POLITECNICO DI TORINO

DIPARTIMENTO DI AUTOMATICA E INFORMMATICA

Master Degree in Computer Engineering

Master Thesis

Meta-Learning for Cross-Domain One-Shot Object Detection

A new Approach

Advisor:
prof. Tatiana Tommasi

Co-Advisor:
dr. Francesco Cappio Borlino

Candidate:
Salvatore Polizzotto

Aprile 2021

Ai miei genitori,
A mia sorella,
A Kety

Chi la Dura La vince
Italian saying

Abstract

Computer Vision research today is focusing more and more on deep learning, because it is the machine learning paradigm that produces the best results. The main feature is that we can train an end-to-end network both to extract features from the data and to create models able to exploit them to solve certain tasks. In this way there is no need to manually extrapolate the right characteristics from the data to pass to the final model, thus overcoming the problems of shallow approaches and guaranteeing results with a much lower error rate. The main limitation is that deep networks have many parameters and therefore, in order to be trained, they need a large amount of labelled data, which is not always available. Another big problem is that many models are trained from scratch for certain tasks, using a fixed learning algorithm, and this means that they are unusable for other applications.

An alternative paradigm that is attracting a lot of interest in research is Meta-Learning, also known as *learning to learn*. Its goal is to make the network capable of modeling by itself the learning algorithm. In practice, the network is meta-trained on a large number of different sub tasks, producing a model ready to be finetuned on a small amount of data and able to generalize to any new task.

The focus of this master thesis work is to use Meta Learning for cross-domain analysis. The main challenge in this field is that models trained on a certain domain are mostly unusable on domains never seen before, a problem known as *domain shift*. More specifically, our objective is to create a visual object detector able to adapt on each test sample before performing predictions. To take advantage of unlabeled target samples it is possible to exploit the power of *self-supervised learning* by enriching the standard object-detector with the auxiliary objective of recognizing rotations applied to the objects. Since it does not need any manual annotation, this simple second task runs seamlessly on each single test image, helping the network to adapt to the style of the new instance.

The core step of this work is to code the logic of meta-learning within the self-supervised rotation task. We build over an existing method that deals with one-shot unsupervised domain adaptation and show that meta-learning allows to adapt more quickly to the various samples, producing good results in the inference phase. In this way we get the best of both worlds and establish the new state-of-the art in detection for social media monitoring and autonomous driving scenarios.

Contents

Introduction

Despite impressive progress of deep learning models over the last years, it is still an open challenge to reliably detect objects across visual domains. Consider for instance the case of a car and pedestrian detector trained on street images in sunny weather: it will have poor performance when applied on foggy weather or at night without a proper adaptation process.

One possible solution to these challenges can be obtained by exploiting the power of *self-supervised learning*[1]. In this framework, some part of the data information is withheld and the task of the network is to predict it back. In this way it is possible to learn a representation that carries good semantic or structural information even though the data was originally unlabeled. In [2] a standard Faster-RCNN model[3] was enriched with the auxiliary objective of recognizing the orientation {0°,90°,180°,270°} of the detected bounding box containing the object. Since it does not need any manual annotation, this simple second task runs seamlessly on each single test image, helping the network to adapt to the style of the new instance.

The proposed strategy shares its aim of learning from limited input information with another family of techniques, known as meta-learning[4]. It can be broadly defined as a class of machine learning models that *learn how to learn*: the network learns on few-shot meta-tasks during training, thus it is inherently prepared for the scenario that will be faced at test time. Meta-learning-based methods have already demonstrated their effectiveness in providing better data efficiency, improved ability in exploiting knowledge transfer, as well as in enhancing robustness and generalization of the training process.

This thesis work is dedicated to combine for the first time self-supervised and meta-learning for cross-domain one-shot object detection. The main contributions are:

- coding the logic of meta-learning within the self-supervised rotation task used by [2] to get the best of both worlds. Inspired by the Model-Agnostic Meta-Learning[5] method, we will reformulate it for the one-shot scenario;
- adding transformations of the visual appearance during the meta-training to more closely simulate the model adaptation phase, following the idea that train and test condition must match[6]. This allows us to make the network capable of adapt faster to the new target samples, reaching state-of-the-art result in different benchmarks;
- presenting two[7][8] of the most recent cross-domain detection models with the purpose of setting the baseline for our evaluations and analyzing the limits of these models in the one-shot scenario.

Chapter 1

Related works

1.1 Object Detection

Object detection is an important computer vision task that deals with detecting instances of visual objects of a certain class (such as humans, animals, or cars) in digital images. For better understanding, if we consider as input an image with one or more objects, such as a photograph, then the output of an object detection algorithm will be one or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each of them.

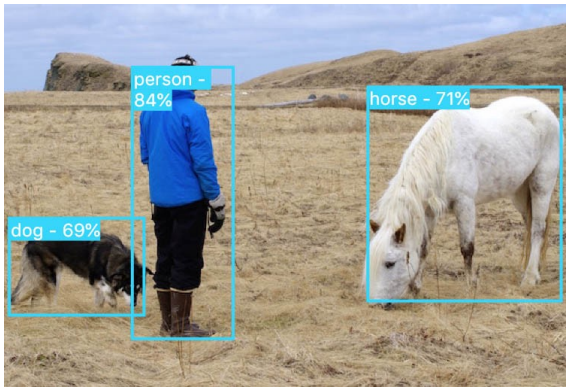


Figure 1.1: Example of a visual object detector output[9].

Thus, it tackles the problem of localizing multiple object categories in the image, using 2 more general tasks: the **detection** and **classification**. Depending on how these two more general tasks are combined, we can refer to two big families of object detection methods, as Jason Brownlee point out[10]:

- **The R-CNN model family**, which includes R-CNN[11], Fast R-CNN[12], Faster R-CNN[3] and Mask R-CNN[13], that uses a 2-stage framework: the first stage scans the image and generates proposals(areas likely to contain an object) and the second stage classifies the proposals and redefines bounding boxes. This bunch of methods achieve

near state-of-the-art results on object recognition tasks, but are not yet optimal for real-time applications.

- **The YOLO model family**, also called **single shot methods**, which includes YOLO[14] and its updated versions, that involves a single neural network that takes an image as input and predicts bounding boxes and class labels directly, without the needs of 2 stages. This family of methods suffers from a reduced accuracy in the localization of objects, when compared to the two-stage detectors, especially for smaller objects, but they are much faster at inference time allowing their use in real time applications.

Nowadays there are several detection applications, such as pedestrian detection, face detection, text detection, video-surveillance, tumor detection, etc. So, depending on what are the purposes of using object detection, we can choose the most suitable family and method.

