Politecnico di Torino

DIPARTIMENTO DI AUTOMATICA E INFORMATICA
Master Degree in Computer Engineering

MASTER THESIS

Visual object detection across different domains by solving self supervised tasks

Supervisors:
prof. Barbara Caputo
prof. Tatiana Tommasi
doct. Antonio D'Innocente

Francesco Cappio Borlino

December 2019

To my family

Abstract

Deep Neural Network models based on convolutions need a large dataset to be trained successfully. In the Computer Vision context this implies that a lot of labeled images have to be collected in order to obtain a model with good performances. A trained model is then often unusable when exploited in a visual domain which is different from the training one; moreover the data collection and labeling process can be physically or economically impossible in some visual domains. From these considerations the need to develop algorithms robust to visual domain shifts.

Self supervised tasks have shown a great potential as a strategy to learn useful features from unlabeled images. They can therefore be used in cross domain analysis as a method to obtain feature alignment between different domains. The purpose of this thesis is to study how self supervised tasks can be used to develop well performing visual object detection models in various cross domain analysis settings: Domain Generalization, Domain Adaptation and a new more general and challenging setting called One-Sample Adaptation.

Contents

Introduction

The purpose of my thesis project is to research and document the possibilities for the employment of a self supervised task as a method to obtain better cross-domain performances on the visual object detection task.

Deep neural networks based on convolutions have, in the last years, *revolutionized* the machine learning world, especially in the context of computer vision. With the use of a deep model it is possible to automatically learn high quality semantic features from large sets of images: these features can then be used to perform various tasks, often with results which are highly superior than those obtained with *shallow* methods.

One of the most important problems that the research in this area has now to deal with is represented by the use of deep models in cross-domain settings. In fact it has been shown that models trained with a dataset collected in a certain visual domain do not perform well in another one. This is a consequence of the fact that the features learned by the models are domain-specific and so suitable only to describe images belonging to the training domain (often called *source* domain). The cross domain analysis is particularly important as in some situations it is not economically feasible, or even not possible at all, to collect a large labeled dataset belonging to the test domain (also called *target*) in order to train a dedicated model. Examples:

- sometimes the labeling process is really expensive. In fact, while we can find trivial the labeling needed to train a visual classifier, some other tasks like *visual object detection*, *instance segmentation* or *semantic segmentation* require more complex labels that are more difficult to produce. In this case the cheapest solution is to use, even for the *target* domain, a model trained in a different domain, for which a large labeled dataset is already available;

- sometimes the *target* domain is not known *a priori*. In this case it is important to have a domain-invariant model which is able to generalize to any possible target;

- for really complex tasks, especially if the applications have critical safety levels, the training requires very large datasets. It may then be a good idea to rely on synthetic datasets that better cover all the possible situations the model could deal with. For this reason it is fundamental to be able to adapt a model trained on synthetic images to obtain good performances also with real world images.

Between the most used strategies implemented in the cross domain analysis there are approaches designed to obtain domain-invariant features or a pixel-level or feature-level alignment between the source and the target domains. These methods are often based on adversarial learning techniques.
Recently the so called *self supervised tasks* have gained momentum as methods to learn useful representations from images without the need for labels. Some works have shown that it is

possible to exploit this ability to obtain feature-alignment between different visual domains, with comparable, or even greater, cross-domain performances than those obtained with previous methods, even the adversarial ones. This approach is still unexplored in the context of visual object detection, which is a more complex task w.r.t. the more traditional image classification, as it requires to build algorithms which are able not only to recognize in an image all the objects belonging to known categories, but also to point out where they are using bounding boxes.
The main contributions of this thesis are:

- the presentation of a strategy to integrate a self supervised task into two state of the art visual object detection algorithms: Single Shot MultiBox Detector (SSD) and Faster R-CNN;
- the description of a method to train these models simultaneously on the main task (visual object detection) and the self supervised task;
- the explanation of the approach to be followed to use this type of framework in order to obtain good results in 3 different cross domain analysis settings:
  - **Domain Generalization**: in these settings the target domain is not known a priori and therefore a model is trained on various source domains in order to learn domain-invariant features that enable the generalization ability;
  - **Domain Adaptation**: in these settings the target domain is known a priori and some unlabeled samples belonging to it are available to obtain pixel level or feature level alignment between the source and the target domain;
  - **One-Sample Adaptation**. These settings represent a novel cross domain analysis problem: the target domain is not known a priori, but the model can use the samples it receives for the inference as source of information to adapt itself.

Chapter 1

Problem definition and related works

1.1 Definition of tasks and settings

1.1.1 Object Detection

Object detection is one of the most classic tasks of computer vision. The idea is really simple: to produce a software which is able to analyze digital images and not only to recognize the represented objects, but even to point out where they are.

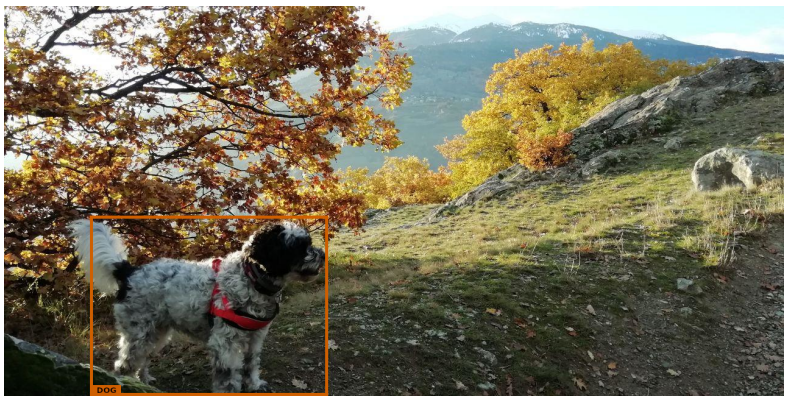


Figure 1.1: Example of output produced by a visual object detector on an image.

A good object detector is the basis of many different real world applications, going from face recognition in digital cameras to obstacle classification for self driving vehicles. The object detection task can, in turn, be considered as the union of 2 different more general tasks:

- the **detection** of objects (and their boundaries) in an image;
- the **classification** of objects into a set of known categories.

Image classification is perhaps the most iconic between the tasks in computer vision. Sometimes referred as *object recognition*, it requires the system to be able to recognize objects in images

when they belong to one of the known categories. A real world application example is the recognition of handwriting. It is clear that a good object detector needs to integrate a good classifier.

Related works

Traditionally computer vision tasks like classification and object detection were addressed using a **shallow approach** based on *hand-designed* features and machine learning algorithms like the Support Vector Machines. More recently *deep learning* methods based on *Convolutional Neural Networks* (CNNs) have been largely used. These methods have substituted the shallow approach because they do not only allow to obtain better performances but they can also be trained end-to-end: this means that they learn everything from data and, as a consequence, they do not require hand-designed features.

The exceptional improvement in performance obtained with the introduction of deep approaches is witnessed by the improvement in the results on the famous *Imagenet Large Scale Visual Recognition Challenge* [1] (see figure 1.2).

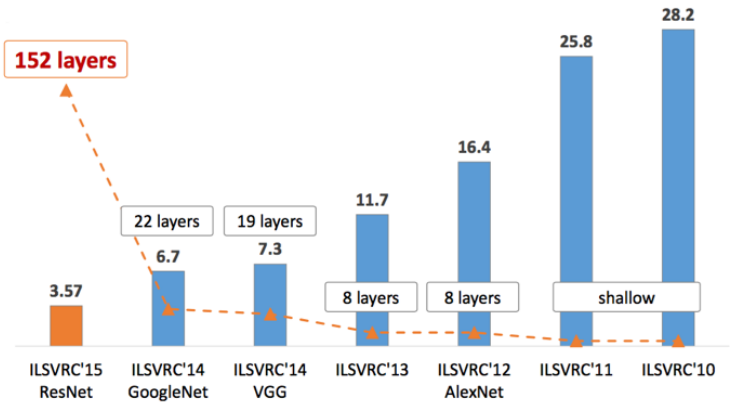


Figure 1.2: Representation of the top-5 error for the winning entries in the ImageNet Large Scale Visual Recognition Challenge [1]. There is a clear difference in performance of deep methods w.r.t. shallow ones.

Given the improvements obtained in the image classification task using deep approaches, many researches have been carried out aiming at using deep methods even for the object detection task. In general, a naive approach to this task consists of a 2-stage procedure:

- first it is necessary to extract the regions that may contain objects;
- then the extracted regions have to be classified as if they where independent images, but adding the class *background* to the list of known classes.

This kind of approach allows to easily integrate in the pipeline a deep model for the classification step. This is exactly what has been done by Girshick *et al.* in [2] with their R-CNN model. In order to increase the accuracy, R-CNN uses a *region proposal* algorithm (Selective Search [3]) and then it sends each region to a deep CNN to extract features used as input for a SVM classifier. In addition to the increase in performance, the use of a deep classifier within an object detector allows to exploit a large labeled dataset, designed for the classification task, in order to improve

the results obtained in the object detection task for which the datasets are usually smaller.
In 2013, the R-CNN approach had a great success, outperforming previous state-of-the-art algorithms in the famous PASCAL VOC 2012 challenge with an improvement in *mean Average Precision* higher than 30% ([2]). Nevertheless this approach was also characterized by a great disadvantage for an object detection model: it was slow in the inference process and, as a result, it could not be used in real time systems. Real time detection ability is critical in some object detection applications, for example in obstacle detection for self driving vehicles. This is why research has focused on improving the speed of the algorithms, not only their accuracy. The current state-of-the-art algorithms for object detection can be grouped in 2 families:

- **R-CNN family**. With algorithm like *Faster R-CNN* ([4]) and *Mask R-CNN* ([5]) that are both faster and more precise than the original R-CNN on which they are based;
- **Single shot methods** like *SSD* ([6]) and *YOLO* ([7]) that are more focused on real-time detection and do not use a 2-stage region proposal plus classification pipeline, but produce regions and class predictions together.

1.1.2 Cross domain analysis

The use of machine learning methods for computer vision tasks is heavily based on data. This holds even more in the particular case of deep learning methods that are trained end-to-end. In fact, trained algorithms based on deep architectures often model knowledge that was exclusively extracted from a dataset. However, in the case of computer vision tasks, the dataset is only a *representation* of the world. In this context, if a dataset is biased, then a model built on it will be equally biased and so essentially useless in domains which are different from the one represented by the original dataset.

In a 2011 study [8] Torralba and Efros showed that many of the popular datasets, even those that have been designed to capture the complexity of the real world like ImageNet [1], feature a strong bias. The clear consequence is that models trained on a single dataset show a deterioration of the performance when used in test settings corresponding to a domain different from the training one. These considerations are the basis for the researches in the so called *Domain Adaptation* (DA) and *Domain Generalization* (DG) settings.

The purpose of **DA** is to be able to build a model that can be used successfully in a specific target domain that is different from the source one. This problem has been studied from various points of view in order to be able to deal with all the possible situations. All these settings suppose that the target domain is known a priori. Examples of different settings are:

- **Unsupervised Domain Adaptation**. In this case a quite large set of images belonging to the target domain is available. These images are not labeled (thus the name *unsupervised*) but they can nevertheless be used at training time as part of the adaptation procedure;
- **Online Domain Adaptation**. In this case the model must be able to adapt to the target domain in a progressive way because it only receives small batches of images one at a time.

Differently from DA, in the **DG** settings the target domain is not known *a priori*. The purpose is therefore to produce models that are domain-invariant and thatcan then be used in any possible target domain without a deterioration of the performances. This result is usually obtained considering, at training time, a set of different domains. A model able to generalize should in fact be trained on more than a single domain in order to be able to understand which features are domain-specific and which are domain-invariant. In this way a good model can learn to apply to previously unseen domains the knowledge it has learned from known ones.

Li *et al.* in [9] showed that, in order to better estimate the generalization or adaptation ability of a model, it is not enough to consider only datasets representing the real world (that is: containing only photos), as in this case even deep classifiers simply trained on the union of the datasets represent a strong baseline. On the contrary it is important to consider also highly different domains like comics and paintings. Indeed the human mind is able to recognize a car whether it is photographed or sketched so why should we not expect the same from our algorithm? This is the reason why Li *et al.* presented in [9] a new DG benchmark dataset focusing on the domains: Photo, Art painting, Comic and Sketch (PACS). Because this dataset is designed for *image classification* it is not the one on which this Master thesis work will be focused on, but it is the dataset from which we take inspiration to build our research settings.

(a) Real world image sample [10]

(b) Watercolor image sample [11]

Figure 1.3: Comparison between a photo and a watercolor picture. There is clearly a large domain shift between the two images.

Related works

The majority of the works focused on addressing the problem of domain-shift between training set and test set fall into the Domain Adaptation settings. These settings are clearly less general then the Domain Generalization ones, but they are still relevant because domain adaptation methods usually allow to obtain better results when the target domain is known a priori.

An example of a real world situation in which it is fundamental to have a good domain adaptation strategy is whenever you want to train a model with synthetic data [12]. For example in the case of self driving cars it is fundamental to have a very large training set spanning all possible weather and lighting conditions with different driving environments. As there images should also be labeled, the production of a dataset of this kind can be really expensive both in terms of time and money. For this reason tools that are able to generate synthetic images and automatically label them are often used. The training is then performed using generated images and in a second moment the model is adapted to real world images using a smaller real world dataset.

Recent DA studies usually try to bridge the domain shift between the target and the source domain. This is often obtained exploiting one of two types of distribution alignment:

- **feature alignment**. The model is pushed to learn features that are domain-invariant. In this way the learned representation can match both the target domain and the source one. Various methods reach this objective by using an adversarial training approach [13] [14].

In practice a domain classifier is used together with an adversarial loss to force the model to learn features that cannot be used to discriminate between the domains;

- **pixel alignment** (or *image alignment*). Transformations are applied on the images of the source domain to make them look like images of the target one. In this way it is possible to train the model on labeled instances that visually belong to the target domain [11]. Some of this methods exploit generative networks like CycleGANs [15] to translate the images.

In general GANs [16] have been exploited in quite a lot of different ways as components of methods developed for the DA settings.

In all the situations in which the target domain is not known a priori all the DA algorithms are useless, and DG methods should instead be chosen. Usually DG studies are based on a small set of different intuitions that justify the used strategies:

- the possibility to project all the data on a new domain-invariant space where the training can take place and that should be suitable also to model new previously-unknown domains [17], [18]. This approach can also be described as an attempt to manipulate the statistics of each domain to make them comparable;

- the possibility to measure the similarity between a sample and the known domains in order to choose the domain that is more suitable to represent that sample and then use a model trained on that domain to perform the inference [19];

- the possibility to generate a domain-agnostic model supposing that each domain-specific model can be split into a domain-specific and a domain-agnostic part [9].

In this research I will present an approach that can be used both in DA and in DG settings and that is different from all the previously used methods as it bases its cross-domain generalization ability on the use of a self supervised task. In particular I followed the procedure of [20], with slight modifications, to train a model which is able to generalize thanks to a strategy of joint supervised and unsupervised (self supervised) learning, focusing on the object detection task.

I will also present a novel research setting that belongs to the cross-domain analysis context and in particular falls between DG and DA but can be seen as more general than both of them: **One-Sample Adaptation**. The rationale behind the need for this new research setting is that sometimes DA is not an exploitable option while DG does not make use of all the available information to perform the required task. In fact when the target domain is not known *a priori* DA can not be used, while regular DG methods try to perform an inference on the samples they receive without exploiting the sample's content to adapt themselves. Citing a definition presented by Li *et al.* in [21]: *In contrast to DA, a DG model is not updated after training, and the issue is how well it works out of the box in a new domain.* This impossibility to adapt is clearly a limitation. Nevertheless this limitation falls if you think that a model cannot adapt to an unlabeled sample, though this is exactly what a self-supervised task enables a model to do.

1.1.3 Self supervised learning

Traditionally, unsupervised learning methods were used mainly for exploratory analysis and dimensionality reduction, while supervised learning methods were primarily used to learn some sort of mapping between an input and an output. As a consequence, there was a clear separation between the applications of unsupervised and supervised techniques. This kind of isolation does not bring any advantage and it can even be seen as *unnatural* if we consider that the human learning approach is based on unsupervised filling of the knowledge acquired thanks to

a supervision [20]. Furthermore the *supervision* in machine learning is obtained usually through data labeling, a process that has two main problems:

- it is **expensive**. The labeling cost of a single data element depends on the task (for example it is lower in the case of classification and higher in the case of object detection, as more data need to be provided), but it is never for free and it is fundamental to remember that it is usually necessary to label a lot of data to build a dataset in order to use it to train a model;
- it is **error-prone** as it is performed by humans who can make mistakes. Any incorrect information in the labeling is assumed as *ground truth* by the model during the training.

These problems make us think that, if it was possible to learn our input-output mapping directly from unlabeled data, the advantages could be considerable. In the past this seemed impossible because, without a supervision, it was not even clear what a model should learn from the data. Recently some progress has been made: the *self supervised learning* expression has been introduced [22] to define a particular paradigm for unsupervised learning that aims at using information contained in the image to produce a supervision-signal. In practice the learning task is addressed like a supervised learning problem, but the supervision signal is not provided by a human, but directly and automatically extracted from the data. For example Doersch *et al.* in [22] extracted patches from images and trained a model to classify the relative position of 2 patches. Other strategies used are:

- rotation of images and training of a model to detect and classify the rotation [23];
- *desaturation* of images and training of a model to colorize them [24];
- split of an image in a grid of patches that are then shuffled in a problem similar to Jigsaw puzzles [25] [20].

