them to the labeled dataset by splitting them between the train and validation sets in a stratified fashion (no images were added to the test set). In order to have more statistically significant evaluation results, we make sure to have a validation set of at least 20% of the labeled dataset, for which we destine more images to the validation set than to the train set. Table 5 shows the final number of images in the labeled and unlabeled datasets for the different splits (train, validation and test). It also shows the percentage of images in each split of the labeled dataset. We recall that the unlabeled images can only be used for training.

	Labeled dataset		**Unlabeled dataset**
	# images	**Percentage**	**# images**
Train	962	0.67	8470
Val	360	0.25	-
Test	115	0.08	-
Total	1437	1	8470

Table 5: image distribution in the train, validation, and test subsets for the labeled and unlabeled datasets.

4.2.2.3 Consistency with geometrical transforms

With this new dataset and after the results obtained in the last section, where we showed that Spatial and Block Dropout are not well adapted to Class 2 objects, we decided to use geometrical transformations in the feature maps to calculate the consistency loss. Although this is not the most typical use of the consistency loss, it has been used in natural images (Jisoo Jeong, 2019) with positive results.

Convolutions are naturally invariant to translations, which makes CNN-based detectors equivariant to translations. This means that if an object is translated in the input image, the detection result will be translated too. However, convolutions are not equivariant to non-translational transformations (like rotations or flips), meaning that if one rotates the input image, the feature maps do not necessarily rotate or flip in a meaningful way (Jisoo Jeong,

2019). We can force equivariance by applying a transform π_i to a feature map and forcing consistency between the output of the original feature map and the output of the transformed feature map, after applying to it the inverse transform π_i^{-1}.

We force consistency for vertical and horizontal flips of the feature maps. Figure 36 compares the AP in the validation set for Fully Supervised training, Semi-supervised training with consistency loss, and Fully Supervised training using horizontal and vertical flip as data augmentation. We can see that the model trained with the consistency loss outperforms by 4 AP points the two others. Figure 37 and Figure 38 show the same three curves in a per-class fashion. The consistency-trained model outperforms the two others for both classes, doubling the AP for Class 2 objects. We also tested consistency with rotations of 90°, 180° and 270° but did not obtain good results.

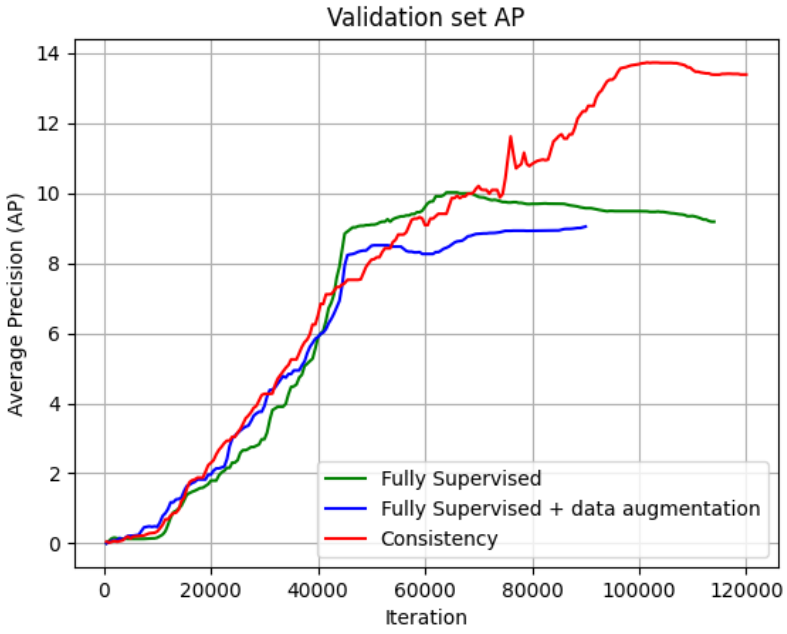

Figure 36: Average Precision (AP) in the validation set.