1.1.1 Faster R-CNN

Faster R-CNN[3] is the first model to be entirely end-to-end and, as mentioned before, it is part of the R-CNN model family. The main innovation brought by Faster R-CNN is the introduction of the Region Proposal Network (**RPN**) which implements a very efficient region proposal system.

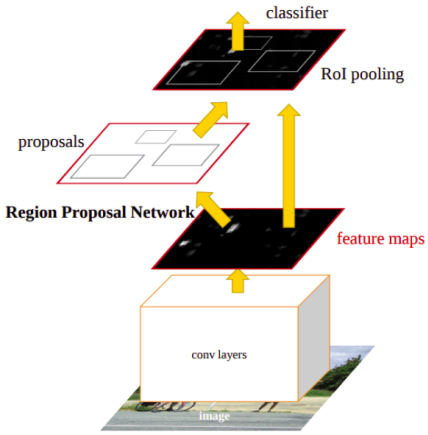


Figure 1.2: Structure of Faster R-CNN.[3]

Region Proposal Network(RPN). This network takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an abjectness score that represent how much is the probability that inside this proposal there is a generic object or not(background).

To generate region proposals, the authors[3] slide a small network over the convolution feature map output by the last shared *conv-layer* (Figure 1.2). So for each location of the input feature map, k anchor boxes are computed, with different sizes and aspect ratio. At the end of this process there are a huge amount of generated anchors. So the next step is to reduce them and take only a mini-batch of 256 total proposals.

First of all the authors assign a binary class label to each anchor: if the anchor has the highest Intersection-over-Union**(IoU)** overlap with a ground-truth box or the IoU is grater than 0.7, then this anchor is classify as positive. Instead they assign negative label if an anchor has an IoU lower than 0.3 with all ground-truth boxes. Positive means that the anchor contains an object inside, instead negative means that the anchor contains only background.

After this step, they choose 128 positive anchors and 128 negative ones and if there are fewer than 128 positive samples in an image, they pad the mini-batch with negative ones. This mini-batch is fed into two sibling fully connected layers:

- a **box-classification** layer that estimate the probability that inside an anchor there is an object or not, a 2-way classification problem.
- a **box-regression layer** that produce the offsets that need to be applied to the anchor to obtain a more accurate box position for the object inside.

The final multi-task loss used to train the **RPN** is:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*). \tag{1.1}$$

where, for each anchor inside the mini-batch, identified by i, they compute:

- a classification loss $L_{cls}(p_i, p_i^*)$, a log loss over 2 classes, where p_i represent the prediction of the **box-classification** layer and p_i^* is the ground-truth label that can assume 0 or 1, depend if the anchor is respectively positive or negative (as said before, this labels was decided in the creation of the mini-batch). This terms is used to improve the identification of any kind of object different from the background.
- a box regression loss $L_{reg}(t_i, t_i^*)$, a $smoothL_1$ loss function, applied on: t_i that represent the 4 coordinates of the bounding-box predicted by the **box-regression** layer and t_i^* that represent the ground-truth box associated with a positive anchor. In fact this term is multiple by p_i^*, meaning that the regression loss is only computed if the ground-truth of the anchor is positive, because the purpose is to achieve better result in localize the object and not the background.

λ is a balancing term, a weight used to regulate the importance of the classifier and the regressor.

This proposal generation process has shown huge advantages with respect to the other methods because produces bounding-boxes that are translation invariant, automatically detected with different size and aspect ratio starting only from an image and its feature map, and automatically learned during the training phase to localize better objects.

Fast R-CNN[12]. Now, the procedure adopted is exactly the same of the Fast R-CNN[12]. In fact the RPN is a new way of producing proposals different from the *selective search* used by the authors of the Fast R-CNN[12], because RPN starts working from the same feature map used by the Fast R-CNN to compute the final predictions. Thus, RPN and Fast R-CNN share the same *conv-layers* and in this way they can be integrated in an

end-to-end architecture and trained simultaneously, creating the Faster R-CNN (Figure 1.2).

The final output of the **RPN** network, during the forward pass, is a set of bounding boxes(4D coordinates vector) and for each of them the probabilities that there is an object inside. At this point a pooling level, called **RoI Pooling Layer**, for each proposal of the RPN, extracts a vector of a predetermined size from the features map outputs by the convolutional layers. RoI pooling works dividing the proposal in a fixed grid of sub-windows and then max-pooling the values in each sub-windows into the corresponding output vector and this is done independently on each feature map channel coming from the backbone(*conv-layers*).
Then, each vector is processed in two directions:

- the first direction is used to estimate the probabilities that in the vector(region) there is an object of the k-th class, for each of the K classes of recognizable objects in the dataset. This is the *classification head*.
- the second direction is use to produce four real numbers for each of the K recognizable classes, which represent the coordinates of the bounding boxes of the object referring to that class. This is the *regression head*.

The final multi-task loss used to train the **Fast R-CNN** is:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v), \tag{1.2}$$

where:

- $L_{cls}(p, u)$ is the classification loss for the ground-truth class u, often implemented with a softmax classifier, calculated on the prediction p of the *classification head*;
- $L_{loc}(t^u, v)$ is the regression loss or also called localization loss, implemented with a $smoothL_1$ loss and performed between the ground-truth 4D bounding box v for the class u and t^u predicted by the *regression head* for the class u. Note that $[u \geq 1]$ means that the localization loss is only computed on objects different from the background($u = 0$), because the main purpose of the object detector is to localize object inside an image and during training is important to manage these situations.
- The hyper-parameter λ controls the balance between the two component.

Joint Training. To train Faster R-CNN in an end-to-end fashion we need to combine the RPN and the Fast R-CNN component into a multi-task loss:

$$L(x, c, b) = (L_{cls}(x, c^*) + L_{reg}(x, b))_{RPN} + (L_{cls}(x, c) + L_{reg}(x, b))_{FAST\ R-CNN} \tag{1.3}$$

where c and b are respectively the ground-truth object category and bounding box coordinate of the objects and x is the input image. Note that the L_{cls} of the RPN use c^*, to highlight that it deal with a binary classification task, to separate foreground and background objects.

1.2 Cross-domain analysis

Recently, deep learning methods based on Convolutional Neural Networks (**CNN**s) have been largely used, because it is possible to create end-to-end models that allow to extract the right features and make predictions at the same time on a set of data, gaining a greater margin of success[15].
The main problem of these models is the need for a large amount of data during the training phase, because their structures are full of parameters that need to be learned, in order to extract knowledge from the training data and generalize to unseen ones during test time.

However, this situation opens the door to another issue: in most of the cases, data, used for learning the model's parameters, belong to the same visual domain, also called *source domain*, that is only a representation of the reality. If the source domain is biased, also the model will be biased and so essentially useless in domains which are different from the one represented by the original dataset, a problem known as *domain shift*.