Self supervised approaches have been studied in the context of *representation learning*. In this case the purpose of the model is to learn intermediate representations of data that can then be used to solve practical tasks. In this context a model is good if, once trained, it internally contains a good representation of the visual world and can therefore be used as a good starting point for a *fine tuning* on a more defined task, like *object classification*, exploiting *transfer learning* techniques.
The reason why self supervised tasks have been used as representation learning techniques is that they allow, in general, to extract, from data, information that can be useful to solve supervised learning problems, but that, is not sufficient by itself. Indeed, some aspects of knowledge, like the semantics, have to come from somewhere.

Related works

Self supervised learning techniques have been used mainly in two different ways:

- to learn a representation of the visual world which should be useful to initialize a model that is then trained using labeled data [22], [25], [23];
- to produce an auxiliary pretext task on which the model is trained simultaneously with the main task [20].

In particular Carlucci *et al.* in [20] exploited a self supervised task as an auxiliary task to improve the results of an *object classifier* in cross domain operation. The purpose of this master thesis project is to understand if it is possible to do the same with an *object detector* and to present the use of a self supervised task as a method to perform *One-Sample Adaptation*.

1.2 Formal problem formulation

The **Domain Generalization settings** in the context of object detection have not been really explored by many previous works so it is necessary to formally present the problem:

- we observe $\mathcal{S}$ domains;
- the i-th domain contains N_i labeled images;
- each label is composed by a set of *instance annotations*, each one containing a class and a bounding box specification;
- the purpose is to build an object detector that can be trained on $\mathcal{S} - 1$ domains and reach the best possible performance on the left out domain. This one represents a random sample for a new target domain containing the same set of categories of the training ones. As the target domain is a random sample, it represents an entire distribution of unknown domains. To ask the model to perform well on this target domain is the same as to ask it to perform well on any possible new domain;
- the object detector must output for each input image the class and the location of all the detected objects.

For what concerns the settings of **One-Sample Adaptation** the problem formulation is similar to the one of the Domain Generalization settings. The only differences are that:

- it is not necessary to have more than one source dataset;
- the detector at test time can use the sample it receives to adapt itself to the domain this sample belongs to.

In the case of **Domain Adaptation** the settings are a bit different:

- often there is only one source domain, but in some cases, when various supervised datasets are available, more domains are used;
- the target domain is known a priori;
- the produced detector should perform well (by outputting for each image the class and the location of all the detected objects) on data belonging to the target domain. It is not required for it to have good performance on the source or any other domain.

1.3 Presentation of domains and datasets

1.3.1 Pascal VOC and Artistic Media datasets

One of the most meaningful Domain Generalization benchmark is represented by the PACS dataset presented by Li *et al.* in [9]. In fact, many other popular benchmarks for cross-domain analysis represent only small domain shifts as they are usually based only on real world images for which the inter-domain differences are small. However, as pointed oud by Li *et al.*, the photos domain is potentially large enough to make useless the studies aimed at developing specific models that are invariant to the small differences between photos subdomains (like the use of different cameras). In practice it could be possible to collect a sufficiently large unbiased dataset of photos and train on it a deep model in order to obtain good performance on all kinds of photos. In other

words: studies on domain-invariant methods are only useful when the domain shift between the considered domains is not negligible and at the same time it is not possible or not feasible to collect a large dataset belonging to the target one. For this reason PACS (Photos, Art paintings, Cartoon and Sketch) takes into consideration domains for which the total available number of images is limited and that at the same time are clearly different.

Nevertheless this master thesis project is focused on the visual object detection task, for which PACS dataset cannot be used because it only has image level annotations while the task requires also instance level annotations:

- *image level annotations*: the labels provided for each image only describe which object (or objects) is contained in an image;
- *instance level annotations*: for each image the label lists all the objects contained even specifying their position through a bounding box or a more fine segmentation approach.

Even if the PACS benchmark is not useful in our research settings the reasons described for its importance are still relevant and, as a consequence, we try to build a new benchmark dataset with a purpose which is similar to the one at the basis of PACS, but specific for the object detection problem.
This result is obtained putting together in a single benchmark 4 different object detection datasets that share also the definition of the known categories. These datasets are:

- **PASCAL VOC 2007 and 2012 sets** [10]. This is a well known dataset for object detection covering the photos domain (sometimes called *natural image*). The data collected in 2007 and 2012 editions of this popular challenge span the same set of 20 classes and for this reason the two sets can be joined and used together;
- **Clipart1k**, **Comic2k** and **Watercolor2k** are datasets collected by Inoue et al. for their study in cross-domain weakly-supervised object detection [11]. These datasets have often been used in researches focused on Domain Adaptation for the context of object detection [26] [27] [28] and are often referred as *Artistic Media Datasets* (AMDs). The three datasets represent three different domains (described by their names) that satisfy the requirement described before of having a limited total available number of images. While Clipart1k span the same set of categories of VOC 2007 and 2012 datasets, Comic2k and Watercolor2k only contains images belonging to a subset of 6 of those classes.

To be able to work with these 4 datasets together, in the context of Domain Generalization, it was necessary to discard all the classes that are not common to all of them. For this reason in my DG experiments only 6 classes are taken into account, they are: bike, bird, car, cat, dog and person.
The **domain adaptation** settings, on the contrary, do not require to use more than 2 datasets at a time (the source and the target). All the experiments I performed in this setting by the described benchmark use PASCAL VOC datasets as *source* domain and one of the AMDs as target domain. In this case, as the source domain is the only one for which labels are used at training time, all the 20 object classes are kept.

1.3.2 Cityscapes and Foggy cityscapes

The *Cityscapes* [29] dataset has been presented in 2016 together with a benchmark, both focused on improving the researches in *visual scene understanding*. One of the most important strengths of this dataset is the attempt at capturing the variability and complexity of real-world inner-city

(a) Clipart1k's sample

(b) Watercolor2k's sample

(c) Comic2k's sample

(d) Pascal VOC 2012 training set's sample

Figure 1.4: Examples of images for the 4 different datasets

traffic scenes. This is done by capturing images in 50 different cities and by providing high-quality pixel level annotations for 5000 images. The dataset has been designed to be used in researches in the context of the autonomous driving vehicles and for this reason it is often used also for the visual object detection task ([30]).

The *Foggy Cityscapes* [31] dataset has been presented as a benchmark for *semantic foggy scene understanding* (SFSU). As fog represents a weather condition that can deeply influence the visual appearance of street scenes, for self driving vehicles it is fundamental to have good performances also in this situation in order to guarantee the safety of the passengers. This is the rationale behind the need for models that perform semantic foggy scene understanding, but as it is clearly not simple to collect a large dataset of foggy images having also a certain variability in the represented scenes Sakaridis *et al.* decided to design a pipeline to add synthetic fog on real images. As a consequence Foggy Cityscapes represent a dataset of images that are partially synthetic given that they are obtained by adding synthetic fog on the clear weather images of the Cityscapes dataset.

Together the Cityscapes and the Foggy Cityscapes datasets are often used as a benchmark for domain adaptation methods: a model trained on Cityscapes is adapted on Foggy Cityscapes to obtain good performance also in SFSU.

(a) Cityscapes' sample

(b) Foggy Cityscapes' sample

Figure 1.5: Comparison between Cityscapes and Foggy Cityscapes: example of addition of a synthetic layer of fog

Chapter 2

Object detection algorithms

In this section I will describe in more detail the theory behind the algorithms that I chose to use for my research. In general it is important to remember that visual object detection algorithms can be divided in two families:

- *2-stage* algorithms are often based on R-CNN [2]: they can be very accurate but they are also slow;
- *single stage* algorithms are faster but less accurate.

I decided to employ for my study an algorithm from both families to test the influence of the simultaneous supervised and self supervised training on both types.

2.1 SSD algorithm

The Single-Shot MultiBox Detector [6] is one of the most used state-of-the-art algorithms for the object detection task. It has been introduced by Liu *et al.* in 2016 [6], its main features are:

- **single-stage pipeline**. There is no need for a proposal generation step and for the consequent feature or pixel resampling as is the case of R-CNN-family algorithms. As a consequence the architecture is simple, trainable end-to-end and highly efficient;
- possibility of usage in real time applications. The original implementation in fact has a framerate of 59 fps on a Nvidia Titan X at test time;
- the accuracy is comparable with that of more complex approaches like Faster R-CNN.

The model has an architecture that can be roughly divided in 2 parts: the detector network, that is able to produce for each image a fixed-size set of detections represented by bounding boxes with an associated class probability, and a *Non-Maximum Suppression* (NMS) layer that allows to reduce the number of detections. Let us talk about these components in a bit more of details.

2.1.1 The detector

The network is built on a standard backbone that, in the original implementation, is a VGG16 [32]. This backbone, originally designed as a deep object classification network, is truncated before any classification layer and 4 extra convolutional layers are added to it. The purpose is to

exploit the natural ability of deep neural networks to learn a hierarchy of features by extracting feature maps at different layers in order to be able to **detect small objects using lower level features** (that have an higher resolution) and **big objects using higher level features** (see figure 2.1). For this reason 6 different convolutional layers are chosen, between those of the original backbone and the extra ones, to output features to be used for the detections.

This feature of SSD is, at least partially, the reason why this model outperforms other single-shot methods like YOLO (*You Only Look Once*) [7]. In fact, YOLO uses a single layer to output feature maps on which the detections are performed and so it has a naturally lower ability to deal with objects of different scales.

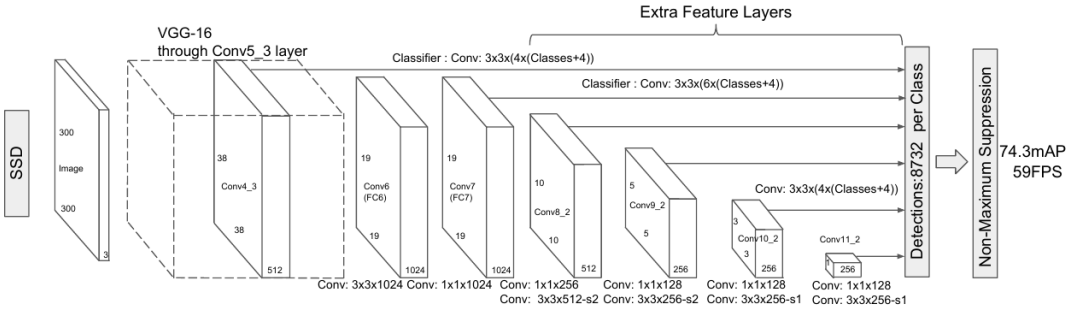


Figure 2.1: The original **VGG16 backbone** is truncated after the layer called "Conv5_3". **Extra layers are added** in order to downscale the produced features. Each chosen output has a resolution lower than the previous one, but also stronger associated semantics. Source: [6]

The SSD network bases its ability to detect objects of different scales and aspect ratios on the principle of default boxes (or prior boxes). This approach can be described in this way:

- each layer of the network that is used to produce an output generates a set of feature maps having a certain size. For example the first layer produces 512 feature maps of size 38×38, while the last layer produces 256 feature maps of size 1×1;
- for each feature map position a number of *default boxes* centered in it are defined. The various default boxes differ in terms of size and aspect ratio, the number of default boxes used is different for the feature maps extracted at the different layers;
- the overall number of default boxes generated for each image is constant;
- for each default box the network produces a prediction of probability for each known class and a bounding box offset prediction.

From a practical point of view the predictions are performed by the network using **classification and regression headers**. For each output of the backbone there are different headers, which get specialized in the detection of objects of different sizes and aspect ratios. In particular these headers are small 3x3 kernels with a stride equal to 1 and work in this way:

- each classification header is defined associating it to a particular default box size and aspect ratio for a particular output of the network. During the training it learns to detect one particular class of objects;
- 4 regression headers are defined together associating them to a particular default box size and aspect ratio for a particular output of the network. During the training these headers

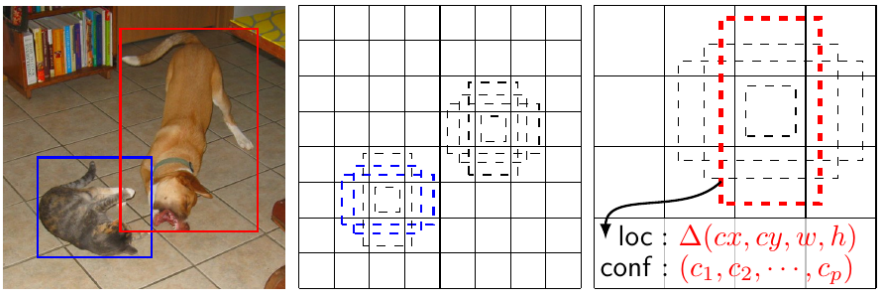


Figure 2.2: On the left an image with the original ground truth boxes. On the center and on the right 2 examples of feature maps with default boxes defined over them. Source: [6]

learn how to define an offset for the bounding box of an object that may be partially contained in that default box.

The regression headers are fundamental to produce higher quality bounding box positions: during the training they learn to specify, given a default box definition, an offset that, once applied to it, produces a bounding box that is more suitable to highlight the boundaries of any possible object that might be contained in the default box.

2.1.2 Non-Maximum Suppression

SSD, like many other detection algorithms, often produces multiple detections for each object. This happens because the network has to produce an high number of predictions (for SSD this number is 8732 per class in the original implementation) in order to be able to detect any number of objects of any possible type in each image.

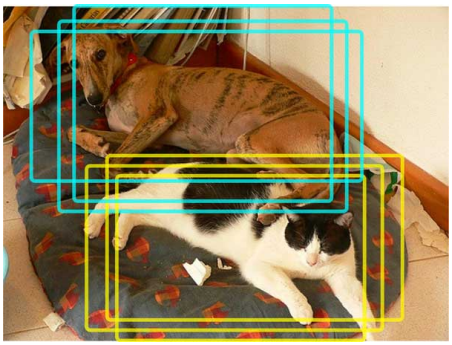


Figure 2.3: Example of situation in which Non-Maximum Suppression is needed. Source: machinethink.net/blog/object-detection

Even if this is the expected output for a network like SSD, it is not the kind of output which is desired at the end of the detection pipeline. To reduce the number of detections to the most reliable ones a layer called Non-Maximum Suppression is used. The idea is really simple:

- a *threshold* value is chosen for the measure of the IoU (see A.1.1 for a presentation of this metric) between different predictions;

- for each class the prediction with the highest associated probability is chosen to be inserted in the final list of detections. All the other predictions for the same class are discarded if their IoU w.r.t. the first is higher than the chosen threshold;

- this process is repeated until each prediction has been inserted in the final output list or discarded.

2.1.3 Training strategy

Matching default boxes with ground truth information

To train the detector it is necessary to define how ground truth information has to be associatedto the generated default boxes in order to **compute a detection loss and perform the backpropagation**. This association is in some terms necessary even when using other detectors, as Faster R-CNN, that has to associate ground truth information with the generated region proposals. The association is done taking into consideration the IoU of default boxes with ground truth boxes:

- a default box is marked as *positive*, and so associated with a ground truth box, if it is the one with the highest IoU with that ground truth bounding box or if its IoU is over a certain threshold (0.5);

- a default box is marked as *negative* in the other cases.

Following this association strategy there can be more than one default box associated to each ground truth box, this allows to have a higher number of positive samples w.r.t. other methods. Figure 2.2 shows an example of result of this matching strategy: a smaller object (a cat) is associated with 2 default boxes coming from a lower level feature map that has a higher resolution, while a bigger object (a dog) is matched with a default box of an higher level feature map with a lower resolution. The result of this training strategy is that the network learns to make feature maps which are responsive to particular scales of the objects. Once the matching has been performed, usually **there is a great imbalance between the number of negative samples and the one of positive samples**. This is because, if the number of default boxes is high, a lot of them pinpoint an area of the background and are, as a consequence, marked as negatives.

Hard Negative Mining

To perform the training via backpropagation it is a good idea to have a certain balance between positive and negative samples so that the network can learn to recognize both the objects and the background. For this reason it is necessary to choose a number of negative samples between the available ones. The strategy used to perform this choice is called ***Hard Negative Mining*** and it has been inherited by methods used before SSD. The idea is to choose the negative samples that can help the most the training of the network. These samples are the *hard negatives*, those with the highest associated loss between the available ones. To extract the hard negatives, the classification loss is computed for all the negative default boxes that are then sorted by the loss in descending order. The samples with the highest associated loss are *hard negatives* because the high loss means that the network believes that they are not background samples. The number of chosen hard negatives depends on the number of positive samples for each image: the objective is to get a ratio between negatives and positives less or equal than 3.

Loss function

The **objective loss function** used to perform the training is a weighted sum of a localization loss (*loc*), that measures how well the predicted boxes locations fit the ground truth boxes locations, and a confidence loss (*conf*) that measures how well the network predicts the class labels. We denote with $x_{ij}^p = \{1,0\}$ an indicator of the matching between the i-th default box with the j-th ground truth. It is clearly possible that $\sum_i x_{ij}^p \geq 1$. The overall loss function is:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \ ,$$

where:

- x is a network input (an image);
- N is the number of positive default boxes (if no positive boxes are available the computed loss is set to zero);
- c represents the ground truth labels;
- g represents the ground truth bounding boxes;
- the localization loss is based on the Smooth L1 loss that has been presented as part of the Fast R-CNN algorithm [33] (see paragraph 2.2.1). It compares the predicted boxes l with the ground truth boxes g for the positive samples:

  $$L_{loc}(x, l, g) = \sum_{i \in Positives}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m) \ .$$

  The ground truth box specification g is used with the default box description d in order to compute $\hat{g}$, i.e. the ground truth offset, that is then compared with the predicted one l;

- the confidence loss is the softmax loss (also known as *negative log likelihood* and *binary cross-entropy loss* in binary classification problems) over the confidences generated for the various classes (c). For each positive or negative prior box selected for the current image x we have, first of all, to apply softmax to compute the probability (or confidence) that the network assigns to each class for that prior box:

  $$\hat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \ .$$

  We then compute the confidence loss function using the negative log likelihood. Usually the formula of this function contains the one-hot GT label vector. In this case it is not necessary, as we suppose to consider directly only the correct class for each prior box:

  $$L_{conf}(x, c) = -\sum_{i \in Positives}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Negatives} log(\hat{c}_i^0) \ ;$$

- the term α allows us to regulate the importance of the two tasks (localization and classification) and is set to 1 by cross validation.