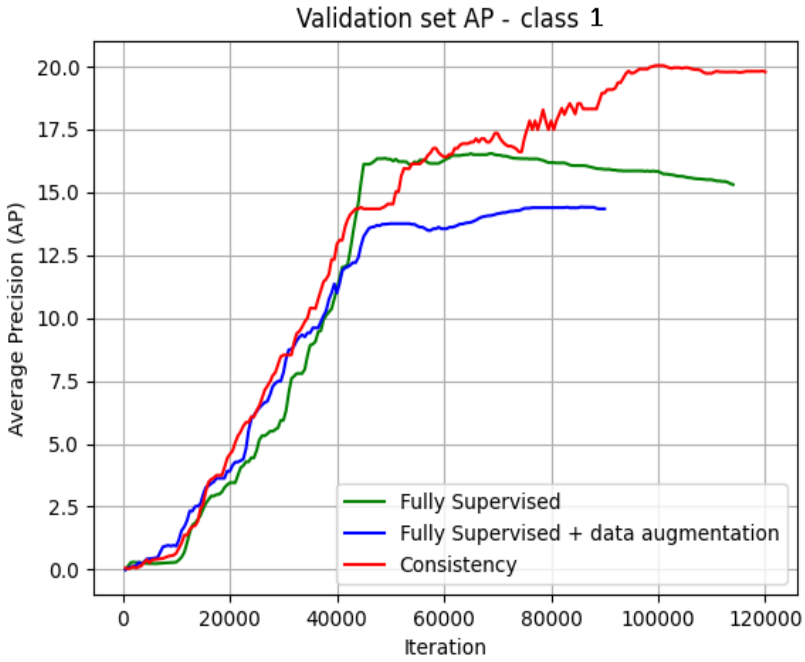

Figure 37: AP in the validation set, class 1.

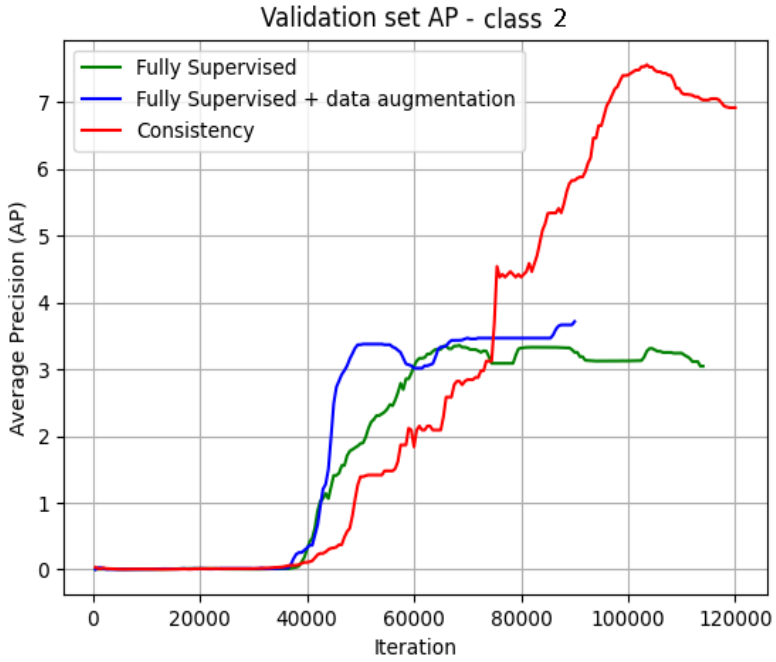

Figure 38: AP in the validation set, class 2.

4.2.2.4 Consistency with less aggressive types of noise

We also forced consistency with other types of noise in the feature maps. As Spatial and Block Dropout showed not to be adapted to Class 2 objects, we tested less aggressive types of noise: regular Dropout and Additive White Gaussian Noise (AWGN). Figure 39 compares the general AP in the validation set for a Fully Supervised training and when using the Consistency loss with those kinds of noise. The drop probability of the Dropout noise is $p = 0.01$ (as in previous experiments) and the standard deviation of the AWGN is $\sigma = 10^{-3}$. The value of σ was carefully set by looking at the range of values of the feature maps.

In Figure 39 we can see that Consistency Learning with both types of noise outperforms the fully supervised-trained model, and that the best results are obtained when using AWGN. In Figure 41 we can see that, even though CL with regular Dropout outperforms the fully supervised-trained model when both classes are considered, it still does not get good results for Class 2 objects. On the other hand, AWGN seems well adapted for the two classes (Figure 40 and Figure 41). We can then conclude that no form of Dropout (Spatial, Block or regular) is well adapted to Class 2 objects and, as a consequence, that Dropout noise is not well adapted to our problem. When comparing the results with Consistency Learning using AWGN and flips (subsection above), we can see that flipping slightly outperforms AWGN by a difference of around 1 AP point for Class 1 objects, and of 1.5 points for Class 2 objects.