The most straightforward solution to this challenge is to create specific models for each task to be performed, but this procedure is not completely feasible and does not reflect the learning process of a person, who solves different tasks under different conditions, trying to make his own strategy more context-independent.

Today, part of the research deals with these problems and the ongoing studies are divided into two broad branches : **domain generalization** and **domain adaptation**, both born with the aim of significantly reducing the performance degradation of the models due to *domain shift*, but using different settings and techniques.

1.2.1 Domain Adaptation

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. Most of the works that deal with the reduction of the domain gap between train and test data fall into this category, which consider part of the target data known a priori during the training phase. This is a strong assumption, because it is not always possible to obtain these data before the final prediction, but the possibility to access to these information guarantees better performances in the generalization process.

Two macro categories belong to this branch of research:

- **unsupervised domain adaptation**, where, during the training phase, the data coming from the target domain are managed without considering their labels. This means that we do not know what they contain, but we only know that they come from a different domain with respect to the source.
- **supervised domain adaptation**, where test data come with their labels during the training phase. In this case we can also exploit the information of each sample, which describes its content, according to the task to be performed.

In both cases, target data are used to guide the training procedure, in order to bridge the gap between source and target domain. There are three main form of alignment used nowadays:

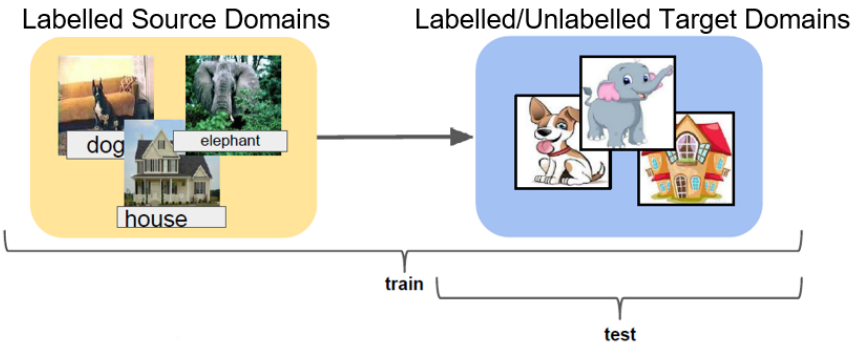


Figure 1.3: Domain Adaptation Setting[16].

- **feature-level alignment**, where the main objective is to close the gap between the features representation of source and target domain. One possible solution is to make the representation of the two domain indistinguishable to the model. In practise, the features representation of the two domains need to overlap, so that the model is pushed to learn features that are domain-invariant. There are several approaches that work in this way, like DANN[17].
- **pixel-level alignment**, where the main idea is to change the style of the source image, so that it appears to belong to the target domain. Thus, the strategy is to change the appearance such that it resemble the target style and the result is that the train is fully supervised, because the source domain comes always with labels. There are a bunch of methods that exploit generative networks, like GAN[18], to change the style of the source images and make this form of alignment[19].
- **distribution-level alignment**, where the main idea is to apply batch-normalization to source and target data, such that the two distributions map the same Gaussian[20].

1.2.2 Domain Generalization

Domain generalization is the most difficult case of domain adaptation, because the target domain is not know a priori and there are not information related to it. Thus, the challenge is to produce a model which is as good as possible to extract the most semantic and useful information from the training data (*source domains*), such that it can generalize better on targets (*target domains*) that it never see before.

Domain Generalization methods typically consider multiple source domains during the training phase, because the model need to generalize from the source data as much as possible. The main strategy is to use mechanisms that extract some domain-invariant representations from the data, in order to describe common aspect of known domains.

There are three main approaches that are used nowadays to accomplish the domain generalization task, as point out by *Da Li et al*[21]:

- **the first approach** is to train a model for each source domain. In this way, during test time, the similarity between the target domain and the source ones is computed and the model, whose domain is similar to the target one, is chosen[22].

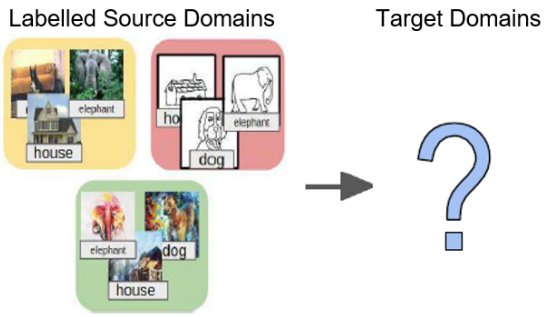


Figure 1.4: Domain Generalization Setting[16].

- **the second approach** is based on the consideration that each domain is composed by a domain-agnostic and a domain specific component. Thus, first we need to extrapolate, from each source domain, the domain-agnostic component and transfer it as a model that can work on new data[23].
- **the third approach** is to learn domain invariant features representation directly from the source domains, so that a model can perform well on a new unseen ones. In order to make the model independent from the *domain shift*, recent works add to the original training task a second one, self-supervised, like solve a jigsaw puzzle[24] or predict the rotation of the image[2]. In this way, they ask to the model to solve a problem that is not necessarily connected to the primary task but, at the end, with a multi-task procedure, the net is able to extract also semantic information from the source data, that are extremely useful to better generalize on the target domains.

In this master thesis project we will follow the third approach and we incorporate into an object detector a rotation task, that is used to learn a domain invariant features representation. This work is inspired by *D'Innocente et al.*[2], that already use this procedure in the detection task. As new entry, we add to this method a meta-learning approach, that share the same goal of the self-supervised task: make the network capable to generalize better and better. This challenge is made more difficult because we use only a single domain during the training procedure, and then we test the learned model on a variety of different datasets, that represent different domains.

1.3 Self-Supervised learning

Supervised learning describes a class of machine learning problems that involves using a model to learn a mapping between input examples and the target variables(labels). The core idea is to start from a dataset composed by input samples and their corresponding outputs/annotations, depending on the task, and then train a model to efficiently solve this mapping. The are some problems related to this approach:

- it is expensive to produce a new dataset for each new task
- in some areas it is hard to obtain annotation, *e.g.* medical data

- the annotation process is made by humans and therefore is prone to errors
- there are a vast numbers of unlabelled images/video that can be exploited

Why do not take advantage of the large amount of unlabelled data to help solving supervised tasks?

The Self-Supervision learning paradigm[1] help us to exploit this amount of information. It is a form of unsupervised learning where the data provides the supervision. In fact the basic concept is: each data are composed of different information and why do not hide some of them and ask to the network to predict it back? By doing so, the supervision is not provided a priori, but it is extrapolate from the data, with simple algorithms. In this way we can create "self-supervised tasks"[25] and force the network to learn what we really care about, *e.g.* a semantic representation, in order to solve tasks.