Data augmentation

To successfully train SSD to obtain a model that is robust, being invariant to object sizes, shapes, light conditions and so on, it is necessary to implement a Data Augmentation technique. This means that a series of transformations is defined in order to increase the overall number and variety of the images available for training. In particular the transformations used for this training are:

- random use of the entire image or of a random crop (maintaining a minimum overlap with the objects);
- random expansion of the image;
- resize to a fixed square (300×300);
- random horizontal flip;
- random photometric distortion;

The SSD model is naturally less good at dealing with small objects than for example Faster R-CNN. This is because the absence of a feature resampling step forces the model to use only low level feature maps to deal with small objects, and these feature maps have weaker associated semantics. The implemented data augmentation allows to improve the results in this ability to deal with small objects because random crops and random expansions perform respectively zoom in and zoom out operations that are fundamental to increase the number both of small and large objects in training samples. The random expansion operation was not even used in the original version of the SSD paper [6] and its introduction in a later version allowed to obtain a 2%-3% improvements in accuracy on multiple datasets.

2.2 Faster-RCNN algorithm

Faster R-CNN [4] is a state of the art algorithm for visual object detection whose main features are:

- use of a 2-step pipeline trainable with a single end-to-end procedure;
- state of the art performances in terms of detection accuracy;
- almost real time detection thanks to a frame rate of 5 fps including all steps.

The algorithm has been developed thanks to a series of sequential improvements performed on the original R-CNN ([2]), the first really successful object detection algorithm based on convolutional (deep) neural networks. For this reason, I will now first present the novelties that were introduced in this context by R-CNN and then the main improvements added to it, which allowed to work out the algorithm that I used for my research.

2.2.1 Faster R-CNN history and development

R-CNN

The results of the 2012 edition of the ImageNet Large Scale Visual Recognition Challenge [1] with the success of AlexNet [34] showed for the first time the great potential of Convolutional Neural Networks (CNNs) and deep models (see figure 1.2). This potential sparked a new wave

of interest in this kind of approach (CNNs have been studied at least from the '90s but at the beginning they were not so successful because the required computational power necessary to train them was not available at that time) and was the rationale behind the research of Girshick et al. that resulted in the presentation of the R-CNN algorithm [2]. The main novelties of this work are represented by the solution of two important problems:

- the successful integration of a CNN in a visual object detection pipeline;
- the training of a CNN with a small dataset (object detection datasets are usually much smaller than image classification datasets) thanks to a pretraining on an auxiliary dataset.

The R-CNN algorithm is based on the *recognition using regions* paradigm:

- a region proposal algorithm is used to produce, for each image, a list of regions (bounding boxes) that probably contains objects. These generated proposals must be *category-independent*. Any algorithm of this kind can be used, but for the original implementation Selective Search [3] by Uijlings *et al.* was chosen;
- each proposal is resized/warped to a fixed size and then passed through a CNN in order to produce feature embeddings for it. These features are then input in a number of SVMs (one for each output class) in order to produce class predictions.

Characteristics of this approach:

- it is highly efficient if compared to previous methods: in fact the most *computation-intensive* steps, that are the region proposal execution and the computation of feature vectors, are category-independent (and must then be executed only one time for all the categories). Moreover the computed features are low-dimensional (vectors of size 4096) so that also the subsequent computations, that are category-dependent, are at least less computation-expensive;
- the CNN used in the pipeline is pre-trained on the ILSVRC 2012 image recognition dataset, that has more than 1.2 million images, so that later it can be fine-tuned on the specific object detection task and obtain good results even when the available dataset is much smaller.

R-CNN allowed to improve results in terms of mAP (mean Average Precision) by more than 30% ([2]) w.r.t. previous approaches on the famous PASCAL VOC [10] object detection challenge. Some of the strategies implemented in this algorithm for the training, for example the *Hard Negative Mining*, although not new [35], are often still used in current implementations of state-of-the art algorithms.

Fast R-CNN

This algorithm [33], from the same author of R-CNN, represents a great improvement over it. As the name suggests its main purpose is to reduce the training and test time of the algorithm by improving its efficiency. In particular the main problems of R-CNN, which the introduction of Fast R-CNN solved, are:

- **No end-to-end training**. The training procedure of R-CNN is a complex multi-stage pipeline;

- **Training is long and space expensive**. The training procedure requires not only a lot of time because of the various steps that have to be followed, but also a lot of disk space as to train the SVMs it is necessary to compute and then to store on the disk the feature embeddings for all the region proposals of the images;
- **Slow test-time detection**.

The main improvements introduced by Fast R-CNN are:

- use of a single stage training pipeline, based on a multi-task loss to train simultaneously a classifier for the objects and a regressor to improve bounding box accuracy;
- no need to store features and consequently lower disk space usage;
- introduction of a *RoI pooling* layer that allows to share between all the region proposals of an image the computation of the features. In practice the whole image is passed through the R-CNN and then fixed-size feature vectors for the various proposed regions are extracted from the image's features;
- an efficient training procedure based on a new strategy for batch sampling.

Let us present some of these improvements in more detail.

RoI pooling It maybe represents the most important novelty of Fast R-CNN because it is the key element that allows to greatly improve the efficiency of the algorithm, in particular at test-time. The main bottleneck in R-CNN is in fact the necessity to pass through the network each proposed image region individually in order to compute a feature embedding for it. With Fast R-CNN this is not necessary anymore: to perform object detection on an image a single forward pass of the CNN is performed on the whole image. After this pass it is necessary to extract from the features computed for the image a fixed-size feature embedding for each proposed region. The *Region of Interest* (RoI) pooling layer allows to do exactly this: each RoI is divided in a fixed number of buckets and max pooling is then applied.

In reality RoI pooling was not the first attempt at sharing the computation of features between proposed regions because SPPnet algorithm [36] used a similar technique called *Spatial Pyramid Pooling layer*. However, as pointed out by Girshick in [33], SPPnet used an inefficient training technique that, together with the way the computation was shared, did not allow to update the weights of the convolutional layers preceding the SPP pooling layer during the training.

Single-stage training pipeline and multi-task loss Fast R-CNN discards the SVMs used by R-CNN for the classification step substituting them with two different *heads* for the network:

- a classification head is used to produce class predictions for each region proposal by using a softmax layer to output a probability distribution over the $K+1$ categories (the K classes of the used dataset and the background);
- a regression head is used to output for each class k an offset for the bounding box t^k.

In practice the network is trained to solve 2 tasks simultaneously: the classification of objects and the regression of bounding box proposals. This result is obtained thanks to a multi-task loss that takes this form:

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

This loss simply puts together two components:

- the classification loss $L_{cls}(p, u) = -log(p_u)$ is the classic negative log-likelihood for the correct class u, often used in classification networks implementing a softmax classifier;
- the hyper-parameter λ allows to regulate the relative importance of the 2 tasks;
- the localization loss, that thanks to the term $[u \geq 1]$ is considered only for foreground classes, is the so called smooth L1 loss:

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L_1}(t_i^u - v_i)$$

This is a robust L1 loss function that is less sensitive to outliers than the L2 loss and that allows to keep the loss limited when the regression targets are unbounded (to prevent the phenomenon of the *exploding gradients*). It is defined as:

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

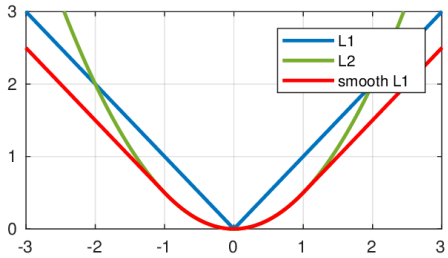


Figure 2.4: Representation of the trend of the smooth-L1 loss in comparison with standard L1 and L2 losses

Efficient batch sampling R-CNN and SPPnet are trained using mini batches containing a single RoI for each input image. For each RoI it is necessary to extract a fixed size feature embedding of a relatively small size while RoIs themselves can be potentially large so that their even bigger receptive field can span the whole image. The consequence for SPPnet of this small output plus large input approach is that the backpropagation does not allow to effectively update the weights of the network.

Fast R-CNN on the contrary uses a technique to produce mini batches based on the sampling of many RoIs from a single image. In practice if the batch size for the network should be 256 then:

- R-CNN and SPPnet would extract a single RoI from 256 different images;
- Fast-RCNN uses 128 RoIs extracted from only 2 images.

In this way it is possible to have a rather large output (the union of the feature embeddings of all the RoIs belonging to a single image) for each large input so that the backpropagation update is more effective.

2.2.2 Faster R-CNN

After the improvements introduced by Fast R-CNN, the main computational bottleneck of R-CNN methods has become the region proposal approach used. As pointed out by Ren et al. in [4], a typical CPU implementation of the Selective Search ([3]) algorithm requires 2 seconds per image to be executed. This is of course a limit as it does not allow to integrate this kind of object detection algorithm in a real time application. Obviously a GPU implementation of the same algorithm could help in saving some tenths of a second, but it may be a much better idea to design a completely new algorithm, which should be implementable in the GPU, and possibly integrable in a pipeline based on a deep network. An algorithm like this one can in fact share some computations with the network itself in order to further improve the efficiency of the method.

This is the main contribution of Faster R-CNN by Ren et al. [4]: the introduction of a *Region Proposal Network* that shares some convolutions with the object detection CNN in order to make negligible the cost for computing proposals.

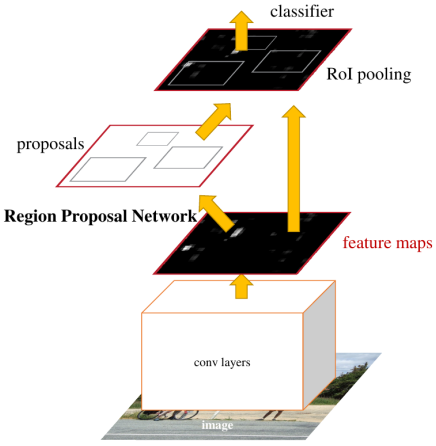


Figure 2.5: High level representation of Faster R-CNN's architecture [4].

Region Proposal Network

This network is not designed as an independent module to be added to an object detection pipeline, but with the aim to share all the *shareable* convolutional layers with the detector. In this way, at test time, the computations performed to generate the proposals are also part of the classification step.

The region proposals are generated following the idea of the anchor boxes:

- after the last shared convolutional layer the features are extracted and used as input for the RPN;
- a sliding window approach over the feature maps is used to extract *anchor boxes* from them. For each position of the window k different anchor boxes are generated considering different sizes and aspect ratios;
- all the anchor boxes are represented by features of the same size that are used as input to 2 sibling fully-connected layers: a box regression layer (*reg*) and a binary classification layer (*cls*);

- the binary classification layer is necessary to produce a prediction of *objectness* for each anchor box. In practice it has to predict if a certain region contains an object or not;
- the box regression layer has the purpose to obtain from the box position a more accurate bounding box for any object that could be inside of it.

To train the RPN a multi-task approach is used. First it is necessary to generate the targets for the task and this is done by sampling a fixed number of anchors from all the available one using this procedure:

- we call *positive* anchor boxes those having maximum Intersection over Union (IoU) with a ground truth (GT) bounding box or an IoU of at least 0.7;
- we call *negative* anchor boxes those having IoU < 0.3 w.r.t. any GT bounding box;
- we build a mini-batch of size 256 by choosing up to 128 positive samples and then padding with negative samples. This means that the ratio between positives and negatives can be at most 1.

At this point it is possible to train the RPN by computing for each sample in the batch:

- an *objectness* classification loss $L_{cls}(p_i, p_i^*)$ based on the classic negative log-likelihood loss (in this case over only two classes). We use p_i to represent the *objectness* score and p_i^* to indicate the Ground Truth that can only take 2 values: 1 if the anchor is positive and 0 if it is negative;
- a box regression loss $L_{reg}(t_i, t_i^*) = smooth_{L_1}(t_i - t_i^*)$, where t_i is a 4D vector containing the predicted coordinates and t_i^* is a 4D vector with the GT coordinates.

The computed losses are then aggregated into one:

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

where:

- N_{cls} and N_{reg} are normalization factors used to bring the 2 functions to a similar order of magnitude;
- λ is a weight used to regulate the relative importance of the 2 tasks;
- the regression loss is multiplied by p_i^* so that it is not considered in the case of negative anchors.

The described procedure for generating proposals has some major advantages over previously used methods:

- it is *translation-invariant*. A box produced for an object that is found in a certain position would be equally produced if the same object was in another position;
- it addresses automatically multiple scales and aspect ratios using images and features of a single scale (without requiring image or feature pyramids);
- it learns how to generate better proposals during the training. Other methods like SSD, on the contrary, generate region proposals in a static way.

Training strategy

To obtain a successful model it is necessary to train appropriately the 2 main modules of the network: the RPN and the detector. The two trainings can not be performed independently as they would influence the weights of the network in different ways. For this reason the available alternatives are:

- *step alternating training*:
  1. a network pre-trained on ImageNet is fine-tuned on the region proposal task;
  2. a Fast R-CNN model is trained using the proposals generated by the RPN trained above. At this point two models have been trained on the 2 different tasks;
  3. the detector layers are used to initialize a new RPN training for which all the layers that are not unique to the RPN are kept fixed. At this point a single model trained on both tasks is available;
  4. the detectors-unique layers are fine-tuned again by using the regions generated by the RPN while keeping fixed RPN-specific layers.
- *approximate joint training*. If a joint training is applied naively on the 2 tasks an approximation is applied. This is because if the region proposals produced for a certain iteration are considered as fixed proposals to train the detector than during the backpropagation we are forced to ignore the fact that the detection loss is influenced by the RPN loss.

A *non-approximate joint training* should be possible but it would require to use a RoI pooling layer that is differentiable also w.r.t. the box coordinates. This is a non trivial problem and for this reason it has not been studied yet.

From the point of view of the **loss function** Faster R-CNN uses the same multi-task loss implemented by Fast R-CNN adding the multi-task loss of the RPN. In practice the final loss (that takes exactly this form only in the case of joint training) is:

$$L(x, c^*, l^*) = L_{rpn,cls}(x, c^*) + L_{rpn,reg}(x, l^*) + L_{cls}(x, c^*) + L_{reg}(x, l^*)$$

Where:

- x is an image sample;
- c^* is the GT for the categories of the objects;
- l^* is the GT for the bounding boxes of the objects.

Chapter 3

Method

3.1 Self supervised auxiliary task

As stated in section 1.1.3, the purpose of this Master Thesis project is to understand if the exploitation of simultaneous training on a main task and on a self supervised auxiliary task as a strategy for cross domain-analysis presented in [20], can lead to improvements even in the object detection task.
To conduct a sufficiently comprehensive study 2 different self supervised tasks have been tested in conjunction with 2 different state of the art object detection algorithms: Single-Shot MultiBox Detector (SSD [6]) and Faster R-CNN ([4]).

I will now present in a more detailed way the 2 self supervised tasks and the reason why they have been chosen. It is important to keep in mind that the rationale behind the decision of using a self-supervised task is that this should push the model to learn semantic features which are useful for visual perception tasks. This means that we are not really interested in the solution of the self supervised task on its own. For this reason in this work the self supervised task is often referred as:

- *pretext* task: as even if a model is trained to solve this task we are not interested in using this ability at test time;
- *auxiliary* task: as there is always another task to be solved and the purpose of the self supervised task is to push the model to learn more, with the final aim of improving the performance on the main task.

3.1.1 Jigsaw task

The definition of the Jigsaw task follows, with small variations, what has been presented by Noroozi and Favaro in [25]. The main idea is the following:

- an image is split in a grid of $n \times n$ patches;
- a permutation of the tiles is applied;
- each one of the possible permutations receives an identifying index;
- of all the possible $n^2!$ permutations only P are kept in a list of possible permutations. Between the kept ones there is always also the permutation corresponding to the original image;

- the problem is defined as a classification problem: the model has to learn to predict to which of the possible permutations a sample image belongs to.

As the problem is defined as a classification problem, it can be easily solved by any existing image classification algorithm. The reason why the Jigsaw task has been chosen between the available options for self supervised tasks is that, as we can intuitively understand, it is useful to teach a model that each object is made of parts which follow a spatial correlation. This seems important in the case of the object detection task because it could improve the performance of the object classification step. In fact literature examples [20] [25] show the effectiveness of this approach. At the same time it is not clear a priori if the Jigsaw task can really help algorithms designed for visual object detection: in this case in fact the self supervised task is applied to images representing scenes and not single objects.

For the implementation of the Jigsaw task I followed the approach used by Carlucci *et al.* in [20]:

- each image is decomposed into a 3×3 grid;
- out of the 9! possible tiles permutations, only a subset of 30 is chosen using the Hamming distance.

3.1.2 Image rotation task

In this case the definition of the self supervised task follows the approach presented by Gidaris *et al.* in [23]:

- a random rotation of a multiple of 90° is applied to an input image;
- the rotated image is fed as input to the model;
- the model outputs a probability distribution over the possible rotations.

Once again the auxiliary task is defined as a classification problem so that any existing algorithm (and in particular deep neural network architectures) designed for the image classification task can be exploited to solve it.

The reason behind the choice of this self supervised task is that, as done in [23], we can argue that, in order for a model to be able to predict a rotation applied to an image, it is necessary for it to understand the concepts of the represented objects.