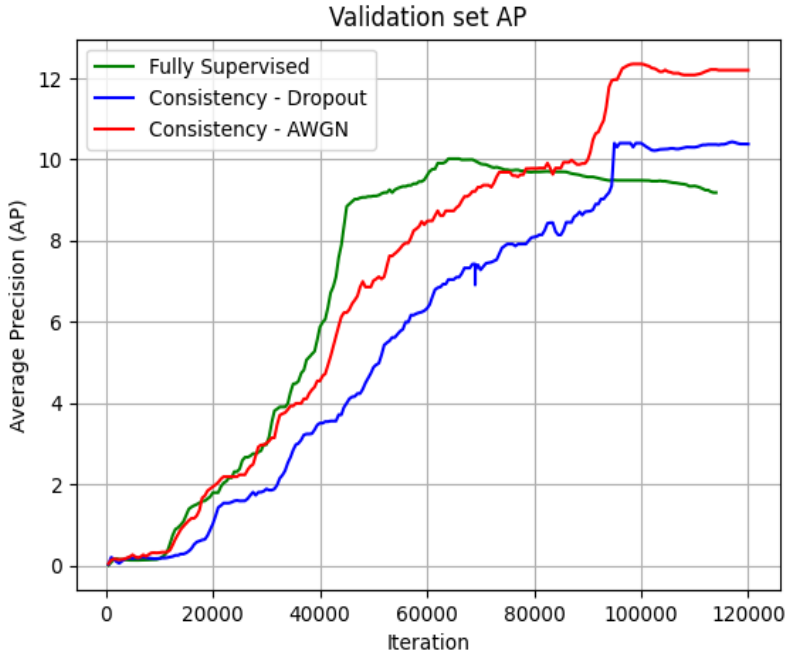

Figure 39: Average Precision (AP) in the validation set.

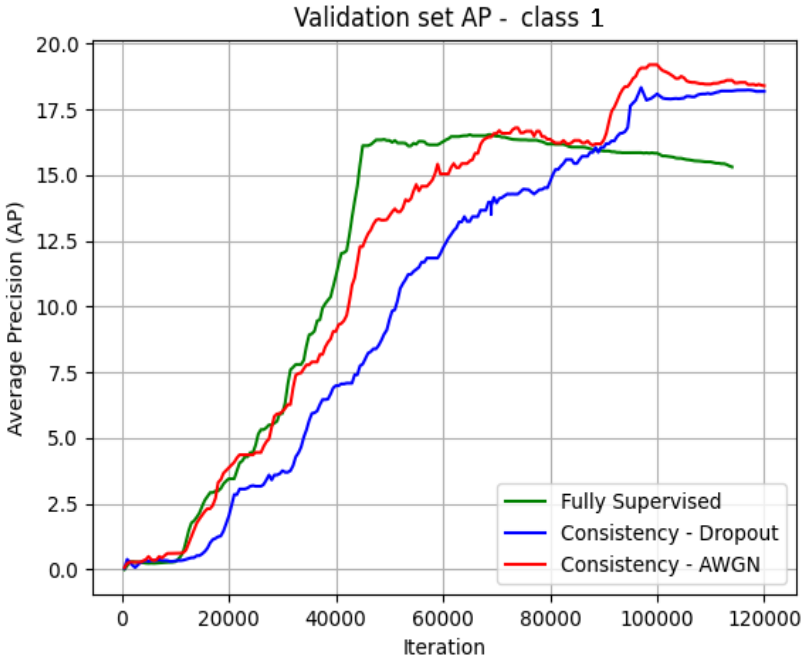

Figure 40: AP in the validation set (class 1).

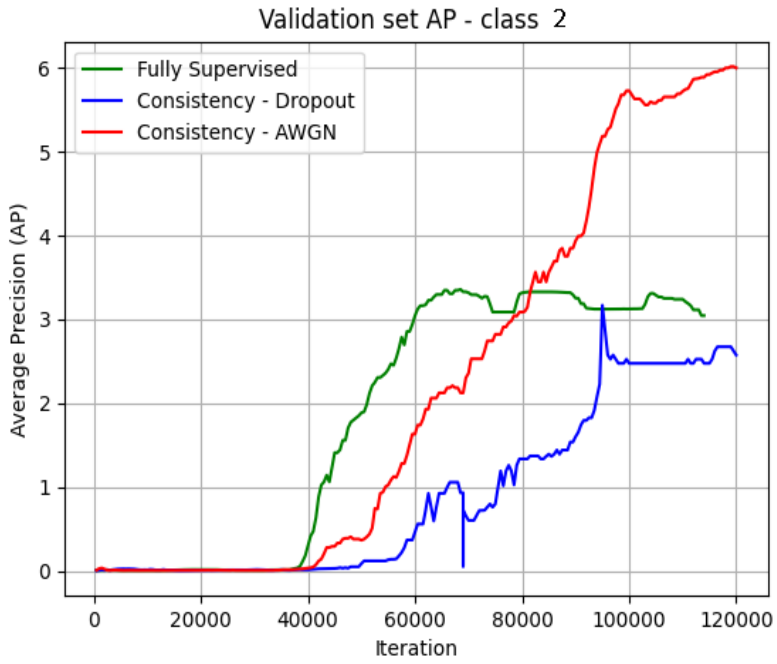

Figure 41: AP in the validation set (class 2).