For instance, when we solve a puzzle, in most of the cases we do not need to understand what is the object inside, instead we need to follow the lines, match the colours. Here, we are trying to do something similar. If we give a task to an autonomous-system that is not necessarily connected to the object or to the style, but something based on correlation or connection between the similarity of part/patches, then the model will be able to extract from the data some high level information that will help to generalize better.

In general there are 2 approaches to exploit self-supervised tasks:

- the **first approach** uses the self-supervised task to pre-train the network, so that the model can be applied to a target task by using only a modest amount of labelled data. This procedure is similar to transfer learning, but a recent work has shown, with a thorough analysis, the potential of self-supervised learning as initialization step also starting from a single image[26].
- the **second approach** combines the self-supervised task with a supervised one in a multitask framework. Recent works show how this procedure improves adaptation and generalization performance[2][24]. Here we refer to the self-supervised task as *pretext task* or *auxiliary task*, because we are not really interested in solve it, but we want to push the model to learn more from it.

In this master thesis work, we follow the second approach and we combine an object detector with a self-supervised rotation classifier in an multitask architecture, where only the features extraction part is shared. In this way, the model not only learn characteristics related to the objects inside the source domain, but also semantic features that are common across-domains and therefore its ability to generalize should obtain better results.

There are different type of self-supervised tasks, like training a network to predict pixel colour starting from a monochrome input[27], where starting from an image we remove the colour and the output is pushed into the model that have to colorize it, or training a network to reconstruct the original image that was first divided in patches and then shuffled before entering in the model[24]. These examples are easy to integrate inside any model that deal with a supervised task.

1.3.1 Image rotation self-supervised task

The definition of this task is really simple and take inspiration from[28]. First of all we need to define a transformation function, that take as input an image and return a rotated version of it. The rotation degree is chosen randomly between 4 possible values: 0°, 90°, 180°, 270° and this random value is the ground truth of the input. Now, first each image is transformed in its rotated version and then is passed to a model that have to predict what is the rotation degree. Thus, it is a 4-way classification problem and it can be implemented with any type of deep neural network architecture. *Gidaris et al.* [28] emphasize that, despite the simplicity of this task, it successfully forces the model trained on it to learn semantic features that are useful for a variety of visual perception tasks, like object detection.

1.4 Meta-Learning

Contemporary machine learning models are typically trained from scratch for a specific task, using a fixed learning algorithm designed by hand. Deep learning-based approaches have substituted the *shallow* ones 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. However, there are clear limitations[29]:

- DNNs need a lot of data to guarantee better performance and the most powerful models typically require large amounts of labelled samples, that are heavy to obtain;
- DNNs need huge amount of computational resources;
- in most of the cases, models are only specific to a given task and it is difficult to reuse them in other situations, also similar ones.

Thus, what if we do not have large datasets? What if we want a general purpose AI system in the real world?[30] A solution to those questions is to implement a new paradigm during the learning procedure, based on the idea that we need to learn priors from previous experience and exploit it on new unseen tasks. One way to follow this approach is through Meta Learning, or commonly known as *learning to learn*[31].

1.4.1 Problem Definition

Meta-learning aims to improve performance by learning *how to learn*. In particular, the vision is to learn a general purpose learning algorithm that can generalize across tasks and ideally enable each new task to be learned better than the last. In fact, the meta-learning framework learns on few-shot meta-tasks during training, so that it is inherently prepared for the scenario that will be faced at test time, *i.e.* small amount of data available to perform a new task.

Thus, we usually assume access to a set of dataset $D_{meta-train} = \{(D_1^{tr}, D_1^{ts}), ..., (D_n^{tr}, D_n^{ts})\}$ that represent our **n** few-shot meta-tasks. These tasks are divided in train and test set and do not exactly belong to the final tasks that we want to accomplish, but something structured related. We use them, during the meta-training procedure, to learn θ^*, representing

our global meta-learning parameters that will contains all the information that we need to solve new tasks:

$$\theta^* = \arg\max_{\theta} \log p(\theta|D_{meta-train}), \tag{1.4}$$

this is the *prior* extracted from $D_{meta-train}$ and whatever useful statistics are inside it. Thus, the meta learning problem is: pull out the right θ^*, starting from the initial model parameters θ and our meta-training data, so that it contains everything we need to know to efficiently solve new tasks. The Equation 1.4 represent the **outer loop optimization**.

Therefore, to obtain θ^*, we need to take advantage of all the few-show meta tasks present in $D_{meta-train}$. How? Starting from the initial model parameters θ, we have to finetune it on the training set of each meta-task and produce:

$$\phi^* = \arg\max_{\phi} \log p(\phi|D^{tr}, \theta) \tag{1.5}$$

representing the set of parameter of the specific task, reached through a meta-optimization problem from θ, called **inner loop optimization**. The key idea is to use ϕ^* as model parameters to assess how the optimization of θ is good for classify new unseen data points, represented by the test part of each meta-task (Figure 1.5). After this step on all the meta-tasks, the **outer loop optimization** is performed on the initial model parameter θ.

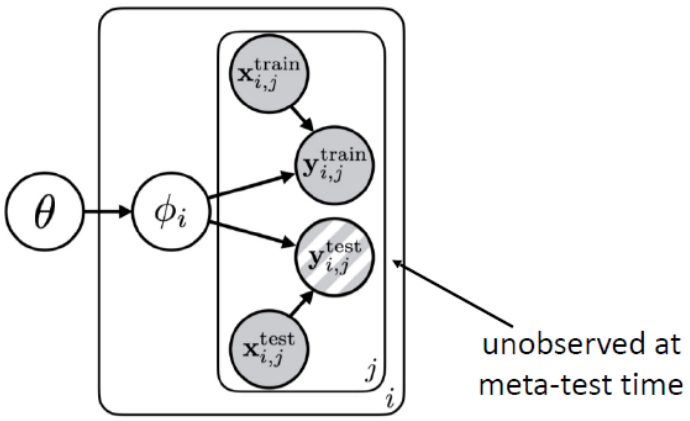


Figure 1.5: For every meta-task, ϕ_i with x_{train} determines y_{train} and x_{test} together with ϕ_i determine y_{test}, that is observed during meta-training and not during meta-test. In fact, the meta-training phase trains on a collections of datasets, like $D_{meta-train}$, instead the meta-test phase is once we have done meta-training and we want to adapt the model to a training set of a new task.

Depending on the way in which ϕ^* is obtained, there are different approach to meta-learning:

- **optimization-based approach**, where the goal is to acquire tasks specific parameters ϕ during an optimization procedure that depends both on the training data and on the meta-parameter θ, like **MAML**[5].

- **black-box based approach**, that trains a neural network to represent ϕ.
- **non-parametric based approach**, that uses method that are non parameterized by ϕ to obtain better performances on new tasks, like Siamese network[32].