3.1.3 Simultaneous training

Many previous works ([25], [23], [22], [24]) have exploited a self supervised task in the context of unsupervised representation learning. In this case the objective is to learn from unlabeled data an inner representation of the visual world. In practice *representation learning* (or *feature learning*) is one of the key differences between a deep method and a shallow one: it represents the ability to learn automatically from data a set of features that characterizes visual objects so that it is not necessary anymore to rely on hand-designed features to build classifiers or detectors. In the particular case of *unsupervised* representation learning only unlabeled data are used, while the most traditional approach is the *supervised* representation learning that exploits labeled data. Unsupervised representation learning can be useful as an alternative to supervised representation learning as it represents a cheaper approach given that it does not require to label data.

Once a representation of the visual world has been learned the model can be used as a feature extractor for a classifier or simply to initialize the weights of another model that is then trained on a specific task (*transfer learning*).

This approach, however, involves 2 distinct training steps: one performed on unlabeled data using a self supervised task and one performed on labeled data to solve the required specific task. As pointed out by Carlucci *et al.* in [20] there is a contrast between this approach and the visual learning strategies of humans. Various studies ([37]) show that infants learn both to categorize objects and about visual regularities (objects parts and their relationships and also objects natural poses/rotations) at the same time. With the purpose of emulating these learning strategies Carlucci *et al.* presented [20] the first end-to-end architecture that learns simultaneously how to generalize across domains and about regularities of visual world.

This architecture has been reproduced with slight modifications that were necessary to simplify the application of the method and integrate it in object detection architectures. I will now describe in a high level way how the simultaneous training works. Later I will describe in detail how I implemented the self supervised branch into the algorithms then how I used this architecture in 3 cross-domain analysis settings.
High level description:

- the architecture implements two functions:
  - $h(x|\theta_f, \theta_d)$ is the *object detector* function, based on the parameters θ_f and θ_d. These parameters represent respectively the network nodes involved in the definition of the features and the ones involved in the classification/regression step.
  - $j(x|\theta_f, \theta_s)$ is the function that solves the self supervised task. It is based on the parameters θ_f (defining the feature space, that is shared with the function h) and θ_s, involved in the classification for the self supervised task.
- during the training, for each iteration a batch of images is extracted from the data loader:
  - a first forward pass of the network is performed with the original images in order to train the model on the detection task. The output of the function $h(x|\theta_f, \theta_d)$ is compared with the ground truth labels and a *detection loss* $\mathcal{L}_d$ is computed;
  - a second forward pass involves transformations of the original images. Each image is transformed following the definition of the self supervised task that also specifies how to produce a self supervision signal (possible transformations are rotations or reshuffling of tiles). The transformed images are passed through the network and an output is computed using the function $j(x|\theta_f, \theta_s)$. Comparing this output with the automatically generated label a *self supervised loss* $\mathcal{L}_{ss}$ is computed.
- the two loss functions are combined to compute the final loss:

$$\mathcal{L}_{tot} = \mathcal{L}_d + \alpha \cdot \mathcal{L}_{ss}$$

- the hyper-parameter α allows to adjust the importance of the self supervised task during the training;
- backpropagation is applied considering the total loss to update the weights of the networks (represented by $\theta_f, \theta_d, \theta_s$).

It is important to keep in mind that, for an effective simultaneous training, it is necessary to design the architecture so that the parameters involved in the solution of the primary task, and the ones involved in the solution of the auxiliary task, are largely shared. This is necessary because, at inference time, the auxiliary task is not considered anymore, but the ability of the network to solve it should be reflected in the performance of the network on the main task.

We will see that, while the integration of the auxiliary task in the SSD network is straightforward, the particularity of the Faster R-CNN architecture requires to integrate this task in a particular way. As a result, in this case, for each iteration, more than 2 forward passes of the networks are performed. I will describe this procedure with more details later. Anyway the main idea of computing a loss $\mathcal{L}_d$ for the detector and a loss $\mathcal{L}_{ss}$ for the self supervised task in order to obtain a final loss that is $\mathcal{L}_{tot} = \mathcal{L}_d + \alpha \cdot \mathcal{L}_{ss}$ does not change.

3.1.4 Integration in SSD

Following the Jigen approach presented by Carlucci *et al.* in [20], **the self supervised task should be introduced inside a network as a new branch** followed by a classification header that should be put in parallel with the original one. The self supervised task is, in fact, usually defined as a classification task for which the class label is generated when applying the specific transformation of the chosen task to the input image. At the same time the 2 different tasks (the self supervised one and the main one) should be performed simultaneously during the training.

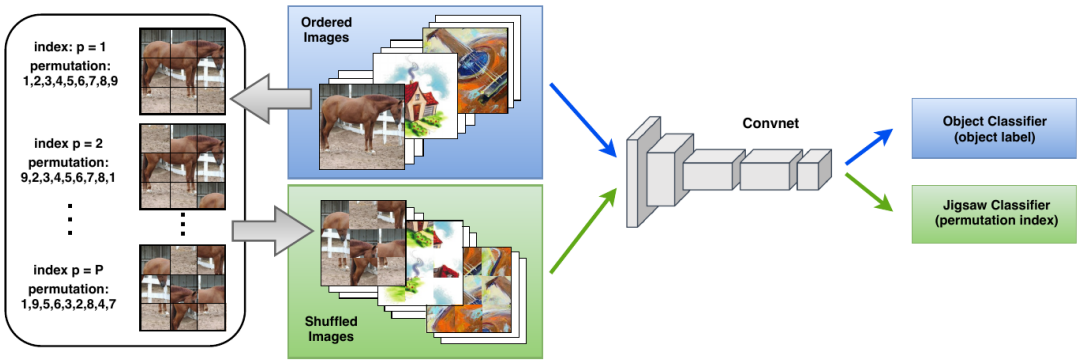


Figure 3.1: Simplified representation of a deep CNN that should solve two tasks: a main classification task and a self supervised task, in this case the Jigsaw task. Image taken from [20].

The reason why we want to attach the new branch at the end of the network is that the purpose of this architecture is to have **a single backbone** whose layers have to learn a representation of the visual world that must be good both to solve the primary task and the auxiliary one. Only through this **feature sharing** we can hope to have some advantage from the training on the auxiliary task that is maintained also at inference time when the auxiliary task's branch is not used any more.

In the case of the SSD architecture, if we want to attach the new classification header in a position where it can influence the whole network, we should consider adding it after the last extra convolutional layer already added after the VGG backbone. However this position is not valid for any possible self supervised task as the last extra convolutional layer produces for each image a set of 256 feature maps of size 1×1. This means that these feature maps do not contain spatial information anymore. It is therefore not possible to use them as input for a classification

header that has to reason on the **spatial correlation of objects parts** like is the case, for example, of the Jigsaw task (see 3.1.1). On the contrary, other tasks (e.g. the rotation task described in section 3.1.2) do not need to have information on the spatial correlations to be performed and can then be attached in this position.

For this reason I decided to implement the classification header for the self supervised task in two different positions:

- in the case of the Jigsaw task the header is added after the second-last extra layer which produces 256 feature maps of size 3×3 and is therefore suitable to perform this task. As a result only the weights of the last extra layer are not influenced by the training on the supervised task;
- in the case of the rotation task the header is added after the last extra convolutional layer.

At training time a mini batch of images is extracted from the training dataset and used to perform a forward pass in order to compute the detection loss. After this step the batch is passed through the transform operation defined by the self supervised task and a second forward pass of the network is performed to compute the loss for the auxiliary task. The weight α is applied to the loss of this task in order to regulate its importance w.r.t. the one of the primary task as described in section 3.1.3.

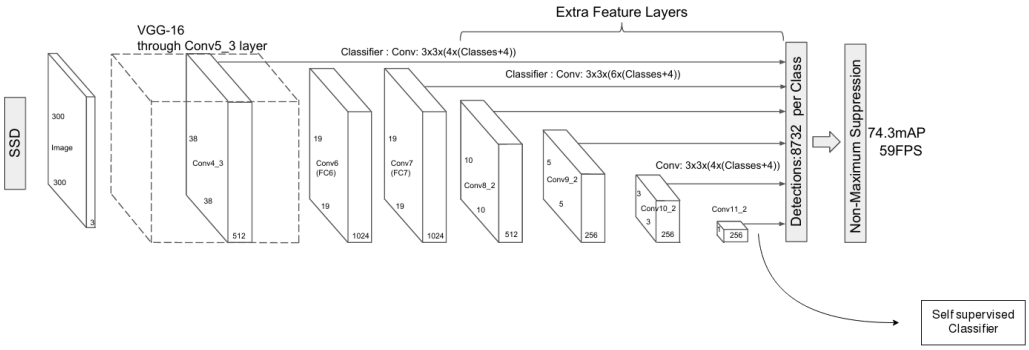


Figure 3.2: Representation of the architecture of SSD with a self supervised branch added in order to perform the rotation task

3.1.5 Integration in Faster R-CNN

The self supervised classification branch in the case of the Faster R-CNN architecture should not be attached to the network using the same logic followed for the SSD architecture. This is a consequence of the fact that Faster R-CNN's training follows the efficient batch sampling strategy of Fast R-CNN [33] that I introduced in paragraph 2.2.1. In particular this means that:

- the number of different images in each training batch is really small (often 1, but bigger batches can be used if there is a lot of GPU memory available);
- only a small data augmentation strategy is used.

As a result the self supervised task applied on a batch of this kind would be trivial and, as a result, harmful for the success of the joint learning strategy. In fact, if the task is trivial the network will solve it using image cues not relevant to extract semantics. That's particularly evident with some types of images. For example images samples from the Cityscapes dataset are all structured in the same way: being taken using a dashcam they represent street scenes with a road in the center, some buildings on the sides and in the background and a car hood in the bottom. If the *rotation* self supervised task is applied to an image of this kind then it can be solved simply by looking at the position of the car hood.

This problem is less relevant in the case of the SSD architecture that relies for the training on:

- a bigger batch size: when many images are available the probability of the task to be trivial for all of them is lower;
- a stronger data augmentation procedure: some data augmentation transformations like random crops can help in excluding some visual cues that make the task trivial.

To solve this problem in Faster R-CNN I designed an architecture focused on keeping the self supervised task more complex for the network. This is achieved by requiring the algorithm to learn how to solve this task on the single regions containing the objects instead of the whole image. By following this principle I had the possibility to choose between 3 types of regions for the task:

- **ground truth bounding boxes** (or *targets*). By applying the self supervised task on the targets I can be sure that the network will use this auxiliary task to learn a representation only of the parts of the images that contain the objects I am interested in. In practice I ask the network to learn something on the objects by automatically discarding the background. For example, if the self supervised task is the *rotation* task, then, through this learning objective, the network will be trained to learn the natural orientation of the objects without the possibility to exploit visual cues like the position of the sky in the image;
- **bounding boxes of detected objects** (or *detections*). If the detected bounding boxes are used for the self supervised task's application I can not have the certainty that the network will be trained only on the objects I am interested in. This is particularly true at the beginning of the training, when I will certainly have many False Positives and False Negatives between the detections. At the same time, this could be a good choice if I want my self supervised task to be completely unsupervised;
- **proposed regions** (or *proposals*). A network trained on the self supervised task defined on region proposals certainly looks not only at the interesting objects, but also, at least partially, at the background. However we should take into account that the RPN (see paragraph 2.2.2) learns during the training to produce proposals of higher quality and so, especially after some iterations, also this strategy could become useful. Furthermore it is possible that, looking at the proposals, being forced to take into account a higher number of regions, the network will find the task complex and so more useful.

A naive way to implement the self supervised task at region-wise level would be to:

1. choose a region type between *targets*, *detections* and *proposals* and extract the regions by cropping:
   (a) in the *targets* case the images could be directly cropped producing a number of different smaller images equal to the number of ground truth bounding boxes;

   (b) in the other two cases a forward pass of the network should be performed. Once the proposals or the detections are defined the original images can be cropped to select only the desired regions;

2. apply to each cropped image the transformation defined by the chosen self supervised task;

3. perform a forward pass of each transformed region, compute the loss using the output of the self supervised classifier and then perform the backpropagation.

This type of implementation, despite of being effective, is in reality **not efficient at all** and, in particular, it is at the **opposite of the philosophy behind Faster R-CNN**. In fact, cropping the image around the regions and then passing each cropped object to the network again means to go back to the strategy used by the original R-CNN algorithm [2] (see section 2.2.1). I consequently decided to follow the paradigm of Faster R-CNN, in order to minimize the number of forward passes of the complete network. I had to **discard the self supervised task based on Jigsaw puzzles** (see section 3.1.1) because its implementation in this pipeline would be too difficult. However this is not a significant limitation considering that my experiments on the SSD architecture showed that this task does not perform well on images representing scenes (see section 4.1).

I therefore implemented only the rotation self supervised task. However **I decided to support all the 3 options for the definition of the regions** in order to be able to assess the effectiveness of all these methods. I wrote my code so that it accepts as a configuration parameter one of those options and this allowed me to perform experiments on all of them to understand which was the best.

An important aspect that has to be taken into account is that it is not possible to apply the rotation on the features that were extracted by the network for each region and then to perform the self supervised task on the same ones. In fact in this case, even if the classification is performed on rotated features, those ones have been computed on the original non-rotated image so that any object contained in it has already been recognized by the trained layers. As a consequence, in this case the self supervised task is once again trivial. This can be easily understood considering that, even for humans, it is more difficult to recognize a certain rotation of an image if we don't know *a priori* its content.

The **requirements** for my implementation then were:

- to support the *rotation* task at the region-level. This means that each single region should be transformed independently and the task must be solved on it independently;

- to use for the self supervised task features which were computed on rotated images and not rotated features computed on the original images;

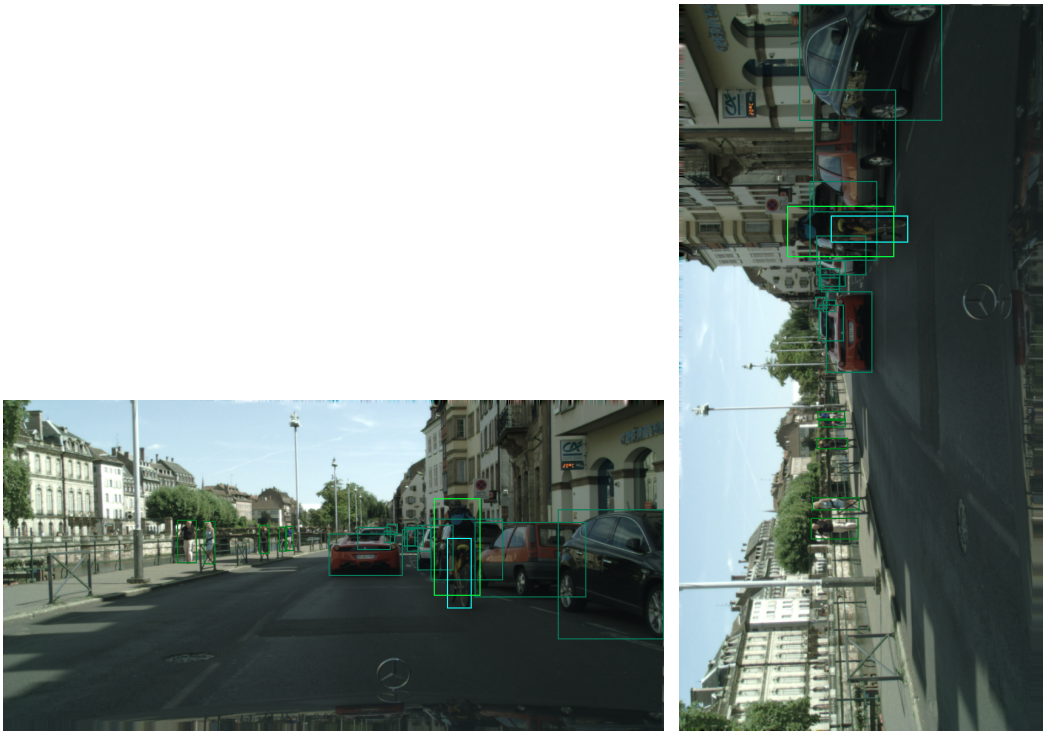
- to minimize the number of the forward passes of the whole network.

I therefore had to modify the Faster R-CNN architecture and in particular I had to intervene on the forward strategy. During the training each time a mini bath of images is passed through the network then, if the self supervised task is enabled:

1. each image is rotated to all the available positions, so that all the variants are available;

2. all the variants of the images are passed through the network backbone;

3. following the launch configuration a type of region is chosen and a list of bounding boxes is defined:



(a) Original *straight* image (b) 90° rotation

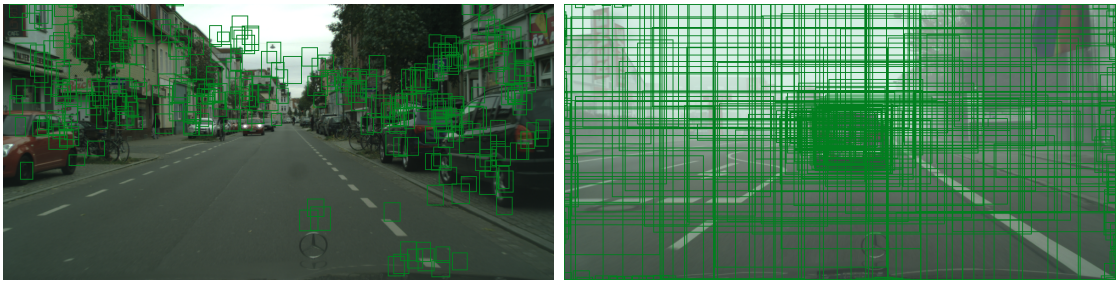
Figure 3.3: Example for the *targets* strategy for the application of the self supervised task

(a) Example of *detections* proposed at the beginning of the training (b) Example of *detections* proposed at the end of the training

Figure 3.4: Use of the *detections* strategy for the application of the self supervised task. Comparison between the detections proposed at the beginning of the training and those proposed at the end of the training

- if the chosen type is *targets* then the ground truth bounding boxes are directly used;
- if the chosen type is *detections* or *proposals* it is first of all necessary to complete the forward pass using the non-rotated image. This pass allows to define proposals and then detections;

4. once the regions have been defined we need to choose randomly for each of them a rotation



(a) Example of *proposals* produced at the beginning of the training

(b) Example of *proposals* produced at the end of the training

Figure 3.5: Use of the *proposals* strategy for the application of the self supervised task. Comparison between the region proposals at the beginning of the training and those of the end of the training

and extract a set of features that must represent that region in the chosen rotated image. This means that, by iterating over all the regions:

(a) we randomly choose one of the possible rotations;

(b) we rotate the definition of the region to follow the chosen rotation;

(c) we use the features extracted from the whole rotated image and the rotated region to apply RoI pooling (see paragraph 2.2.1);

(d) we obtain a fixed-size feature vector representing the chosen region in the chosen rotated image. We apply the self supervised classifier to compute probabilities for the possible rotations. We put the extracted probabilities vector in a list;

(e) we put the index corresponding to the chosen rotation in a list that represents the target for our self supervised task;

5. we pass the two lists to the negative log-likelihood loss function that will compute the loss associated to the self supervised task for this forward execution.