4.2.2.5 Effect of consistency classification loss

We study the impact of the consistency classification loss. We decided to test the Mean Squared Error (MSE), as Tarvainen et al. suggest that it conducts to better performances (Antti Tarvainen, 2018), and compare its results when using the Kullback-Leibler (KL) divergence, as in Equation 12. Figure 42 and Figure 43 show the per-class AP in the validation set when using these two losses, and with flips as feature maps transformation. We observe that, even though the KL divergence leads to better performances for Class 1 objects, the AP for Class 2 objects is increased by more than 2 points of AP when using MSE. However, more experiments should be conducted to determine how using the MSE affects the performance when using AWGN as feature maps transformations.

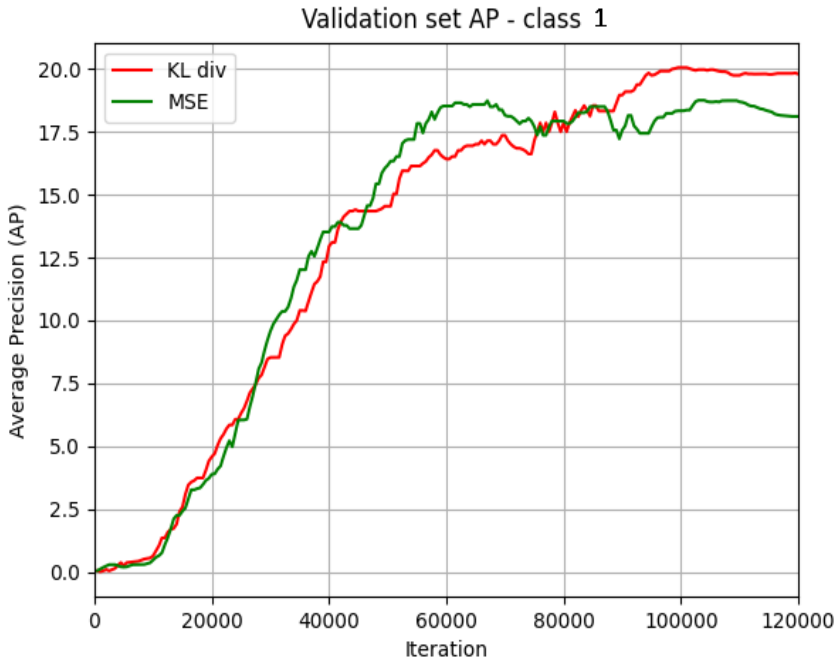

Figure 42: AP in the validation set, for different consistency classification losses (class 1).

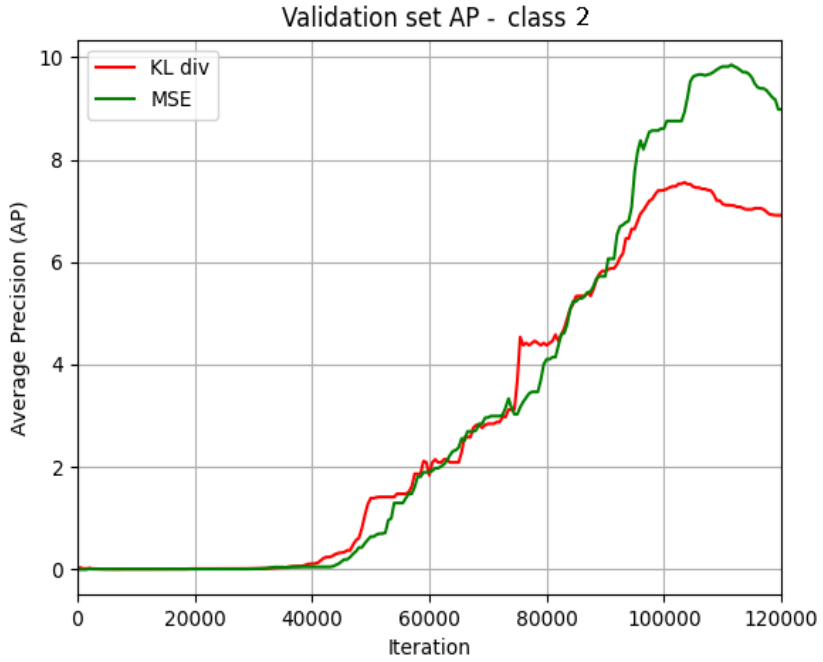

Figure 43: AP in the validation set, for different consistency classification losses (class 2).

4.2.2.6 Effect of consistency regression loss and combining transformations

In this subsection, we study the impact of taking the mean or average value in Equation 12 instead of the minimal value, and of combining different transformations (flips and AWGN). The results are shown in Figure 44 and Figure 45, where Flip H stands for only horizontal flips, Flip V for only vertical flips, and Flip H&V for vertical and horizontal flips at the same time (the flips that we used in the previous subsections). We can first see that the performance generally increases when using the mean value. The best result for Class 1 objects was obtained when using the mean in Equation 12 and when combining all the transformations. For Class 2 objects, the best result was obtained when using the mean in the consistency regression loss, and only AWGN as the feature maps transforms. However, more experiments should be conducted to determine the best combination of transformations, which will probably not be the same for the two classes.