To summarise, in meta-learning the goal of the trained model is to quickly learn a new task, from a small amount of data and training iterations during meta-test, after it was trained on a large number of different tasks, during meta-training, with the aim to quickly adapt to them. As explain by *Vinyals et al.*: "*our training procedure is based on a simple machine learning principle: test and train conditions must match*[6]". Thanks to the great capability to generalize and quickly adapt to unseen task, meta-learning can be applied in few-shot image recognition, human motion and pose prediction, domain adaptation/generalization, reinforcement learning and in all the most famous task of computer vision like image classification, detection, etc.

1.4.2 Model-Agnostic Meta-Learning (MAML)

This thesis work is inspired by one of the most popular optimization-based approaches to meta-learning, MAML[5]. As explained by the authors, the key idea underlying this method is to train the model's initial parameters such that the model has maximal performance on a new task, after the parameters have been updated through one or more gradient steps computed with a small amount of data from that new task.

Thus, the model, during the meta-training phase, is trained on a set of tasks, drawn from a distribution of tasks $p(T)$,with the purpose of quickly adapt to them. In this way, the prior knowledge acquired during meta-training can be used as initialization of the model at meta-test time, in order to finetune the parameters on new tasks. In fact, as the authors pointed out, if the model learn an internal representation suitable to many tasks, then simply fine-tuning slightly these parameters on new tasks allows the adaptation to happen in the right space for *fast learning*.

Formally, we consider a model parametrized by θ and a set of datasets $D_{meta-train} = \{(D_1^{tr}, D_1^{ts}), ..., (D_n^{tr}, D_n^{ts})\}$, where each of them represents a meta-task sampled from $p(T)$. At each iteration of the meta-training procedure, the authors sample a batch of tasks from $D_{meta-train}$ and for each training-set of the sampled meta-tasks they obtain:

$$\phi_i^* = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, D_i^{tr}) \tag{1.6}$$

that is computed using one or more gradient descent updates starting from the model with parameters θ. Thus, this procedure will produce different versions of ϕ^*, each one representing the fine-tuning steps performed on the corresponding training set of each meta-task, starting always from the model with parameters θ (Figure 1.6). This is called **inner-loop optimization**.

After this loop, the idea is to improve the initial model parameters by considering how the test error on new data changes because it depends on the different ϕ^* reached from the initial model parameters θ.
The **meta-objective** is:

$$\min_\theta \sum_i \mathcal{L}(\phi_i^*, D_i^{ts}) = \sum_i \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, D_i^{tr}), D_i^{ts}), \tag{1.7}$$

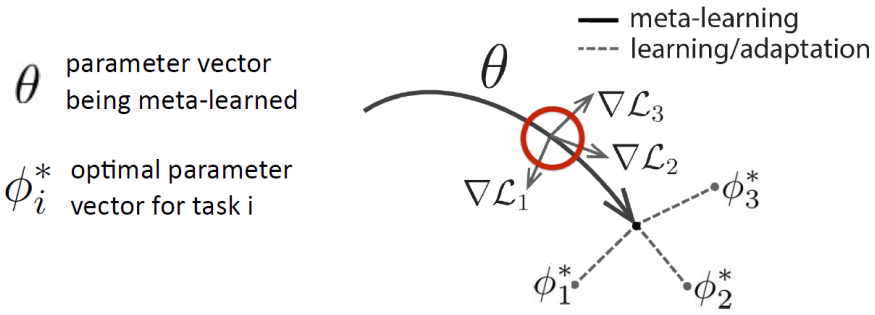


Figure 1.6: Fine-tuning steps, starting from the model's parameters θ and reaching different ϕ^*, one for each sampled meta-task[5].

where the test error, obtained on each test set of the sampled meta-tasks, serves as the training error of the meta-learning process. In practise, the authors want to verify how good are the fine-tuning steps on different tasks starting from the model parameters θ, that is the situation that the model will face at meta-test time when it deal with new unseen tasks. In this way the model parameters θ can quickly adapt to different tasks.

At this point the **meta-optimization** across tasks is performed via stochastic gradient descent (SGD) and the model parameters θ are update as follows:

$$\theta^* = \theta - \beta\nabla_\theta \sum_i \mathcal{L}(\phi_i^*, D_i^{ts}) \tag{1.8}$$

Note that, the meta-optimization is performed over the model parameters θ, called **outer loop optimization**, whereas the meta-objective is computed using the updated model parameters ϕ^*. This is the novelty of this method, that aims to optimize the model parameters such that one or a small number of gradient steps on a new task will produce maximally effective behavior on that task. Generally, tasks used for meta-testing are held out during meta-training.

To sum-up, **MAML** produce a weights initialization of the model during the meta-training phase, creating an internal representation that is suitable to many tasks. In this way, fine-tuning, with any amount of data and any number of gradient step, on a new tasks can produce good results.A simple example of the application of MAML is when we train a model to classify different objects, but then we want to use it to classify samples that belong to classes never seen before and we can consider only few samples to fine-tuning the model. The main advantages of the this algorithm are:

- it is task and model agnostic and this means that it can be applied to any distribution of tasks and to any model that is trained with a gradient descent procedure;
- it does not introduce any other additional parameters to learn during training;
- it optimize the model such that it is able to adapt to new tasks with only few examples;

- it is a new form of initialisation of the model parameters, especially when tasks are poor of labelled samples;
- it prepare the model to the situation that it will faced at meta-test time.

The major draw-back of **MAML**[5] is the calculation of the second derivative during the back-propagation on the meta-objective (see Equation 1.7), that slows down the algorithm and can exhibit instability during training. For this reason, the authors apply a first order approximation, imposing $\frac{d\phi_i}{d\theta}$ as identity in the back-propagation. Thanks to this change, they achieve a faster training procedure without losing in performance.

The authors show state-of-the-art results in few-shot classification and regression. The main objective of this master thesis work is to apply the ideas of *Finn et al.*[5] also in one-shot unsupervised cross-domain detection[2], where an object detector is trained only on one single domain and tested on different ones. More specifically, the idea is to prepare the model to the situation that it will face at test-time, *i.e.* starting from one target sample of an unseen domain, the model should be able to obtain good predictions after fine-tuning steps performed on the target image itself by the self-supervised task. The complete architecture of the **OSHOT** algorithm[2] will be presented in the next section.