Experiments performed with this architecture showed that the *proposals* strategy is too slow as too many regions are used for the application of the self supervised task. For what concerns the *detections* strategy the results are good, but the experiments are still a bit too slow. For this reason for my final experiments I decided to define the regions on which to apply the self supervised task by the *targets*, when they are available, the *detections*, in the other situations.

3.2 Model usage in the different settings

I will now outline how I used the described models to obtain well performing algorithms in the 3 cross-domain analysis settings presented before: Domain Generalization, Domain Adaptation and One-Sample Adaptation.

3.2.1 Domain Generalization

The Domain Generalization settings have been designed to enable the development of computer vision algorithms that can be used in the wild. The idea is that every time the target domain is

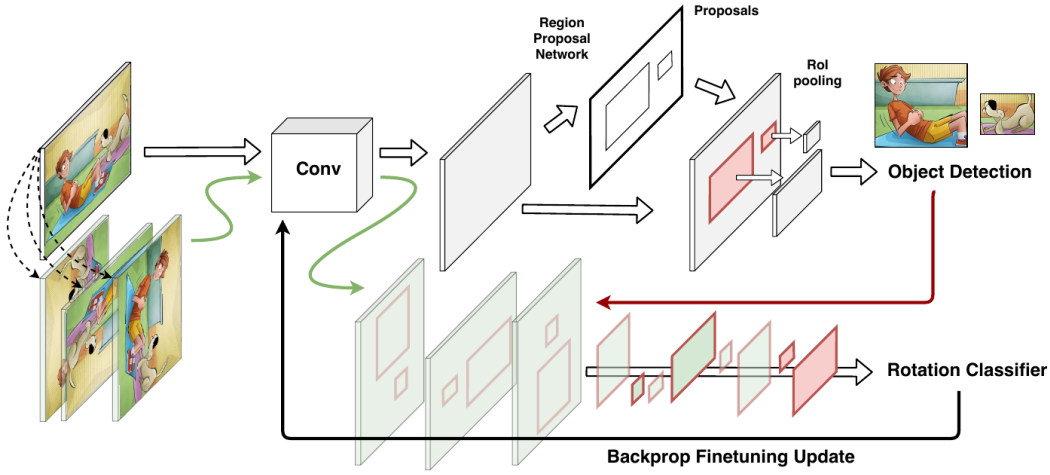


Figure 3.6: High level representation of Faster R-CNN's architecture implementing the region-wise self supervised task using the *detections* as regions.

not known or accessible *a priori*, than Domain Adaptation algorithms are useless. In these cases a model trained in the DG settings should be really able to generalize and it could then be used successfully.

The use of a self supervised task to improve the generalization ability of an object detection model is straightforward:

- use of multi source training. This means that each mini batch is made of equal parts coming from different datasets and representing different domains. All the source domains must be supervised;
- the model is trained following the multi-task approach: all the images are used to train both the object detector and the classifier of the self supervised task:
  - thanks to the training on the main task, the model learns how to perform the visual object detection task;
  - thanks to the training on the auxiliary task, the model learns about the spatial correlation of objects parts or about the visual appearance of straight vs rotated objects in the various different domains.

We argue that this allows to obtain a domain-invariant model as the representation learned through the self supervised task does not differ between the domains. The performance of the trained model are measured by testing it on a target domain that is different from all the source domains. At test time the model is ready to be used: the self supervised task is not performed any more, but the advantage of its usage during the training is maintained by the learned features.

3.2.2 Domain Adaptation

In the Domain Adaptation settings we still want to use our self supervised task to build a model that is domain-invariant. In particular we want a model that is invariant to the differences

between the source domain and the target domain that in this case is known *a priori*. For my experiments I used a single source approach, but the my strategy can be exploited even in a multi source scenario. The approach can be summarized as follows:

1. we have a supervised dataset representing a source domain and an unsupervised set of images coming from the target domain;

2. we train the model following a multi-task approach:

    - the supervised data is used for the training on the detection task;
    - both the supervised and the unsupervised data are used for the training on the auxiliary self supervised task.

At test time samples extracted from the target domain are used to test the performance of the model. These samples can be the same used for the adaptation (*transductive settings*) or new samples (*non-transductive settings*).

It has to be noticed that both in Domain Adaptation and in Domain Generalization settings, the described training strategy is end-to-end and based on a single step. This is not always the case in cross domain analysis settings as some methods require to follow some kind of multi-step procedure [14] or to perform preprocessing operations [11] [27].

3.2.3 One-Sample Adaptation

The idea of One-Sample Adaptation is very similar to the one of Domain Generalization: to build a model ready to be used in the wild. From this point of view the two settings can be assimilated. The main difference is that *traditional* Domain Generalization algorithms do not take into account the possibility to use, at test time, the sample they receive for the inference as a source of information to learn something on the sample itself, before performing the inference. This idea is more similar to what is done in the **Online Domain Adaptation** settings but there are some differences even w.r.t. these settings:

- in the online domain adaptation settings an algorithm should be able to adapt to a domain thanks to the samples it receives from this domain in batches. Each new batch allows the algorithm to adapt itself more and more;

- in the One-Sample Adaptation settings only one sample at a time is handed to the model. This is the sample on which the model should perform an inference, but before doing so it can use this sample to adapt itself. Once the inference has been performed, the adaptation done on the sample is discarded so that the model is not biased towards that sample when receiving the following one.

It should be noticed that, from the point of view of the end user, there is no difference between an algorithm designed for the DG settings and one designed for OSA: both are ready to be used in the wild.

I will therefore now present how a self supervised task can be used to build a model that at test time can perform One-Sample Adaptation. The approach used has been called **Self-ADE** (***Self*** *supervised cross-domain* ***A****daptive* ***DE****tection*).

Train time

In order to get a model which is able to adapt to a sample, at test time, using a self supervised task, it is necessary for it to already be able to solve this task. For this reason at training time the model should be pushed to learn how to perform both the supervised task and the self-supervised one.

The Self-ADE strategy requires to use at train time the data coming from the source domain to train the model on both tasks. For each batch of samples:

- both the images and the labels are used for the training on the main *supervised* task (visual object detection);
- the images on their own are used for the training on the self supervised task. Depending on the chosen auxiliary task, the necessary transformation can be applied on the images;
- a task weight α is, as always, applied to the loss of the self supervised task in order to regulate the relative importance of the 2 tasks.

We call this one the **pre-training phase** as its output is a model, trained on both the main and the auxiliary task, that is then used to re-initialize the network any time a new inference has to be performed.

Test time

Once the model has been trained it is ready to perform inferences. We call this one the **adaptive phase**, as before each inference an adaptation is performed on each sample. However it should be noticed that the adaptation is completely transparent to the user and, as a consequence, this phase simply represents the test time operation. The **inference process** of the Self-ADE approach can be described as follows:

1. a sample is received;
2. the model trained on the source domain is loaded from scratch;
3. the sample is used to perform a number of training iterations that represents a *finetuning*:
   (a) each batch of images is built by taking a number of copies of the sample. If the chosen algorithm, like SSD, requires to use a Data Augmentation procedure this one is applied. Together with the application of the transformation of the self supervised task, this allows to obtain different images in the batch even if all of them have originally been copied from the same sample;
   (b) for each batch a forward pass of the network is performed. Given that the labels for the main task are not available, only the loss associated to the self supervised task is computed;
   (c) through the backpropagation the model is trained on the sample;
4. the adapted model is used to perform an inference and returns the result.

Chapter 4

Experiments and results

4.1 Domain Generalization with SSD

The first experiments I performed had as a purpose the assessment of the effectiveness of the self supervised approach as a domain generalization technique for the visual object detection task. This part of my work is an extension of the work of Carlucci et al., presented in [20], to the object detection task.

As the domain generalization settings have not been studied in the context of the object detection task before, there are not other works that can be used for a direct comparison with my results. For this reason, as a method of assessment of the effectiveness of my approach, I made a comparison with the results obtained by a baseline.

		Clipart	Comic	Watercolor	Average
Baseline		38.59	35.72	54.37	42.89
Jigsaw	$\alpha = 0.05$	38.91	37.03	54.41	43.45
	$\alpha = 0.1$	38.64	35.35	55.33	43.11
Rotation	$\alpha = 0.05$	39.38	37.04	55.18	43.87
	$\alpha = 0.1$	38.36	36.59	**56.26**	43.74
	$\alpha = 0.2$	39.76	37.06	55.33	44.05
	$\alpha = 0.4$	**39.82**	36.64	56.07	44.17
	$\alpha = 0.8$	39.36	**37.29**	55.90	**44.18**

Table 4.1: Summary of results of our SSD based DG method on the VOC → AMDs benchmark

4.1.1 Description of the experimental protocol

I performed my experiments in DG with SSD using a benchmark built on the Artistic Media Datasets (see section 1.3.1) and Pascal VOC 2007 and 2012 sets [10]. My implementation of SSD is based on the code provided in [38] that uses PyTorch [39] as deep learning framework. In particular I used, for all my experiments, the variant of SSD often called *SSD300* which relies on a VGG16 [32] backbone and images of size 300x300. Due to the absence of other works that deal with the object detection problem in the Domain Generalization settings I had to build my experimental procedure from scratch taking inspiration from works that study DG for other tasks (like [20]) and works focused on the object detection in the domain adaptation settings (like [11]):

- I used **multi-source training**. Of the 4 available domains I decided to always keep the real world domain (represented by the photos in Pascal VOC datasets) between the source domains, while one of the other 3 is always left out during the training in order to be used as target domain. Details:
  - for the real world domain I used the *trainval* split both for the Pascal VOC 2007 and 2012 sets. In total there are 16551 annotated images for this domain;
  - for all the AMDs I used both the *train* and *test* split together. This means that when e.g. comic2k is a source domain then all the 2k images are used during the training and when, on the contrary, it is the target domain then all the 2000 images are used to evaluate the performance of the model;
  - as specified before, as comic2k and watercolor2k contain only 6 classes of objects that are a subset of the 20 classes of clipart1k and Pascal VOC it was necessary, for this multi-source experiments, to filter out all the instances of the other 14 classes. As a result only 724 of the 1000 images of clipart1k were used, while the number for Pascal VOC goes from 16551 to 11263. The classes that I kept are: bike, bird, car, cat, dog and person. From a practical point of view the filtering was performed in this way:
    * the annotation for each image in the datasets contains a list of instances, each one consisting of a class label, a bounding box definition and a flag which specify if the instance is *difficult*;
    * *difficult instances* are a minority in the datasets, they are discarded at train time and they are not considered when computing the performance of a model. This means that, if there are two different models that detect in the same way the non-difficult instances but one detects also the difficult instances while the other no, then the two models will obtain the same computed performances;
    * when a dataset is instantiated in the code, a loop over the list of all its images is performed. For each image we check if there is at least one instance belonging to one of the classes we are interested in and that is not marked as difficult. If this is not the case the image is discarded;

- for each experimental setting of Table 4.1 the most important result is the average mAP. In fact, for each configuration 3 experiments have been performed using each time one of the AMDs as target domain. However, in DG we want to evaluate the effectiveness of a configuration and this can be assessed only by looking at the averaged result;

- as the results of the experiments in cross domain settings often vary considerably between one execution of the training and another, all the different experiments have been performed 5 times. The results of the various runs have then be averaged.

All the experiments share the main hyperparameters and the training procedure:

- the Data Augmentation strategy used is the same of the original SSD paper ([6]). For this reason I decided not to use the *random expand* transformation I described in section 2.1.3, which was added only in a second version of the paper;

- I used a batch size of 24 built by taking each time 8 samples from the 3 source domains. For what concerns the experiments with the self supervised task, at training time, each image is used both for the training of the detector and, once transformed, for the training on the auxiliary task. As a consequence, in these cases, the actual batch size is 48;

- I performed 120k iterations with an initial learning rate of 10^{-3} and two decay steps at 80k and 100k;
- I used 10% of data from each source domain to build a validation set. This is used to perform a periodic evaluation of the performance. In particular every 5000 iterations the model was tested on the validation set and after the 120k iterations the model that obtained the best results was used to compute the performances on the target domain.

4.1.2 Description of the methods

Baseline experiments The baseline experiments have been performed by using the SSD architecture as it is, i.e. by training on all the source domains without enabling the self supervised task. This type of baseline is often called *Deep All* ([9] [20]) and, as stated by *Li et al.* in [9], it often performs well, especially when the domain shift between the analyzed domains is small.
It should be noticed that the effectiveness of a Deep All method does not allow us to use it even in non cross-domain settings: if the target domain is known a priori and a large training dataset belonging to it is available it is still better to use a standard training procedure not based on a DG method. The rationale behind this consideration is that in multi-source training settings the network is forced to learn features that are good to represents all the data and so taht are probably domain-invariant. As a result, these features are not as suitable to represent only one of the source domains as domain-specific features learned in a standard single-source training could be.

Experiments with the self supervised task The experiments with the self supervised task have been performed following the same approach used for the baselines. The purpose of these experiments is to analyze how different self supervised tasks and various values for the weight of this task, w.r.t. the main one, can influence the performance of the model in the DG settings.

I performed my experiments with 2 different self supervised tasks and a range of different weights: $\{0.05, 0.1, 0.2, 0.4, 0.8\}$. The weight α is used to regulate the relative importance between the main task and the auxiliary one. If the value of this weight is too high, then the self supervised task may influence the training of the network too much forcing it to learn features that are more suitable to solve the auxiliary task than the main one. At the same time a too small weight can induce the network to neglect the auxiliary task in favor of the main one.
I chose to try as self supervised tasks:

- the Jigsaw task (see section 3.1.1), as this is the one that guarantees the best performance in the object recognition task as reported by *Carlucci et al.*;
- the rotation task (see section 3.1.2), that may give better results with images representing scenes. The Jigsaw task, in fact, is designed to teach the network about spatial correlation between object parts and this could be ineffective if there are more objects in each image or if single patches contain entire objects.

4.1.3 Detailed results and analysis

The results presented in Table 4.1 demonstrate our hypothesis: the performances obtained with the rotation self supervised task are better than those obtained with the Jigsaw task. After having ascertained this, I decided to not perform the complete series of experiments with the Jigsaw task in order to focus on the other one for which the experiments showed greater improvements. In particular we can see that different values of α allow us to obtain the best improvement over

the baseline for the 3 different target domains. In the Domain Generalization settings, however, what really matters is the average improvement, as the target domain represents only a randomly sampled domain over an unknown distribution of possible target domains. As a result we can conclude that the model that guarantees the best improvement over the baselines is a model trained with the rotation self supervised task and $\alpha = 0.8$.

		bike	bird	car	cat	dog	person	**mAP**
Baseline		58.07	31.76	**39.48**	12.53	24.28	65.42	38.59
Jigsaw	$\alpha = 0.05$	60.17	32.30	38.50	15.24	22.17	65.04	38.91
	$\alpha = 0.1$	58.64	**33.05**	38.49	15.02	24.26	65.04	39.08
Rotation	$\alpha = 0.05$	58.96	32.61	37.04	16.72	**25.41**	65.53	39.38
	$\alpha = 0.1$	56.88	32.82	38.23	14.70	21.87	65.68	38.36
	$\alpha = 0.2$	61.50	32.10	38.61	15.61	24.83	65.94	39.76
	$\alpha = 0.4$	**61.90**	32.31	37.21	**17.21**	24.53	65.77	**39.82**
	$\alpha = 0.8$	59.53	33.00	38.40	16.21	22.73	**66.30**	39.36

Table 4.2: Results of the experiments using Clipart1k as target domain

		bike	bird	car	cat	dog	person	**mAP**
Baseline		36.95	23.01	30.83	32.21	29.90	61.41	35.72
Jigsaw	$\alpha = 0.05$	**40.21**	22.03	34.61	33.79	29.86	61.68	37.03
	$\alpha = 0.1$	38.36	23.17	32.62	28.53	27.57	61.83	35.35
Rotation	$\alpha = 0.05$	36.62	23.05	34.82	**35.87**	30.35	**69.01**	37.04
	$\alpha = 0.1$	37.40	23.21	32.77	33.56	30.31	62.26	36.59
	$\alpha = 0.2$	39.33	22.96	34.03	34.76	29.43	31.82	37.06
	$\alpha = 0.4$	36.28	**23.60**	33.23	34.40	30.77	61.58	36.64
	$\alpha = 0.8$	28.15	21.71	**35.96**	34.70	**31.22**	61.99	**37.29**

Table 4.3: Results of the experiments using Comic2k as target domain

		bike	bird	car	cat	dog	person	**mAP**
Baseline		58.07	31.76	39.48	12.53	24.28	65.42	38.59
Jigsaw	$\alpha = 0.05$	60.58	53.11	48.89	44.34	45.84	73.68	54.41
	$\alpha = 0.1$	61.41	53.28	50.45	47.29	45.70	73.84	55.33
Rotation	$\alpha = 0.05$	62.43	51.80	51.32	46.21	45.80	73.49	55.18
	$\alpha = 0.1$	64.76	52.52	53.74	46.73	46.12	73.70	56.26
	$\alpha = 0.2$	61.95	54.24	50.55	44.81	46.47	74.01	55.33
	$\alpha = 0.4$	63.69	52.82	51.35	45.93	48.35	74.30	56.07
	$\alpha = 0.8$	63.86	53.35	52.58	46.15	45.25	74.20	55.90

Table 4.4: Results of the experiments using Watercolor2k as target domain

4.2 Domain Adaptation

The DA settings for the visual object detections task have already been studied more than the DG settings. This allowed me to compare the performances of our method w.r.t. the current state of the art algorithms. I performed experiments using both SSD and Faster R-CNN for the popular VOC → AMDs DA benchmark. I also tested my Faster R-CNN implementation in the Cityscapes → Foggy Cityscapes benchmark.