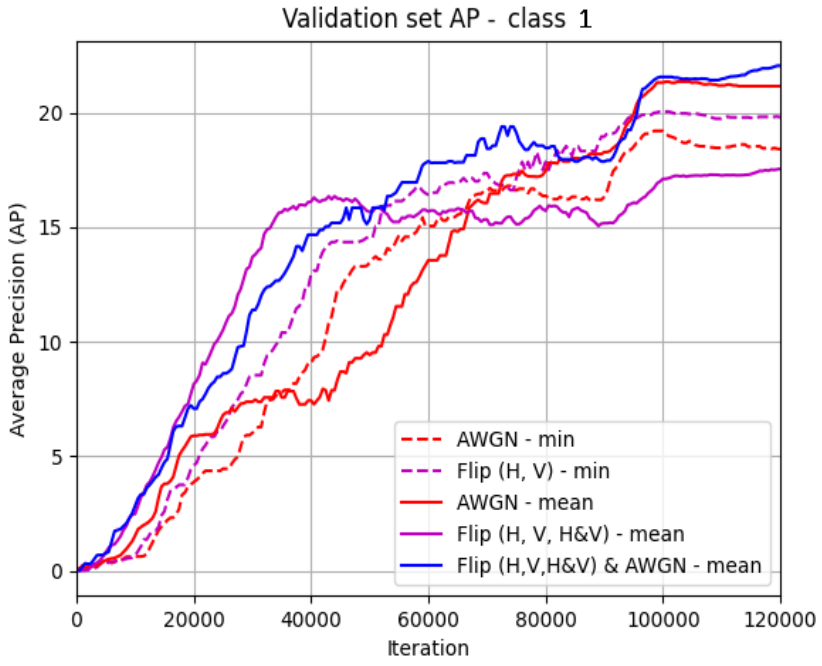

Figure 44: AP in the validation set for class 1, when varying the consistency regression function and combining transforms

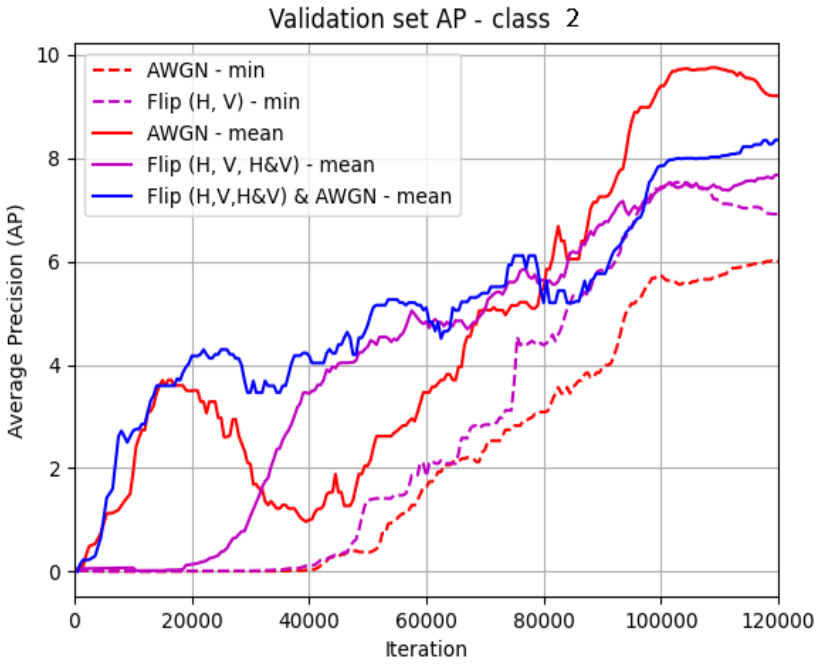

Figure 45: AP in the validation set for class 2, when varying the consistency regression function and combining transforms.

4.2.2.7 Mean Teacher-student model

To improve the performance, we implement a Mean Teacher-student (MST) model at the R-CNN level (Figure 46). Images are forwarded through a common RPN that outputs the feature maps of the selected regions of interest. The original versions of these feature maps are forwarded through the Student R-CNN network, and the transformed versions through the Teacher R-CNN network. Supervised losses are calculated with the outputs of the Student R-CNN; consistency losses are calculated between the outputs of the Student and the Teacher R-CNNs. The Student's weights are updated via backpropagation and those of the Teacher by an Exponential Moving Average (EMA) of the Student's weights. The reason to use this modified version of the classical Mean Teacher-student model introduced above is that we consider that the RPN gives good results, and want to have something directly comparable to the experiences above (in which we applied transformations to the feature maps and not to the images).