Chapter 2

Method

2.1 Introduction

This master thesis work starts from excellent results obtained by *D'Innocente et al.*[2] in cross-domain analysis. Their goal was to show how the addition of a self-supervised task, specifically the classification of the rotation of an image, is important in the training phase of an object-detector to allow it to extrapolate, from the data, knowledge not only related to the visual appearance and to the objects inside, but also to intrinsic characteristics contained in any type of image, *e.g.* semantic information that does not depend on the source domain, but are common across domains. By doing so, they obtained a model, called **OSHOT**, capable of generalizing well on domains never seen during the training phase, exploiting a procedure in the adaptation phase called **One-Sample Adaptation**, which allowed them to obtain results that far exceed those obtained by other methods on the same settings.

Even if OSHOT shows progress in cross-domain analysis, the problem always remains the same: a model trained on a dataset, belonging to a single domain, will always lose performance if used on domains never seen before. *D'Innocente et al.*[2] reduce this *domain gap* even more with their technique, but the results are not yet comparable with those obtained in classification[33].

For this reason, the objective of this thesis work is to further enhance the **OSHOT** by revolutionizing its way of training, in order to make it as close as possible to the situation that the model have to face during the adaptation phase. In practice, we want to make the algorithm capable of *learning how to adapt* and **Meta-learning** can help us in this process.

Meta-learning arises from researches and discoveries conducted in the twentieth century, but only today a ferocious number of scientific works are adopting its fundamental principles. The innovation that fascinates many of the researchers is the ability to make a model capable of learning what is the best strategy to perform certain tasks, the so-called *learning to learn*, learning the "*algorithm*" to be performed. This approach differs greatly from the classic training paradigm adopted up to now, where the main objective is to finetuning the model parameters to obtain the most performing configuration.

The basic principle of the meta-learning is based on the idea that a machine learning model have to gain experience over multiple learning episodes – often covering a distribution of related tasks – and use this experience to improve its future learning performance[4]. Thus, the goal is to learn prior from previous experience such that lead to efficient downstream learning[30]. This means that during the meta-training procedure we need a set of datasets, that represents our prior that we want to learn and then use the learned *prior* as initialization for new tasks. In this way, the final model should be capable of generalize better, especially when there are few target samples available.

In this chapter first we will introduce the OSHOT and then we will discuss how to reformulate it in order to follow the meta-learning principles. Specifically, we have been inspired by **MAML**[5], a meta-learning method increasingly used today because it is model-agnostic and the only constraint is that the model, on which it is applied, is trained with a gradient descent procedure. Although OSHOT meets this constraint, there are other type of problems in the integration process that will be discussed also in the following sections.

2.2 One-Shot Unsupervised Cross-Domain Detection

In this section we will explain the method where we apply the MAML. It was created by professor Tommasi and researcher Cappio Borlino together with other professors/researchers of the Polytechnic of Turin. Below is reported the basic idea and the mathematical formulation of the problem, which refers to the following article[2].

Focus is on cross-domain object detection when only one target sample is available for adaptation, without any form of supervision. The authors propose an object detection method able to adapt from one target image, *i.e.* one-shot, that comes from a different domain with respect to the one used during the pre-training phase.

Specifically, they build a multi-task deep architecture that adapts across domains by leveraging over a self-supervised task, also known as *pretext task*. This auxiliary knowledge is further guided by a cross-task pseudo-labelling procedure, that injects the locality specific of object detection into self supervised learning.

Their strategy is to train the parameters of a detection model such that it can be ready to get the maximal performance on a single unsupervised sample coming from a new domain, after few gradient update steps on it.

The designed algorithm, called **OSHOT**, include:

- an initial **pre-training phase**, where they extend a standard deep detection model, Faster R-CNN[3], adding an image rotation classifier, a specific type of a self-supervised task. They implement this strategy to learn a representation that exploits inherent data information, because in this phase the only available data are that coming from the source domain. In fact the challenge is to better generalize from a single domain during the pre-training phase, such that the network is able to deal with sample coming from different distributions during the adaptation phase;

- a following **adaptation stage**, also called **one-sample adaptation phase**, where the backbone features are updated on the single target sample, by further optimization of the rotation task. In this phase, they exploit the *cross-task pseudo-labelling*

procedure to focus the auxiliary task on the local object context. In this way there is no need to access the source data while adapting on the target samples.

2.2.1 Pre-training phase

In this stage authors extend Faster R-CNN to include the image rotation recognition task and they train the network to optimize the following multi-task objective:

$$\arg\min_{\theta_f,\theta_d,\theta_r} \sum_{i=1}^{N} \mathcal{L}_d(G_d(G_f(x_i^s|\theta_f)|\theta_d), y_i^s) + \lambda \sum_{j=1}^{M} \mathcal{L}_r(G_r(G_f(R(x^s)_j|\theta_f)|\theta_r), q_j^s). \tag{2.1}$$

$\mathcal{L}_d$ is the detection loss of the Faster R-CNN and it is composed by G_f, parametrized by θ_f, that represent the feature extraction model(backbone), that transform any input image, x_i^s, into its feature map. This feature map is the used by G_d, parametrized by θ_d, to predict the bounding boxes and class label of each of the objects inside the image. The loss is calculated considering this prediction and the real label of the source image, represented by y_i^s. G_d is the union of the RPN and ROI-pooling of the Faster R-CNN, since both of them elaborate on the feature maps produced by the convolutional block and influence each other in the final optimization of the multi-task objective function.

$\mathcal{L}_r$ is the cross-entropy loss of the auxiliary rotation classifier and it is composed by the same feature extraction model, G_f parametrized by θ_f, that now transforms the rotated version of the same input image, called $R(x^s)_j$, into its feature map, because the main purpose of the auxiliary task is to predict the rotation of this image, which is represented by q_j^s, known a priori and chosen randomly between 4 possible value of rotations. G_r is the auxiliary rotation classifier, parametrized by θ_r, and it is used to predict the rotation not of the entire image, but it is created such that it can exploit the ground truth location of each object inside the original source image x_i^s, following a *boxcrop* operation. In fact starting from the features map coming from G_f and the bounding box of the source image itself, G_r extracts, one by one, only those features that are related to the objects that are really inside the image, then re-scales the features dimension (pooling step) and predicts the rotation of these objects.

In this way, as the authors say, the network is encouraged to focus only on the object orientation, without introducing noisy information from the background and provide better results with respect to the whole image option. In fact, if the rotation task is executed on the whole image, only simple elements, like the position of the sky, are enough to understand the orientation, instead working on objects-crops is more complicated because the network need to exploit information coming only from the object itself.

Implementation details

OSHOT is based on the Faster R-CNN that, as explained in the related works, uses the RPN to generate proposals for the Fast R-CNN. The number of generated proposals is huge and we need a lot of memory, especially GPU memory, to be able to support this computation. For this reason, the authors of the Faster R-CNN set the batch size equal to one, which means that, at each iteration of the training phase, the network manage only one image at a time. OSHOT has to deal not only with the object detector, but also with

the rotation classifier. Thus, the amount of requested memory is more than the one used in the Faster R-CNN and the resources available are not enough to have a batch size greater than one, especially in terms of GPU memory. To sum-up, OSHOT is trained with batch size equal to one and with samples coming from a single domain, that means belonging to the same data distribution.