4.2.1 Discussion on the methods

Before showing the results of my experiments I will briefly describe all the methods that appear in my tables in order to highlight the differences between our approach and the others.

SSD methods

Baseline The baseline model is simply an SSD network trained on PASCAL VOC 2007 and 2012's *trainval*s splits and tested on the target domains.

SSD RST This is our approach. The detector is trained on the source images while the rotation classifier receives both source and target images (RST = Rotation Source Target). For the description of how I implemented the self supervised task for the Domain Adaptation settings see section 3.2.2.

DANN Domain Adversarial training of Neural Network (DANN) is a widely used and known method for Unsupervised Domain Adaptation presented by Ganin *et al.* in [13]. It was adapted to be used for object detection with SSD by the authors of [26]. DANN is based on the integration into a standard network of a domain classifier attached as a new branch through a special layer called Gradient Reversal Layer (GRL). The purpose of this layer is to reverse the gradient during the backpropagation so that the *feature extractor*-part of the network and the domain classifier can be trained using an adversarial approach. This simple method of adversarial training based on a single network allows to push the *feature extractor* towards learning how to extract domain-invariant features. This process is often called *feature alignment* between domains.

ADDA Adversarial Discriminative Domain Adaptation was presented by *Tzeng et al.* in [14]. Similarly to DANN it is based on adversarial training, but it is composed by a multi-step procedure:

- **Pre-training**, a classifier is trained on the source images in order to produce a feature extractor that outputs features which are relevant for the classification task;
- **Adversarial Adaptation**, a *feature extractor* is trained for the target domain. The training objective is to maximize the confusion of a domain discriminator that receives features extracted from source domain samples by the *source feature extractor* and features extracted from target domain samples by the *target feature extractor*. in this way the target feature extractor learns how to extract from target samples features that are similar to those extracted by the source feature extractor from source samples;
- **Testing**. To perform an inference the model uses the target feature extractor to produce a representation of the sample that is then classified using the source classifier.

ADDA was adapted to the object detection task on the SSD algorithm by the authors of [11].

Self training and Adversarial Background regularization [26] The problem of using an adversarial approach to obtain global feature alignment is that usually the background is not transferable from one domain to another. For this reason the proposed method tries to obtain a feature alignment only on the foreground objects. The method is based on:

- **Weak Self Training** (WST). *Self training* strategies have been widely used before. They consist in the use of a model trained on the source domain to produce predictions on the target domain samples, the idea is to use those predictions as ground truth for the training of another model. The problem of a naive application of this method in object detection is that the intermediate detector produces a lot of False Positives and False Negatives that are used as ground truth and that, therefore, degrade the performance of the final detector. As a result *Weak Self Training* is used in order to:
  - reduce False Negatives. This is obtained using *weak negative mining* instead of *hard negative mining*. The result of this approach is that the only considered negative samples are the most reliable ones;
  - reduce False Positives. This is obtained computing a reliability score for the detections (considering how many RoIs around each detection received the same class prediction) and discarding those detections that have a lower reliability score;

  This strategy allows to obtain more reliable pseudo labels useful to perform a better training of the second model. These labels are used only in terms of class information, not in term of bounding box predictions that are not reliable in a cross-domain environment;
- **Adversarial Background Score Regularization** (BSR). The purpose of this module is to force the feature extractor of the base network to output features suitable in the discrimination of the background from the foreground. It is based on the use of a **GRL** and a classifier that tries to distinguish between background and foreground. In this way the network learns which are the most discriminative features.

The two main features of this method, WST and BSR, are complementary:

- WST provides class labels for the objects;
- BSR allows the network to distinguish between the background and the objects.

Domain Transfer [11] In [11] Inoue *et al.* proposed a method for *weakly* supervised domain adaptation for the object detection task. Their method is based on a 3-step procedure:

1. pre-training of an SSD300 model on the source domain;
2. **Domain Transfer** (DT). A CycleGAN [15] is used to transform the images of the source domain to bring them in the target domain (i.e.: transform Pascal VOC's images to bring them to one of the AMDs). Then a finetuning of the network is performed using these translated images for which the ground truth information correspond to the source domain's one;
3. **Pseudo Labeling** (PL). Image-level annotations for the target domain images are used (for this reason the settings are called *weakly supervised*) to produce instance-level annotations using the previously trained model. These annotations are used to perform another fine tuning step.

The weakly supervised settings can not be compared with the unsupervised domain adaptation (UDA) settings. However the proposed method, stopped after the DT step (for which there is no use of image-level annotations), respects the unsupervised constraint and it still has state-of-the art performances on the single-stage object detection task.

Faster R-CNN methods

Baseline The baseline model is a standard Faster R-CNN network with a ResNet-50 [40] backbone pretrained on ImageNet. The model is trained only on the source, and tested on the different targets.

Faster R-CNN-RS and RST I integrated the self supervised rotation task in the network as I described in section 3.1.5. I call **Faster R-CNN RS** the model trained both on the main and auxiliary task using only source domain data, while **Faster R-CNN RST** is the model trained on the main task using source data and on the auxiliary task using both source and target data. It has to be noticed that Faster R-CNN-RS can be seen as a different type of baseline as it is trained only on the source data.

DivMatch *Diversify and Match* [27] is a DA method for object detection based on Faster R-CNN that represents the current state-of-the-art for these settings. The method can be seen as an evolution of Domain Transfer as, like this one, it uses CycleGANs to generate variants of the source domain. DivMatch, however, is a more complex and effective approach based on 2 steps:

1. **Domain Diversification**. The ability of CycleGANs to produce different variants of an image is exploited not only to translate images from the source domain to the target one, but also to generate new domains. This allows to obtain 4 different versions of the source domain images, all of them with instance-level annotations;
2. **Multi-domain-invariant Representation learning**. A joint training of a detector and of a domain classifier is then performed using adversarial learning in order to align the features of all the domains produced by DD. The strategy used for this step is similar to the approach used by DANN, but with more than 2 domains that have to be distinguished by the domain discriminator.

StrongWeak [28]. Images coming from different domains could be characterized by strongly different scenes structures and object combinations. As a result a global feature alignment between source and target domains could not be possible. For this reason this method employs GRLs and adversarial learning techniques at two different levels:

- to obtain a strong alignment between lower level features. These features usually describe textures and colors, that are often fundamental aspects characteristic of each domain and for this reason should be aligned in order to obtain good cross-domain performance;
- to obtain a weak alignment between higher level features. This weak alignment should help without hurting the performance of the model.

4.2.2 Description of the experimental protocol

In order to make fair comparisons I had to re-run some of the methods that I presented before. This was necessary for the methods for which the original reported results used a different

configuration w.r.t. mine. I will therefore now specify how I configured the experiments I performed. I repeated each of my experiments 3 times in order to obtain more reliable results.

SSD experiments

Baseline For my experiments I used the same implementation of SSD described in section 4.1.1. Some differences in the configuration of the experiments are due to the need of having results comparable with the other methods. Main hyperparameters:

- initial learning rate of 0.001 stepped down by 1/10 after 80k and 100k iterations;
- batch size equals to 32;
- 120k iterations in total. As the DA benchmarks do not include a validation split we use the model obtained after the last iteration for evaluating the results;
- the data augmentation techniques are those of the last version of the original paper: random expand, crop, horizontal flip and photometric distortion. The images are resized to 300x300 before entering them in the network;
- non standard parameters: NMS threshold of 0.45 and confidence threshold of 0.01;
- I decided to use *non transductive* settings. This means that the target domain samples used for the adaptation during the training are different from the target domain samples on which the trained model is tested. This is the same procedure followed in [11], but not the same followed in [26]. In the latter work, in fact, *Kim et al.* decided to use *transductive* settings when using clipart1k as target;

SSD RST The main hyperparameters are the same used for the baseline. In addition:

- I used a batch size of 32 both for the supervised source domain data and the unsupervised target domain data. This means that the overall batch size is 96: 32 source images for the detector training and 64 rotated images (32 from the source dataset plus 32 from the target one) to train the rotation classifier;
- I used $\alpha = 0.8$ as weight for the self supervised task as this was the one that guaranteed the best performance in the DG settings.

Faster R-CNN experiments

Baseline I used a public PyTorch based implementation of Faster R-CNN developed by Facebook [41]. Configuration:

- use of a ResNet-50 [40] backbone pretrained on ImageNet;
- the RPN produces 300 proposals after NMS. The extractions of anchors is done at 3 scales (128, 256, 512) and 3 different aspect ratios (1:1, 1:2, 2:1)
- images resized in order to have the shorter side equals to 600;
- batch size equals to 1.

The training strategy implemented depends on the benchmark:

- VOC → AMDs: there is no validation split so the network is trained for a fixed number of iterations that is 70k. The initial learning rate of 0.001 is decayed after 50k iterations;
- Cityscapes → Foggy Cityscapes. This benchmark contemplates a validation set that is the validation split of Cityscapes dataset. The training is performed for 30k iterations without learning rate decay. The model used for the final evaluation is the one with the best performances on the validation split.

Faster R-CNN RS and RST I modified the implementation of Faster R-CNN to include the self supervised task as described in section 3.1.5. The main configuration differences w.r.t. the baseline are:

- use of a batch size equal to:
  - 4 for the RS model: 1 source image is used for the detector training, the same image plus 3 rotated versions are used for the rotation classifier training;
  - 8 for the RST model: the source images are used as done by the RS model, 4 rotated versions of a target image are also used for the rotation classifier training.
- use of $\lambda = 0.05$ as weight for the self supervised loss function.

DivMatch In [27] *Kim et al.* presented their method using an implementation of Faster R-CNN based on a VGG16 backbone. In order to align their results with mine I downloaded their code and rerun the experiments using a ResNet-50 backbone. The obtained results are a bit higher than the ones obtained with the original backbone. As I used the official implementation of their method all the other hyperparameters were kept the same.

StrongWeak In [28] *Saito et al.* used a ResNet-101 backbone for their Faster R-CNN implementation. Once again in order to obtain results which are comparable with the others, I had to download their code and rerun the experiments using a ResNet-50 backbone.

4.2.3 Results and Analysis

	Clipart	**Comic**	**Watercolor**	**Average**
Baseline	25.94	19.32	46.83	30.70
DANN [26]	31.80	22.50	41.50	31.93
ADDA [11]	27.40	23.80	49.80	33.67
Self training [26]	35.70	26.80	49.90	37.47
DT [11]	**38.00**	29.80	50.40	39.40
SSD-RST (Our)	34.85	**30.92**	**53.05**	**39.61**

Table 4.5: Summary of results of SSD based DA methods in the VOC → AMDs benchmark

	Clipart	**Comic**	**Watercolor**	**Average**
Baseline	26.38	18.09	42.84	29.10
Faster RCNN-RS	28.96	17.93	47.14	31.34
DivMatch [27]	**39.73**	**36.92**	**54.70**	**43.78**
StrongWeak [28]	26.43	20.99	41.96	29.79
Faster RCNN-RST (Our)	34.32	32.03	52.04	39.46

Table 4.6: Summary of results of Faster R-CNN based DA methods in the VOC → AMDs benchmark

	mAP
Baseline	26.81
Faster RCNN-RS	29.33
DivMatch [27]	**34.20**
StrongWeak [28]	28.80
Faster RCNN-RST (Our)	30.76

Table 4.7: Summary of results of Faster R-CNN based DA methods in the Cityscapes → Foggy Cityscapes benchmark

Looking at the results we can notice that:

- for what concerns SSD, our self supervised based DA method outperforms previous state of the art methods. The improvement may seem quite small but it is more relevant if we consider that the previous top performing approach called *Domain Transfer* [11], is a more complex 2-step method. It is based on the use of CycleGANs [15] for the transfer of the source domain images to the target domain, followed by a fine tuning step;
- the results for Faster R-CNN in the VOC → AMDs benchmark are not as good as they do not reach state of the art performances. However the average mAP is still 10 points higher than the baseline;
- the results obtained by the **Faster R-CNN RS** baseline show a small improvement over the original baseline: this means that the self supervised task improves the generalization ability of the algorithm even when training only on the source domain;
- also in the Cityscapes → Foggy Cityscapes benchmark our method obtains results below the state of the art, but some points above the baseline.

In general we can conclude that the self supervised task improves cross domain performance also in the Domain Adaptation settings. This is a further confirmation of the effectiveness of this kind of approach in cross-domain analysis.

4.3 One-Sample Adaptation with SSD and Faster R-CNN

As the One-Sample Adaptation cross-domain settings are new, there are no previous methods available for a comparison. For this reason, in order to assess the performance of our method, we develop 3 different types of comparison:

- **0% target**. This corresponds to the One-Sample Adaptation settings. In this situation no target domain data is available during the training (the target domain could be unknown). In these settings we can compare our method only with baselines detectors trained only on the source domain;
- **100% target**. This corresponds to Unsupervised Domain Adaptation settings. We compare Self-ADE with DA methods even if our method was not designed for DA settings. This comparison is useful to show that, even in these settings, Self-ADE has competitive performances, even if it does not always reach state-of-the-art results. It must be kept in mind that this is not a fair comparison, as Self-ADE does not use target data during training like the others do and for this reason it has access to an overall lower number of target samples;
- **1% target**. This represents DA settings with reduced target sets available at training time. The purpose of these settings is to show that traditional DA methods do not perform well when they can access only a small number of target samples in the training phase.

We compare with the same methods that were already presented in section 4.2.1.

Experimental protocol

Self-ADE on SSD We evaluate each test sample only after a finetuning performed by initializing the model with SSD-RS and training on the self supervised task using multiple rotated and augmented versions of the sample. The finetuning is performed using batches of 32 images, for 50 iterations, with a learning rate of 0.001 that undergoes a linear warmup policy in the first 20 iterations.

Self-ADE on Faster R-CNN Before evaluating on each test sample we perform a finetuning initializing the model with Faster R-CNN-RS and training on the self supervised task using rotated versions of the sample image. The finetuning is performed using batches of 1 image, for 320 iterations, with a learning rate of 0.001 that undergoes a linear warmup policy in the first 20 iterations.

In **both cases** there is only one small change to the architecture used during the training of the original model: a dropout with probability $p = 0.5$ is applied before the rotation classifier. We empirically verified that this can help to regularize and prevent overfitting on the self supervised task.

4.3.1 0% target

Tables 4.8 and 4.9 confirm that, in our newly designed settings, Self-ADE shows better performances than all the other methods. We have an average improvement of 8.9 mAP points for SSD and of 7.2 mAP with Faster R-CNN. The effectiveness of the method is confirmed by the results obtained by Faster R-CNN's performance in the Cityscapes $\rightarrow$ Foggy Cityscapes benchmark.

		Clipart	Comic	Watercolor	Average
SSD	SSD baseline	26.8	24.9	49.6	33.8
	SSD-RS	26.1	20.4	47.3	31.3
	Self-ADE	**36.5**	**31.9**	**52.3**	**40.2**
Faster R-CNN	Faster R-CNN baseline	26.4	18.1	42.8	29.1
	Faster R-CNN-RS	29.0	17.9	47.1	31.3
	Self-ADE	**35.0**	**22.4**	**52.0**	**36.5**

Table 4.8: Summary of results of Self-ADE applied to SSD and Faster R-CNN on the VOC → AMDs benchmark. Comparison with methods that do not use target domain data at training time

		Foggy Cityscapes
Faster R-CNN	Faster R-CNN baseline	26.8
	Faster R-CNN-R	29.3
	Self-ADE	**32.2**

Table 4.9: Summary of results of Self-ADE Faster R-CNN on the Cityscapes → Foggy Cityscapes benchmark. Comparison with methods that do not use target domain data at training time

4.3.2 100% target

The comparison tables show that the Self-ADE approach performs well when compared with DA methods even if it is in a disadvantageous position. In fact all the DA methods in the table were trained using the train splits of the target domains as unlabeled datasets to be used for the adaptation. Only after this adaptation step those methods were tested on the test split of the target domains. On the contrary Self-ADE results have been computed by training the base network only on the source domain and then by adapting it on one sample at a time during the inference process. This means that the DA methods in Table had the possibility to look at an overall higher number of samples belonging to the target domain. It is interesting to notice that, for what concerns the SSD algorithm, Self-ADE has state of the art performances despite its disadvantageous position.