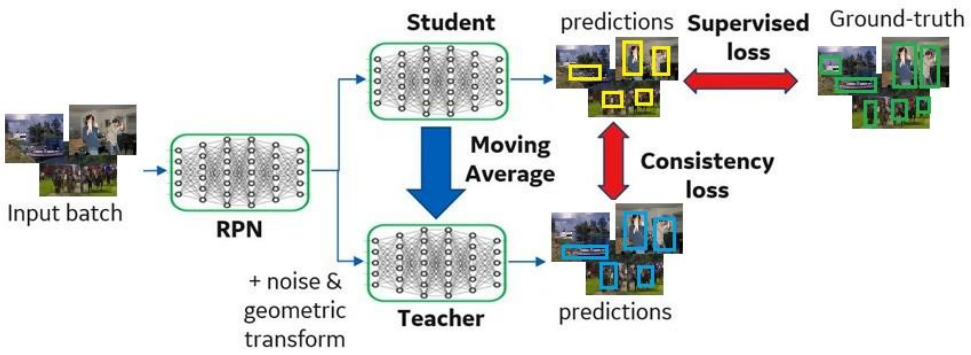

Figure 46: Mean Teacher-student model at the R-CNN level.

Figure 47 and Figure 48 show the results obtained using the Teacher-student model. We tested the MTS model using flips and AWGN (separately), and with the original consistency losses of Equation 12. As we noticed that the training under the MTS model was quicker, we were able to decrease the Ramp Up period from 60k to 20k and to obtain better results. We first remark that the Teacher outperforms the Student in all cases, as found by Tarvainen et al. (Antti Tarvainen, 2018), and that the MTS outperforms regular Consistency Learning, under its original configuration. For Class 1, the best model is the one using AWGN and for Class 2 the one using flips, both with a Ramp Up period of 20k.

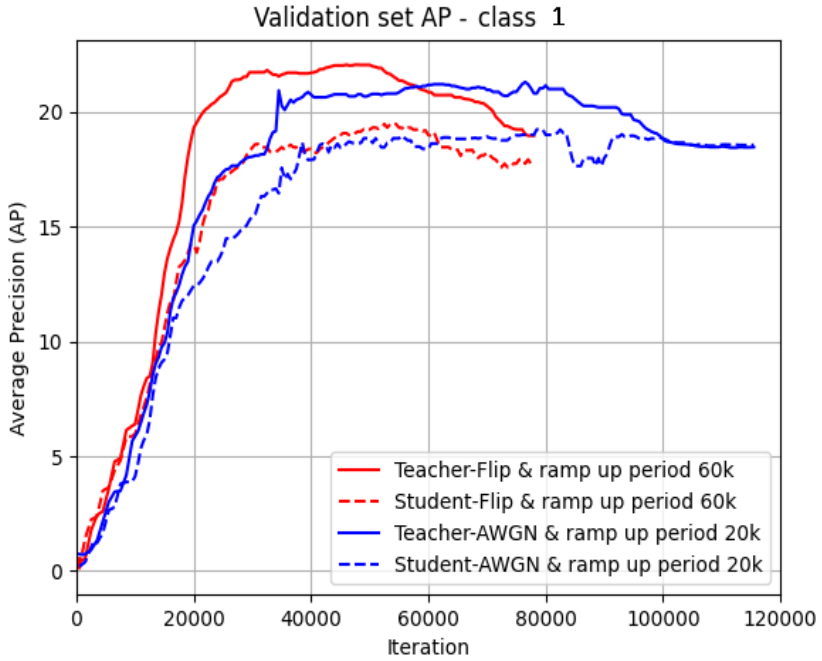

Figure 47: AP in the validation set for the Teacher-student model, class 1.