2.2.2 Adaptation phase

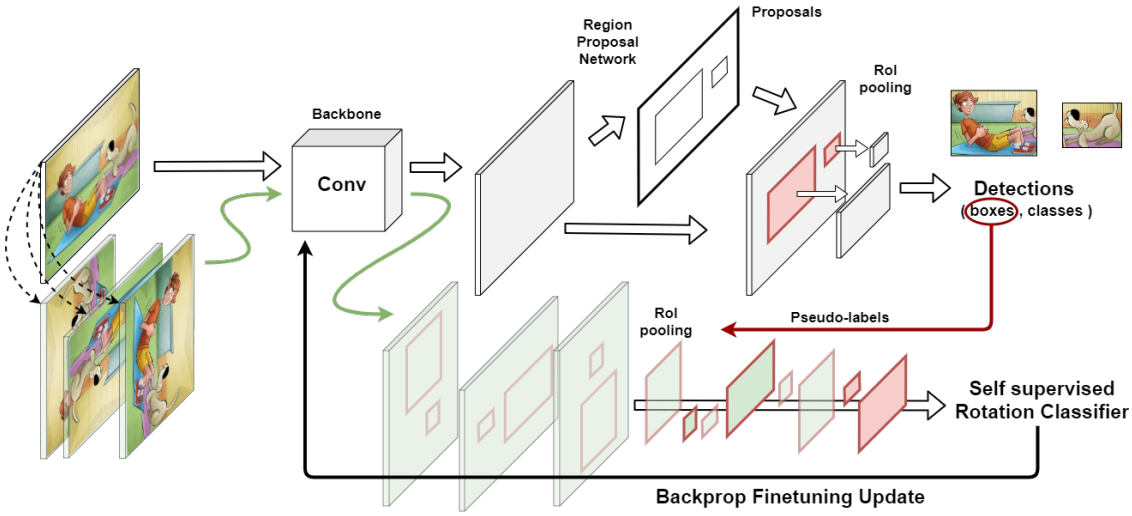


Figure 2.1: Visualization of the OSHOT adaptive phase with cross-task pseudo-labeling[2]

.

After the pre-training phase, the backbone of the model, G_f parametrized by θ_f, is fine-tuned by iteratively solving a self-supervised task on each single target image x^t. The number of finetuning steps starts from 0 up to 30, that correspond to the number of gradient steps computed on the backbone before producing the final prediction on x^t, made by the detector G_d. This is the mathematical formulation of the finetuning step:

$$\arg\min_{\theta_f,\theta_r} \mathcal{L}_r(G_r(G_f(R(x^t)|\theta_f)|\theta_r), q^t) \tag{2.2}$$

As explained by the authors, this process involves only G_f and G_r, while the detector G_d remain unchanged. In fact the idea is: instead of reusing the pseudo-labels produced by the pre-training model on the target to update the detector, they exploit them for the self-supervised rotation classifier G_r, following a *cross-task pseudo labelling* procedure.

In practise, they start from the $(\theta_f, \theta_r, \theta_d)$ model parameters of the pre-training stage and pass the x^t target image to the backbone G_f. After that, the features map is passed to the detector G_d that predicts y^t, *i.e.* a set of boxes and class labels, where the authors take only the bounding box detected on x^t, called *pseudo-labels*.

At the same time a random rotated version of x^t, called $R(x^t)$, is passed to the backbone and then G_r will consider both the features map produced by the backbone and the pseudo-labels produce by G_d. Thus, G_r works in the same way of the pre-training phase, but on

bounding boxes produced as inference of the detector on the target image, because in this case there are not ground truth boxes. This is the reason why is called *cross-task pseudo labelling* procedure (Figure 2.1).

After the rotation prediction of G_r, the authors calculate the loss (Eq. 2.2) and only the backbone is updated accordingly. This is the fine-tuning step. After γ steps, the inner features of the backbone are represented by θ_f^* and the detection prediction on x^t is obtained by:

$$y^{t*} = G_d(G_f(x^t|\theta_f^*)|\theta_d). \tag{2.3}$$

There are two things to point out in this procedure:

1. the adaptation phase is carried out on one target image at a time, therefore each sample could come from a different domain and the network should be ready to adapt independently;

2. after each adaptation phase on a single target image, the authors restored the backbone's parameters to those obtained after the pre-training phase, *i.e.* θ_f, to prevent that the network bias its internal representation to a specific domain and loses its generalization capability achieved during training.

As we can understand from this procedure, the authors create a new way to exploit the characteristics of the target samples before making the final prediction, having no prior knowledge about the target domains during the training phase. However, note that, during pre-training, the authors use the rotation task to force the network to learn domain-invariant features.

To summarize, the authors introduced:

- a new setting, called **One-Shot Unsupervised Cross-Domain Detection** (*One-Sample Adaptation* setting shortly), a scenario where the target domain can change from sample to sample and the adaptation can be performed one image at the time;

- a new algorithm, called **OSHOT**, able to perform one-shot unsupervised cross-domain adaptation, achieving state-of-the-art results in different benchmarks[2].

2.3 META OSHOT

In this section we show how to pass from the algorithm just described above, **OSHOT**, to a version that broadly follows the ideas proposed by *Finn et al.*[5], described extensively in the related works. In practise, we want to implement the meta-learning principles of MAML in order to make the network capable of autonomously *learning how to adapt* to the target samples.

2.3.1 Ideas and related problems

OSHOT has achieved excellent results in cross-domain analysis, now we want to further increase its performance. In order to accomplish this task, most of the inspiration comes from a popular optimization-based meta-learning algorithm, the so called **MAML**. The

interesting thing about this method is that it wants to simulate in training what will happen in test, because it is based on the idea that train and test conditions must match[6].

MAML[5] and OSHOT[2] are explicitly trained such that few gradient steps on new data produce good generalization performance. MAML, unlike OSHOT, uses a training methodology based on the match between train and test condition. Thus, why not use the idea of MAML and perfectly simulate the adaptation phase during the pre-training of the OSHOT?

The algorithm of *Finn et al.*[5] is model-agnostic and can be applied to any model that is trained with a gradient descent procedure. Therefore, the integration process inside the OSHOT is simple, because it already uses a gradient descend procedure in the pre-training phase. On the other hand, to ensure the effectiveness of MAML, other features are also needed, such as:

- a set of meta-task, each with a train and test part, structured related to the tasks that the model have to face during the meta-test phase;
- at each iteration, the **inner-loop optimization** need to be performed on a sample of meta-tasks, considering only the training set of them;
- instead the **meta-objective** is calculated on the test part of each sampled meta-task, in the same iteration;
- the algorithm was designed for classification and regression problems and meta-tasks are related to these fields.