		Clipart	Comic	Watercolor	Average
SSD	SSD baseline	26.8	24.9	49.6	33.8
	WST-BSR [26]	35.7	26.8	49.9	37.5
	DT [11]	**38.0**	29.8	50.4	39.4
	SSD-RST	34.9	30.9	**53.1**	39.6
	Self-ADE	36.5	**31.9**	52.3	**40.2**
Faster R-CNN	Faster R-CNN baseline	26.4	18.1	42.8	29.1
	DivMatch [27]	**39.7**	**36.9**	**54.7**	**43.8**
	StrongWeak [28]	28.3	24.8	45.5	32.9
	Faster R-CNN-RST	34.3	32.0	52.0	39.5
	Self-ADE	35.0	22.4	52.0	36.5

Table 4.10: Summary of results of Self-ADE applied to SSD and Faster R-CNN on the VOC → AMDs benchmark. Comparison in the 100% target settings (DA)

		Foggy Cityscapes
Faster R-CNN	Faster R-CNN baseline	26.8
	DivMatch [27]	**34.2**
	StrongWeak [28]	28.8
	Faster R-CNN-RST	30.8
	Self-ADE	32.2

Table 4.11: Summary of results of Self-ADE Faster R-CNN on the Cityscapes → Foggy Cityscapes benchmark. Comparison in the 100% target settings (DA)

4.3.3 1% target

Domain Adaptation settings are not the settings in which the Self-ADE approach can show its real potential. In fact, if the target domain is known *a priori*, it is still a better idea to use DA specific methods, which can be more effective and are surely more efficient at test time. One-Sample Adaptation settings, on the contrary, describe exactly the situation in which an approach like Self-ADE is the only option available to obtain good results. To prove its effectiveness, in absence of other methods designed for these settings, we can try to adapt DA methods bringing them in our settings. More specifically we will show that DA methods do not perform well when the number of samples available for the adaptation step is limited. On the contrary Self-ADE performances do not change as it does not need those samples at all.

It might be argued that, by reducing the number of target samples used at training time, we are not really bringing DA methods in our settings. What we should really do is, in fact, to exploit one sample at a time to adapt the model and then to use this adapted model to perform an inference on the same sample. This is, indeed, exactly what Self-ADE does. However this type of experiment is not feasible as DA methods often need a long training to adapt the model to the target domain. This adaptation time usually does not depend on the number of samples used to perform the adaptation. As a result, an experiment that involves the adaptation of the model using one sample at a time in order to test on the same sample, if performed on an entire dataset (even for a small 1000 samples one), **could last months**. Furthermore, what these experiments would probably tell us, is that DA methods are not able to perform the adaptation step when only one sample is available. To prove this statement we performed experiments that should assess the effectiveness of DA methods when we decrease the number of samples available for the adaptation.

		Clipart	Comic	Watercolor	Average
SSD	SSD baseline	26.8	24.9	49.6	33.8
	DT [11]	30.7	30.3	50.3	37.1
	SSD-RST	26.1	25.2	50.9	34.1
	Self-ADE	**36.5**	**31.9**	**52.3**	**40.2**
Faster R-CNN	Faster R-CNN baseline	26.4	18.1	42.8	29.1
	DivMatch [27]	26.3	20.8	45.4	30.8
	StrongWeak [28]	26.4	21.0	42.0	29.8
	Faster R-CNN-RST	27.8	20.4	44.8	31.0
	Self-ADE	**35.0**	**22.4**	**52.0**	**36.5**

Table 4.12: Summary of results of Self-ADE applied to SSD and Faster R-CNN on the VOC → AMDs benchmark. Comparison in the 1% target settings (reduced target)

		Foggy Cityscapes
Faster R-CNN	Faster R-CNN baseline	26.8
	DivMatch [27]	**37.0**
	StrongWeak [28]	25.7
	Faster R-CNN-RST	30.4
	Self-ADE	32.2

Table 4.13: Summary of results of Self-ADE Faster R-CNN on the Cityscapes → Foggy Cityscapes benchmark. Comparison in the 1% target settings (DA)

The results in these settings clearly show that traditional DA methods see their performance fall when the number of target samples available at training time is low. In particular in the VOC → AMDs settings the performance of all methods are degraded to a point in which only a small adaptation occurs. W.r.t. these methods Self-ADE has better performance with a margin going from 3 to 5 mAP points. Only in the Cityscapes → Foggy Cityscapes benchmark one method, DivMatch, outperforms Self-ADE in these settings. This is probably due to the fact that the domain shift in this case is quite small (it is an artificial fog layer added on top of photos) and, as a consequence, the CycleGANs used by DivMatch do not need many samples to learn how to replicate it.

Performance decrease trend In order to understand which is the trend of the decrease in performance that involves DA methods when the number of target samples is reduced I decided to test some of the methods even with different sizes for the target samples splits. In particular I performed experiments for the VOC → AMDs benchmark not only with 100% and 1% of the data but also with 50%, 25% and 10% splits. I then averaged the results obtained on the 3 different AMDs and plotted them. The charts are in Figures 4.1 and 4.2, respectively for SSD based and Faster R-CNN based methods. In the same plots I included also the trend of the performance of Self-ADE: the performance is constant as a consequence of the fact that the target data is not used at all during the training.
What these charts tell us is that:

- all DA methods undergo a decrease in performance when the number of available samples for the adaptation is reduced;
- DA methods based on CycleGANs are more robust to a decrease in the size of the target training split w.r.t. other methods. This may mean that the CycleGANs need only a small

amount of samples to successfully learn the main characteristics of a domain or that even noisy translated samples may be useful for a network to adapt itself.

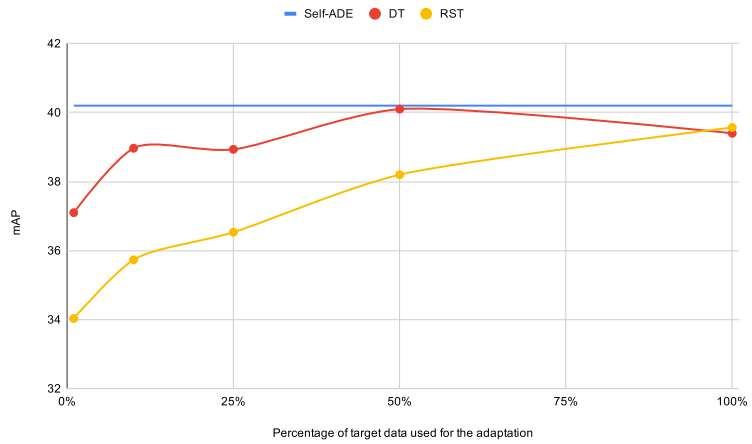


Figure 4.1: Trend of the mAP of various methods using SSD on VOC → AMDs w.r.t. the percentage of target training data used for the adaptation

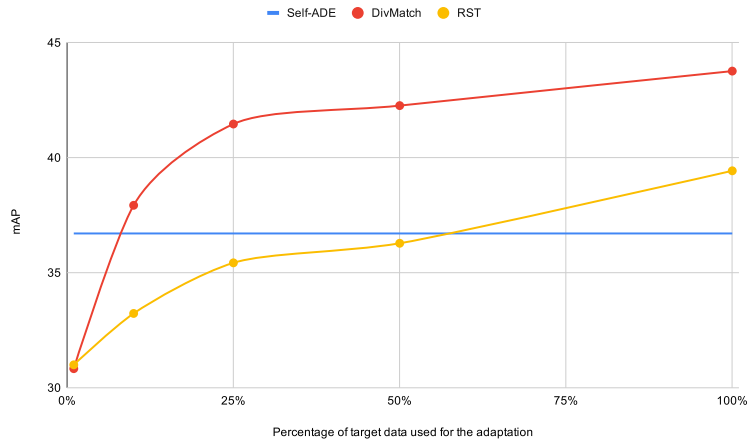


Figure 4.2: Trend of the mAP of various methods using Faster-CNN on VOC → AMDs w.r.t. the percentage of target training data used for the adaptation

4.3.4 Finetuning iterations and performance

The performance of Self-ADE depends on the number of iterations performed in the finetuning on each sample: when the model is being adapted more iterations mean a better adaptation and better detection results. We prove this fact by computing and plotting the performance of Self-ADE using some different numbers of finetuning iterations, but also by visualizing the bounding boxes of detected objects after various numbers of iterations for some sample images.

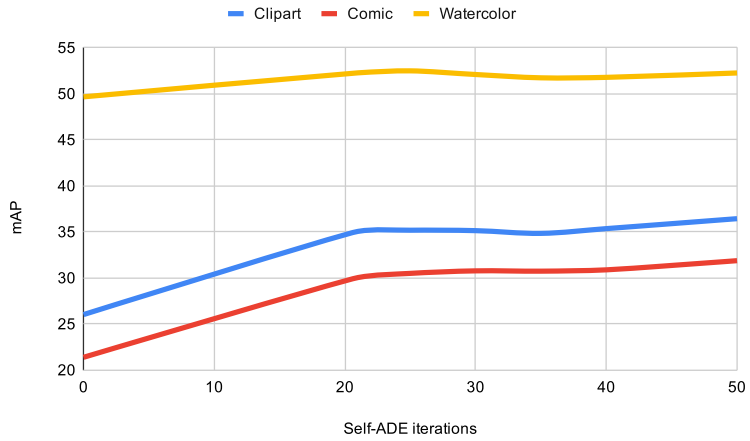


Figure 4.3: Trend of the mAP on VOC → AMDs with different iterations for SSD Self-ADE finetuning

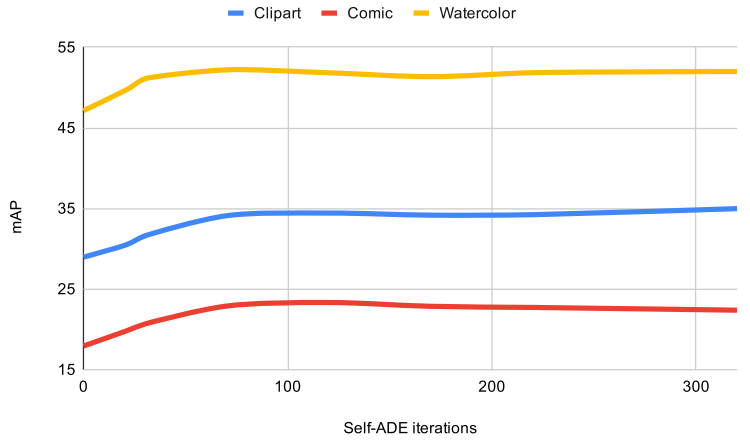
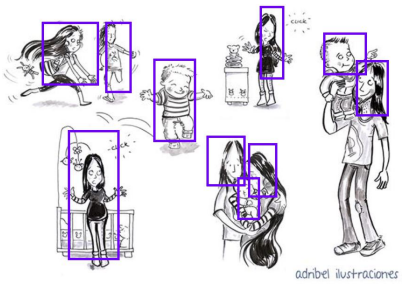


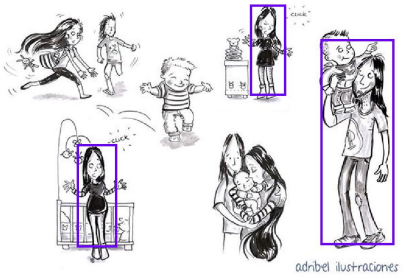
Figure 4.4: Trend of the mAP on VOC → AMDs with different iterations for Faster R-CNN Self-ADE finetuning

The charts in Figures 4.3 and 4.4 highlight the fact that the improvement in the results obtained with a higher number of iterations is greater for the first iterations and lower at the end. This means that, even if it could probably be possible to obtain better results by further increasing the number of iterations, this improvement would be small. As there is a trade off between the performances and the efficiency of the method it could be a better idea not to increase the number of iterations too much.



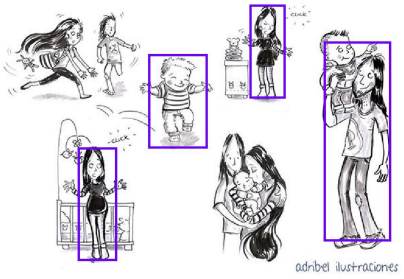
(a) Comic2k's image with GT bounding boxes

(b) Comic2k's image with GT bounding boxes



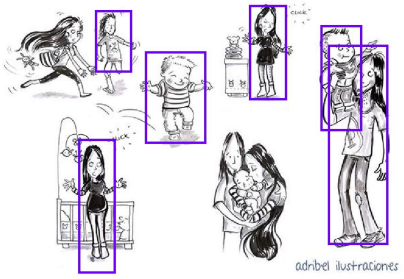
(c) Detections after 1 fine tuning iteration

(d) Detections after 1 fine tuning iteration



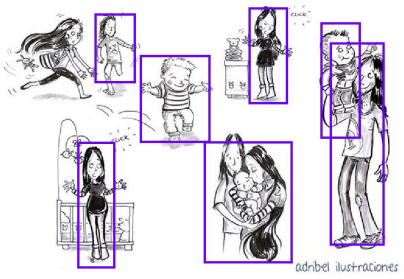
(e) Detections after 30 fine tuning iteration

(f) Detections after 30 fine tuning iteration



(g) Detections after 70 fine tuning iteration

(h) Detections after 70 fine tuning iteration



(i) Detections after 120 fine tuning iteration

(j) Detections after 120 fine tuning iteration

Figure 4.5: Example of situations in which an higher number of fine tuning iterations allows the Self-ADE Faster R-CNN implementation to correctly detect more objects

4.3.5 Final considerations

The obtained results clearly show that the introduction of a test time, self supervised based, adaptation strategy does not only allow to obtain good performances, but represents a strategy applicable in a great number of contexts and, as such, it represents a more general approach w.r.t. DA or DG ones. This method can, in fact, be applied:

- when the target domain is not known a priori;
- when each test sample comes from a potentially-different target domain;
- without performance degradation in the original source domain.

The possible practical applications of this kind of model are a lot. To give just one example let us consider a self driving car that uses cameras as a source of information on the environment. The images captured by these cameras are processed by the car in order to perform visual object detection (or, probably more often, *semantic segmentation*) to localize the possible obstacles. The object detection model used to perform this task should have good performances in any possible situation, for example when driving in sunny conditions inside a big city but also when traveling by night in a rainy countryside. This is a perfect example of situation in which there are various domains to deal with: each domain is represented by a combination of one weather condition, one driving location and possibly other factors. To train such a model there are a few possible alternatives:

- *Deep All strategy.* A great number of annotated samples for each possible domain is collected and then a deep model is trained on all the collected images. The performance of this model could be quite good, but only in the domains considered a priori. Furthermore the collection of a dataset of this kind would be really expensive;
- *Domain Generalization Strategy.* Like for the *Deep All* model there is the need for a large dataset spanning various domains, but the performance reached by the trained model (based on a DG method) could be better even for the domains not considered a priori;
- *Domain Adaptation Strategy.* In this case a large labeled dataset can be collected just for one single domain and smaller unlabeled datasets for the other ones. Then a number of different models should be trained using a DA procedure, one for each possible target domain. At test time a *domain discriminator* should be used to decide which of the trained models should be used in each condition. The performance of this kind of approach could be good, but it is clear that a new not previously seen domain could be difficult to deal with for the model and in any case the collection of the datasets could still be expensive;
- *One Sample Adaptation Strategy.* A large labeled dataset is needed only to train the first model. The data can come from one single domain or various domains, but no other data is needed. This means that there is no need to collect images from all the domains as the model should adapt successfully, at test time, to the sample it receives, whichever is the domain it belongs to. It has to be noticed that, for this type of application, Self-ADE is not ready yet as it is not a real time approach.

Chapter 5

Conclusions

We have investigated the effectiveness of the usage of a self supervised task as a method to obtain good cross domain visual object detection performance in various different settings. We have seen that this approach allows to obtain good results when applied on both single stage and two-stage detectors on various cross domain benchmarks and this demonstrates its generality. In particular **we reached state of the art** in:

- Domain Generalization and Domain Adaptation settings when using a single-stage detector;
- One-Sample Adaptation settings when using both single stage and two-stage detectors.

We have introduced **One-Sample Adaptation** as a new cross domain analysis research setting that is both more general and more practically useful than DG and DA and we presented an algorithm dedicated to this setting performing better than all current state of the art DA methods when adapted to the setting. It is the first algorithm able to adapt an object detector at test time to a new domain using one single sample and without the need to use target data at training time. As a consequence it represents a new category of algorithms with a clearly wider application scope in the context of cross-domain analysis.

Self-ADE has been presented in a paper **submitted for the 2020 edition** of the Conference on Computer Vision and Pattern Recognition (**CVPR**).

Possible **future works** in this context could develop following different strands: the investigation on self supervised tasks, in order to understand the features and the potential of each of them and the possibilities of integration in a single stage or two stage object detection pipeline as methods to obtain even better feature alignment across domains; the investigation on the possibilities to improve the efficiency of the Self-ADE method, necessary to bring it from a test-time adaptation method to a real-time ready approach; the investigation of the possibilities to use a self supervised task in an object detection pipeline with purposes different from the cross-domain analysis; the investigation of the possibilities of use of a *Self-ADE*-like approach in other computer vision's tasks.

Bibliography

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation", *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014. DOI: 10.1109/cvpr.2014.81.

[3] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition", *Int J Comput Vis*, vol. 104, no. 2, pp. 154–171, Sep. 2013, ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-013-0620-5.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-Cnn: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2577031.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN", in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector", *Lecture Notes in Computer Science*, pp. 21–37, 2016, ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[8] A. Torralba and A. A. Efros, "Unbiased Look at Dataset Bias", in *CVPR 2011*, Jun. 2011, pp. 1521–1528. DOI: 10.1109/CVPR.2011.5995347.

[9] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Deeper, Broader and Artier Domain Generalization", in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 5543–5551. DOI: 10.1109/ICCV.2017.591.