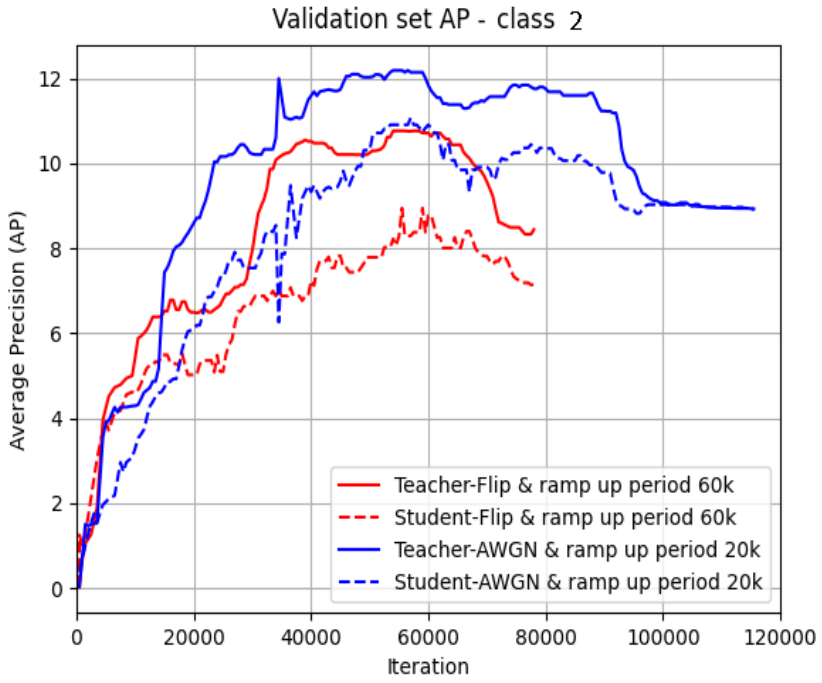

Figure 48: AP in the validation set for the Teacher-student model, class 2.

4.2.2.8 Performance comparisons

In Table 6, we sump up the main results obtained in the previous experiments. They have been divided in four categories: Fully Supervised Learning, Consistency Learning with the original configuration (where the consistency losses are those of Equation 12), Consistency Learning with modified consistency losses and feature maps transforms combinations (as in sections 4.2.2.5 and 4.2.2.6), and Mean Teacher-student model.

		Class 1	Class 2
Fully Supervised Learning	FSL	16.3	3.4
	FSL + Data Augmentation (Flips H&V)	14.5	3.8
Consistency Learning (original configuration)	CL – Flips (H&V)	20.0	7.5
	CL - Dropout	18.1	2.5
	CL -AWGN	18.3	6.0
Consistency Learning (Consistency loss modifications & features map transforms combinations)	CL – Flip - MSE	18.1	9.8
	CL – Flips (H&V) & AWGN - mean	**23.0**	8.3

	CL –AWGN - mean	22.5	9.6
Mean Teacher-student	MTS - Flip	**23.0**	10.8
	MTS - AWGN	21.0	**12.2**

Table 6: per-class Average Precision obtained in the validation set in the most relevant experiments.

Table 6 shows that, when used with appropriate transformations, Consistency Learning increases the performance of the detector for the two classes (increase of AP in 3.7 points for both classes, with the original configuration). We see that the performance is generally higher for Class 1 objects (max. is 23.0 AP) than for Class 2 objects (max is 12.2 AP), which is mostly a consequence of semantic differences between the classes that make Class 2 more difficult to detect. However, it can also be a consequence of the fact that the objects of Class 1 are, in average, bigger than those of Class 2 (see Figure 22 and Figure 23) and, therefore, easier to detect. We can also see that we manage to increase the performance by modifying the consistency losses (using MSE in the classification loss and the mean in the regression loss) and combining transformations. Using a Mean Teacher-student model also increases the performance, as predicted by the theory. One interesting remark is that the models that maximize the AP are different for Class 1 and Class 2 objects. While for Class 2 the AP is maximized by using the Teacher-student model with AWGN, for Class 1 the best models are the Teacher-student with flips, and regular Consistency Learning using AWGN and flips. In both cases, the increase in the performance with respect to Fully Supervised Learning is remarkable: 6.7 points of AP for Class 1 and 8.4 for Class 2.

5 Conclusions & future directions

A Deep Learning-based object detector for images was designed and implemented. After a careful literature review in object detection by Deep Learning and some initial trails, we decided to base it on Faster R-CNN, the current state-of-the-art in two stages detectors. The detector was trained with an annotated dataset of 1000 images, which is quite small. Validation was performed with a subset of the labeled dataset that contains images that are not in the train set, in order to avoid biasing the detector. A test set of 115 other images was put aside for final model validation. An unlabeled database of 8470 images was incorporated to training by using Semi-supervised learning techniques, in particular Consistency Learning. Consistency Learning, when used with feature maps flips and AWGN, increase the performance of the detector by 3.7 points of AP for Class 1 and Class 2 objects. In addition, the performance can be increased even more by modifying the consistency loss, combining features map transformations, or using the Mean Teacher-student model. The Semi-supervised-trained detector achieves a maximum performance of 23.0 and 12.2 AP points for detecting Class 1 and Class 2 objects, respectively. This implies a remarkable increase of 6.7 and 8.4 AP points, with respect to the detector trained using FSL.