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge", *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[11] N. Inoue, R. Furuta, T. Yamasaki, and K. Aizawa, "Cross-Domain Weakly-Supervised Object Detection Through Progressive Domain Adaptation", in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 5001–5009. DOI: 10.1109/CVPR.2018.00525.

[12] X. Peng, B. Usman, K. Saito, N. Kaushik, J. Hoffman, and K. Saenko, "Syn2Real: A New Benchmark forSynthetic-to-Real Visual Domain Adaptation", Jun. 25, 2018. arXiv: 1806.09755 [cs]. [Online]. Available: http://arxiv.org/abs/1806.09755 (visited on 11/22/2019).

[13] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-Adversarial Training of Neural Networks", *Advances in Computer Vision and Pattern Recognition*, pp. 189–209, 2017, ISSN: 2191-6594. DOI: 10.1007/978-3-319-58347-1_10.

[14] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial Discriminative Domain Adaptation", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017. DOI: 10.1109/cvpr.2017.316.

[15] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks", in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2242–2251. DOI: 10.1109/ICCV.2017.244.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets", in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (visited on 11/25/2019).

[17] M. Ghifary, W. B. Kleijn, M. Zhang, and D. Balduzzi, "Domain Generalization for Object Recognition with Multi-Task Autoencoders", in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 2551–2559. DOI: 10.1109/ICCV.2015.293.

[18] K. Muandet, D. Balduzzi, and B. Schölkopf, "Domain Generalization via Invariant Feature Representation", *ICML*, Jan. 2013.

[19] M. Mancini, S. R. Bulo, B. Caputo, and E. Ricci, "Best Sources Forward: Domain Generalization through Source-Specific Nets", *2018 25th IEEE International Conference on Image Processing (ICIP)*, Oct. 2018. DOI: 10.1109/icip.2018.8451318.

[20] F. M. Carlucci, A. D'Innocente, S. Bucci, B. Caputo, and T. Tommasi, "Domain Generalization by Solving Jigsaw Puzzles", presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2229–2238. [Online]. Available: http://openaccess.thecvf.com/content_CVPR_2019/html/Carlucci_Domain_Generalization_by_Solving_Jigsaw_Puzzles_CVPR_2019_paper.html (visited on 11/25/2019).

[21] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Learning to Generalize: Meta-Learning for Domain Generalization", Oct. 10, 2017. arXiv: 1710.03463 [cs]. [Online]. Available: http://arxiv.org/abs/1710.03463 (visited on 10/29/2019).

[22] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised Visual Representation Learning by Context Prediction", in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1422–1430. DOI: 10.1109/ICCV.2015.167.

[23] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised Representation Learning by Predicting Image Rotations", presented at the International Conference on Learning Representations, Feb. 15, 2018. [Online]. Available: https://openreview.net/forum?id=S1v4N2l0- (visited on 11/25/2019).

[24] R. Zhang, P. Isola, and A. A. Efros, "Colorful Image Colorization", *Lecture Notes in Computer Science*, pp. 649–666, 2016, ISSN: 1611-3349. DOI: 10.1007/978-3-319-46487-9_40.

[25] M. Noroozi and P. Favaro, "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles", *Lecture Notes in Computer Science*, pp. 69–84, 2016, ISSN: 1611-3349. DOI: 10.1007/978-3-319-46466-4_5.

[26] S. Kim, J. Choi, T. Kim, and C. Kim, "Self-Training and Adversarial Background Regularization for Unsupervised Domain Adaptive One-Stage Object Detection", Sep. 2, 2019. arXiv: 1909.00597 [cs]. [Online]. Available: http://arxiv.org/abs/1909.00597 (visited on 09/13/2019).

[27] T. Kim, M. Jeong, S. Kim, S. Choi, and C. Kim, "Diversify and Match: A Domain Adaptive Representation Learning Paradigm for Object Detection", p. 10, 2019.

[28] K. Saito, Y. Ushiku, T. Harada, and K. Saenko, "Strong-Weak Distribution Alignment for Adaptive Object Detection", p. 10, 2019.

[29] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, p. 11.

[30] Q. Cai, Y. Pan, C.-W. Ngo, X. Tian, L. Duan, and T. Yao, "Exploring Object Relation in Mean Teacher for Cross-Domain Detection", p. 10,

[31] C. Sakaridis, D. Dai, and L. Van Gool, "Semantic Foggy Scene Understanding with Synthetic Data", Aug. 25, 2017. DOI: 10.1007/s11263-018-1072-8. arXiv: 1708.07819 [cs].

[32] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", in *International Conference on Learning Representations*, 2015.

[33] R. Girshick, "Fast R-CNN", in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks", *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 24, 2017, ISSN: 00010782. DOI: 10.1145/3065386.

[35] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part Based Models", p. 20, 2010.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", *Lecture Notes in Computer Science*, pp. 346–361, 2014, ISSN: 1611-3349. DOI: 10.1007/978-3-319-10578-9_23.

[37] J. Bisanz, G. L. Bisanz, and R. Kail, *Learning in Children: Progress in Cognitive Development Research*. New York: Springer-Verlag, 1983, ISBN: 978-1-4613-9501-0. [Online]. Available: https://www.springer.com/gp/book/9781461395010 (visited on 08/07/2019).

[38] C. Li, "High Quality, Fast, Modular Reference Implementation of SSD in PyTorch", 2018. [Online]. Available: https://github.com/lufficc/SSD.

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic Differentiation in PyTorch", in *NIPS Autodiff Workshop*, 2017.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", Dec. 10, 2015. arXiv: 1512.03385 [cs]. [Online]. Available: http://arxiv.org/abs/1512.03385 (visited on 11/18/2019).

[41] F. Massa and R. Girshick, "Maskrcnn-Benchmark: Fast, Modular Reference Implementation of Instance Segmentation and Object Detection Algorithms in PyTorch", 2018. [Online]. Available: `https://github.com/facebookresearch/maskrcnn-benchmark`.

Appendix A

Object detection metrics

Visual object detection has no real standard metrics used to compare model performances. This is due to the fact that the output of a detector is more complex than for example the output of an image classifier: in fact in this context a single image can contain more than one object and the model also needs to locate them. As a consequence popular object detection challenges have defined their own performance metrics and often provided reference code implementations to compute them.

The performance metrics used in this master thesis project are those defined by the PASCAL VOC Challenge ([10]), in particular the *Precision* $\times$ *Recall curve* and the *average precision* that I will now present in detail.

A.1 Preliminary definitions

In order to understand the metrics used it is necessary to have clear in mind the meaning of some fundamental concepts.

A.1.1 Intersection over Union

Often referred simply as IoU it is a cardinal notion for object detection in general as it is a measure of the overlap between 2 bounding boxes. It is used to understand how well a proposed bounding box match the corresponding ground truth. As the name suggests it is based on the measure of the surface of the intersection and the union of the two bounding boxes and it is computed like this:

$$IoU = \frac{Intersection's\ surface}{Union's\ surfare}$$

The IoU once computed is usually compared with a threshold to understand if a detection can be considered correct.

A.1.2 True Positive, False Positive, False Negative and True Negative

In the context of object detection these terms are used for:

- **True Positive** (TP): a correct detection, that is for which the IoU with a ground truth bounding box is greater or equal than a certain threshold;

- **False Positive** (FP): an incorrect detection, that is for which the IoU with a ground truth bounding box is lower than the threshold. We also consider as FP a prediction A for which $IoU_A \geq threshold$ w.r.t. a ground truth box if there is another prediction B for which $IoU_B \geq IoU_A$ w.r.t. the same ground truth box;
- **False Negative** (FN): a missing detection, that is a ground truth bounding box not detected;
- **True Negative** (TN): not applicable for the object detection task as it would correspond with a correct detection of the background that is not relevant.

The value of the threshold used to separate *True Positives* and *False Positives* can be chosen, usually it is one of: $\{0.5, 0.75, 0.95\}$.

A.1.3 Precision

We call *precision* the ability of a model to detect only the relevant objects and we measure this ability as the percentage of correct positive predictions:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{All\ detections}$$

A.1.4 Recall

The *recall* measures the ability of a model to detect all the objects, it corresponds with the percentage of the object detected:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{All\ ground\ truth\ boxes}$$

Both the *precision* and *recall* can be computed only considering one class at a time and as a result also the metrics that are based on these two measures can be computed only for one class at a time.

A.2 Metrics

A.2.1 *Precision* × *Recall* curve

It is a common metric both in visual object detections and in other machine learning contexts. We know that a good model should have both an high precision and an high recall simultaneously. At the same time we can also intuitively understand that the two measures are in a certain sense in opposition:

- efforts to increase the *recall* can also increase the chance to detect non-existing objects and as a result they can lead to a lower *precision*;
- efforts to increase the *precision* usually are directed towards a reduction of the FP but they risk to produce a reduction even of the TP and as a consequence of the *recall*.

For this reason we draw a curve that shows the level of *precision* for various levels of *recall*. The curve is drawn by varying the confidence threshold. This threshold is used to understand if a certain detection can be marked with a given class.

To be noticed that given that the number of instances in a certain dataset is limited the curve cannot be drawn as a continuous function but has to be approximated.

A.2.2 Average Precision

The *Precision* × *Recall* curve as a metric is not useful to compare the performance of two models. In fact, while it is clear that a good model should have this curve as similar as possible to the *Heaviside step function*, it can be not trivial to understand, between two curves, which is closest to this target.

For this reason a numeric metric could be a better choice and the *average precision* plays this role as it corresponds simply to a measure of the surface below the curve (for this reason this metric is often referred as AUC, that is *Area Under the Curve*).

Using the *average precision* (or AP) it is easy to compare the performance of two models:

- each model is tested on the same dataset: the *ap* is computed for each class;
- the average between the *ap* is computed and is referred as *mAP* (*mean Average precision*);
- the *mAPs* of the two models are compared.

Appendix B

CycleGANs and dataset sizes

In order to obtain the values I inserted in the tables and charts about the performances of the DA methods with reduced target datasets I had to run the code provided by the authors of those methods. This was necessary because the results provided in the original papers were obviously obtained by performing the experiments using the complete target sets.

Two of these methods, DT [11] and DivMatch, [27] are based on CycleGANs [15]. CycleGANs allow to transfer the domain of a certain dataset to an image belonging to a different domain. In other words: CycleGANs allow to translate images from one domain to another. To be trained a CycleGAN needs two different datasets that represent two different domains A and B: the training produces 2 generators, the first specialized in the translation from A to B and the second in the opposed one. It was really interesting to see how CycleGANs perform when the size of the dataset they can use at training time is reduced.
The two methods use CycleGANs in different ways:

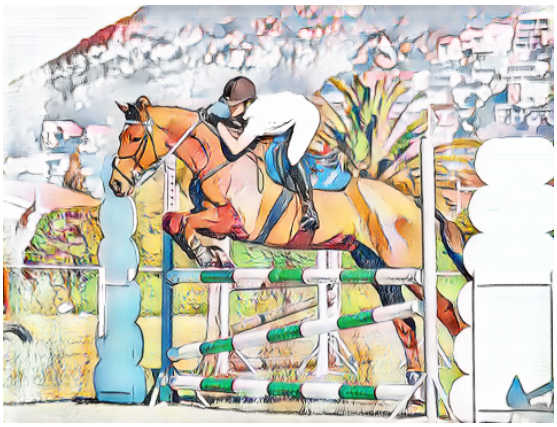
- DT uses a CycleGAN to translate the source dataset into the target domain. This way it is possible to perform a supervised training using the ground truth labels of the original images also for the translated images;
- DivMatch is in a certain way an evolution of DT: thanks to a process called Domain Diversification (DD), based on 3 different CycleGANs, 3 different versions of translated source domain datasets are produced.

What I noticed, thanks to the training of all these CycleGANs, is that:

- CycleGANs do not need very large datasets to learn the features of a domain. If the available samples are highly characteristic for a certain domain the CycleGANs are able to extract domain-specific features from them. However if the set is too small CycleGANs overfit and starts learning specific characteristics of these samples;
- when the samples available for a domain do not allow to understand clearly which are the characteristics of that domain the translation is weak;
- DA methods based on CycleGANs are more robust to strong data deprivation w.r.t. other methods probably because even if the translations produced by CycleGANs are noisy this noise is used as a source of regularization.

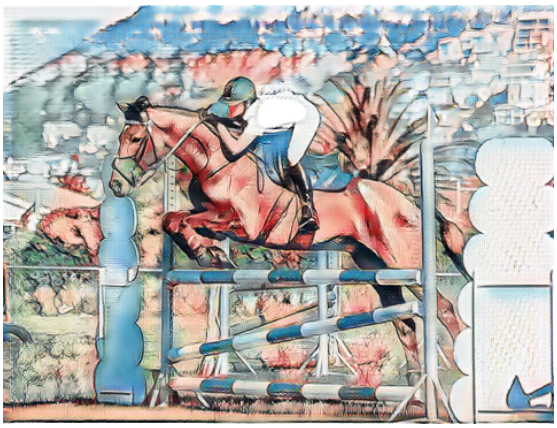
(a) Original Image of VOC2007 trainval set

(b) Translation to the *comic* domain using a model trained on 100% of comic's train split

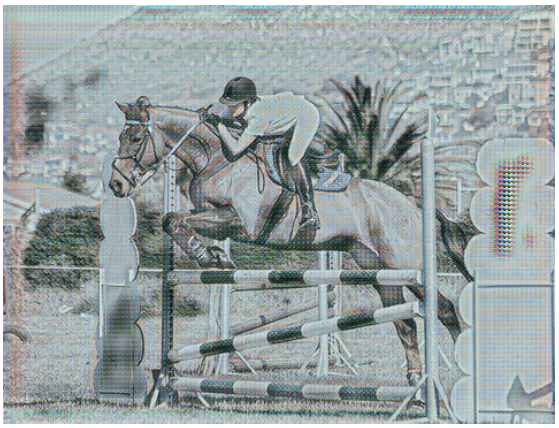


(c) Translation to the *comic* domain using a model trained on 50% of comic's train split

(d) Translation to the *comic* domain using a model trained on 25% of comic's train split



(e) Translation to the *comic* domain using a model trained on 10% of comic's train split



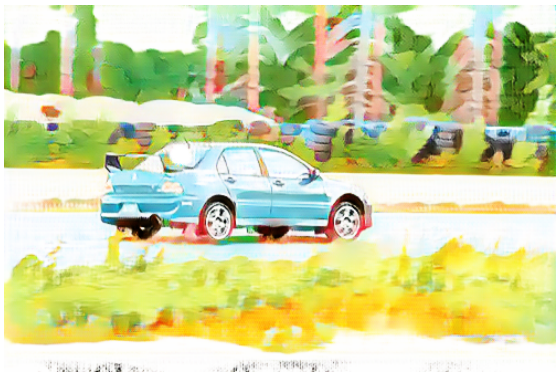
(f) Translation to the *comic* domain using a model trained on 1% of comic's train split

Figure B.1: Example of images translated from real world photos to the comic domain using CycleGANs trained on various data splits

(a) Original Image of VOC2007 trainval set

(b) Multi set

(c) Comic



(d) Clipart

(e) Watercolor

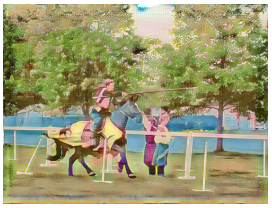
Figure B.2: CycleGANs and differences between domains. These examples clearly show that when the samples used to train a CycleGAN comes from a well defined domain the translation is well done. On the contrary when the domain is the union of various disjoint domains the translation is weak or not well defined

(a) Original Image of VOC2007 trainval set

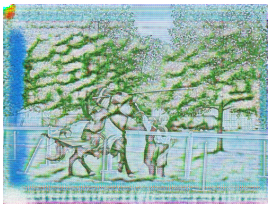
(b) Tranlation to comic domain using standard CycleGAN

(c) CPR transformation. CycleGAN trained on 100% train split of clipart dataset

(d) CPR transformation. CycleGAN trained on 50% train split of clipart dataset

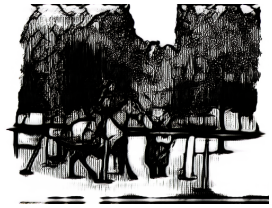


(e) CPR transformation. CycleGAN trained on 25% train split of clipart dataset



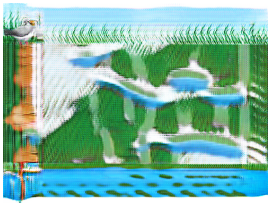
(f) CPR transformation. CycleGAN trained on 1% train split of clipart dataset

(g) CP transformation. CycleGAN trained on 100% train split of clipart dataset



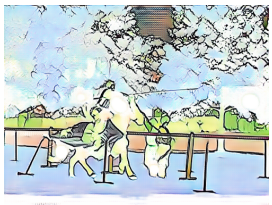
(h) CP transformation. CycleGAN trained on 50% train split of clipart dataset

(i) CP transformation. CycleGAN trained on 25% train split of clipart dataset



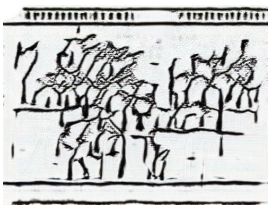
(j) CP transformation. CycleGAN trained on 1% train split of clipart dataset

(k) R transformation. CycleGAN trained on 100% train split of clipart dataset



(l) R transformation. CycleGAN trained on 50% train split of clipart dataset

(m) R transformation. CycleGAN trained on 25% train split of clipart dataset



(n) R transformation. CycleGAN trained on 1% train split of clipart dataset

Figure B.3: Example of images translated using DivMatch ([27]) strategy for Domain Diversification: each image of the source domain is translated into 3 different variants produced from the target domain.