The detector has, though, some limitations. It is unable to detect objects of Class 3, which is principally due to the low representativity of this class in the dataset. Besides, even though it attains 23 points of Average Precision for Class 1 objects, the detection performance for Class 2 objects is not very high and should definitely be increased. The fact that the detector is optimized differently for each class (different feature maps transformations), suggests that using class-specific detectors could enhance the performances. That is, training three binary detectors specialized in detecting each of the classes, and running them in parallel to perform the multi-object detection in an image. These detectors could have shared blocks, like the Backbone, to reduce training and

inference time. It would also be interesting to incorporate the Focal loss of RetinaNet into our Faster R-CNN-based detector, to better balance the impact of underrepresented classes and of hard examples in the loss function. In addition, Feature Pyramid Networks could be incorporated into the Backbone, for improving multi-scale detection. Finally, the detection performance in all classes could be enhanced by using Self-supervised Learning for pre-training the detector's weights. As explained in section 2.3.1, Self-supervised Learning would use a part of the detector (for instance, the Backbone) to solve a pretext task that is completely different from its original task. By solving this auxiliary task, the network gains semantic knowledge that enables it to better fulfill the detection task.

References

Abdul Vahab, M. S. (2019). Applications of Object Detection System. *International Research Journal of Engineering and Technology (IRJET)*.

Aiham Taleb, C. L. (2019). Multimodal Self-Supervised Learning for Medical Image Analysis.

André Araujo, W. N. (2019). Computing Receptive Fields of Convolutional Neural Networks.

Antti Rasmus, H. V. (2015). Semi-Supervised Learning with Ladder Networks.

Antti Tarvainen, H. V. (2018). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results.

Ashia C. Wilson, R. R. (2018). The Marginal Value of Adaptive Gradient Methods in Machine Learning.

Bernhard E Boser, I. G. (1996). A Training Algorithm for Optimal Margin Classifier.

Chieh-Chi Kao, T.-Y. L.-Y. (2018). Localization-Aware Active Learning for Object Detection.

Gao Huang, Z. L. (2018). Densely Connected Convolutional Networks.

Girshick, R. (2015). Fast R-CNN.

Golnaz Ghiasi, T.-Y. L. (2018). DropBlock: A regularization method for convolutional networks.

Herbert Bay, T. T. (2006). SURF: Speeded Up Robust Features. *European Conference on Computer Vision*.

Jifeng Dai, K. H. (2015). Instance-aware Semantic Segmentation via Multi-task Network Cascades. *arXiv*.

Jisoo Jeong, S. L. (2019). Consistency-based Semi-supervised Learning for Object Detection. *NeurIPS 2019.* Vancouver, Canada.

Jonathan Tompson, R. G. (2015). Efficient Object Localization Using Convolutional Networks.

Joseph Redmon, S. D. (2016). You Only Look Once: Unified, Real-Time Object Detection.

Kaiming He, G. G. (2018). Mask R-CNN. *arXiv*.

Kaiming He, X. Z. (2015). Deep Residual Learning for Image Recognition. *arXiv*.

Keze Wang, D. Z. (2017). Cost-Effective Active Learning for Deep Image Classification.

Keze Wang, X. Y. (2018). Towards Human-Machine Cooperation: Self-supervised Sample Mining for Object Detection.

Lee, A. (s.f.). Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics. *Swarthmore College*.

Liang Chen, P. B. (2019). Self-supervised learning for medical image analysis using image context restoration.

Lin TY, M. M. (2015). Microsoft COCO: Common Objects in Context.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*.

Michael Calonder, V. L. (2010). BRIEF: Binary Robust Independent Elementary Features. *11th European Conference on Computer Vision.*

Niall O'Mahony, S. C. (s.f.). Deep Learning vs. Traditional Computer Vision.

Ozan Sener, S. S. (2017). Active Learning for Convolutional Neural Networks: A Core-Set Approach.

Pedro Felzenszwalb, R. G. (2013). Visual Object Detection with Deformable Part Models.

Peng Tang, C. R. (2020). Proposal Learning for Semi-Supervised Object Detection. *arXiv*.

Phill Kyu Rhee, E. E. (2017). Active and semi-supervised learning for object detection with imperfect data.

Ren S, H. K. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

Ross Girshick, J. D. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation.

Samuli Laine, T. A. (2017). Temporal ensembling for semi-supervised learning. *ICLR 2017*.

Ting Chen, S. K. (2020). A Simple Framework for Contrastive Learning of Visual Representations.

Tsung-Yi Lin, P. D. (2017). Feature Pyramid Networks for Object Detection.

Tsung-Yi Lin, P. G. (2018). Focal Loss for Dense Object Detection.

Wei Liu, D. A.-Y. (2016). SSD: Single Shot MultiBox Detector.

Yoav Freund, R. E. (1996). Experiments with a New Boosting Algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference.*

Yuxin Wu, K. H. (2018). Group Normalization.

Zhong-Qiu Zhao, P. Z.-t. (2019). *Object Detection with Deep Learning: A Review.*

Zhong-Qiu Zhao, P. Z.-t. (2019). Object Detection with Deep Learning: A Review. *arXiv*.

Zhou, Z.-H. (2018). A Brief Introduction to Weakly Supervised Learning.