All these characteristics are not present in OSHOT because, as already mentioned, in the pre-training phase it uses only data coming from a single domain and at each iteration, for computational problems, it can consider only one image. Furthermore, our task is to locate objects within the data, which is more complex than simple classification.

A first idea to overcome these problems is to create object-detection meta-tasks for each image of the pre-training phase. The problem with this approach is that each sample contains objects that are different in number and in category and therefore, to create tasks in a reasonable way, we should analyze the whole dataset. Moreover, even if we are able to obtain tasks, then we need to split each of them in train and test, which is even more difficult, due to the different number of instances in each sample. Summing up the discussion, this path is not feasible also because, in this way, the network does not *learn how to adapt*, which is our objective.

The main component that allows **OSHOT** to reach a certain degree of generalization is the rotation task. It is used, in the training phase, to ensure that the network is able to extract, from the data, not only characteristics related to the domain on which it is trained, but also inherent information that are independent across domains. Instead, it is used in the adaptation phase to make the parameters, learned during training, more consistent to the target instance that will be analyzed.

Therefore, why not use the rotation task as a meta-task to be performed in the **inner-loop optimization** of *Finn et al.*[5]? In this way, at each iteration, the image will be used to perform some fine-tuning steps on this meta-task, in order to prepare the backbone for the Faster R-CNN, that have to perform the localization of the objects inside the same image. In this way, the meta-objective will be computed on the output of the object-detection task, using the parameters obtained from the fine-tuning steps of the rotation meta-task. With this approach, during the pre-training phase we perfectly simulate what happens in the adaptation phase, but we move away from the original implementation of MAML, still following its meta-optimization procedure.

2.3.2 Problem formulation

OSHOT is composed by 3 main blocks:

- G_f, the features extractor, parametrized by θ_f, that from an input image return a features map;
- G_r, the self-supervised rotation classifier, parametrized by θ_r, that predict the orientation starting from the features map of the rotated version of the input;
- G_d, the object-detector, parametrized by θ_d, that, from the features map produced by G_f, predict the bounding boxes and labels of all the object inside the input.

Pre-training phase. We start from N annotated samples of the source domain $S = \{x_i^s, y_i^s\}_{i=1}^N$, where the structured labels $y^s = (c, b)$ describe class identity c and bounding box location b of each object inside the image x^s. At each iteration, one sample x_i^s is loaded from S and γ steps of the rotation classifier G_r are performed on the rotated version of the input image, called $R(x_i^s)$. The orientation q_i^s is chosen randomly in all the steps, so we may have different $R(x_i^s)$ as result of the transformation on the input image before passing through the rotation classifier. Thus, from the model parametrized by $\theta = (\theta_f, \theta_r, \theta_d)$, we obtain the following objective function:

$$\arg\min_{\theta_f, \theta_r} \lambda \mathcal{L}_r(G_r(G_f(R(x_i^s)|\theta_f)|\theta_r), q_i^s). \tag{2.4}$$

where λ is used to adjust the importance of the auxiliary self-supervised rotation task with respect to the main one, *i.e.* the localization of objects. Although G_r and G_d work separately in this new OSHOT version, they both share the same backbone G_f, which means they work on the same features map. Therefore, the higher the λ, the more the rotation task influences the training of the model, updating the backbone in such a way that it is more able to solve the rotation task rather than the localization. If, on the other hand, λ is too low, we will not exploit the rotation task too much and we will lose its added value, *i.e.* making the network able to generalize better.

Note that the rotation meta-task is carried out following the same *cross-task pseudo labeling* procedure used by the OSHOT during the adaptation phase, in order to strictly follow this procedure.

It is clear that the Equation (2.4) involves only the backbone of the network and the rotation classifier and minimizing it we obtain:

$$\theta_f^{'}, \theta_r^{'} = (\theta_f, \theta_r) - \alpha \nabla_{\theta_f, \theta_r} \mathcal{L}_r(G_r(G_f(R(x_i^s)|\theta_f)|\theta_r), q_i^s), \tag{2.5}$$

that are the update parameters of the model after the fine-tuning step on the rotation classifier, using the **SGD** gradient-based learning rule. This is our **inner-loop optimization**, in the MAML terminology, and it is performed γ times on x_i^s. In practise, this is the same procedure used in the adaptation phase before performing the final prediction on the target image. In fact our objective is to make the model aware of the situation that it will face at inference time.

Now, following the same procedure of the adaptation phase, we need to compute the final prediction on x_i^s, *i.e.* bounding boxes and classes of all the instances inside the input. First of all, we discard $\theta_r^{'}$ because it is not part neither of the features extractor nor of the detector. Then, we pass to G_d, the Faster R-CNN detector, the features extracted from the input image x_i^s by the backbone G_f with fine-tuned parameters $\theta_f^{'}$. In this way, following the idea of *Finn et al.*[5], we want to asses the performance of the model after one or more gradient step performed on the backbone by the rotation classifier, that is our meta-task. Thus, our **meta-objective** is:

$$\arg\min_{\theta_f, \theta_d} \mathcal{L}_d(G_d(G_f(x_i^s|\theta_f^{'})|\theta_d), y_i^s), \tag{2.6}$$

computed using the updated model parameters $\theta_f^{'}$ and the parameter of the detector θ_d. Note that, the meta-objective wants to minimize the initial model parameters (θ_f, θ_d). In fact, our goal is to make the network capable of *learn how to adapt* and for this reason we use the fine-tuned parameter of the backbone as a starting point to produce a test-error on the localization of the objects, that serve as the training error of the meta-learning process. At the end, the meta-optimization is performed with the **SGD** gradient-based learning rule and the result will be:

$$\theta_f^*, \theta_d^* = (\theta_f, \theta_d) - \alpha \nabla_{\theta_f, \theta_d} \mathcal{L}_d(G_d(G_f(x_i^s|\theta_f^{'})|\theta_d), y_i^s). \tag{2.7}$$

This is our **outer-loop optimization** of the initial model parameters (θ_f, θ_d), where we apply a first order approximation imposing $\frac{d\theta_f^{'}}{d\theta_f}$ as identity in the back-propagation.

To sum-up, we train the network in order to prepare it for the adaptation phase. Thus, we followed guidelines provided by MAML and we matched train and test conditions, so that the network can learn better how to adapt to new target samples. The important thing to note is that this procedure only modifies the pre-training phase of the OSHOT, instead the adaptation phase is unchanged.
This new algorithm is called **META OSHOT**, the meta-learning version of the OSHOT. This is the baseline on which further modifications will be made, in order to create an architecture that completely resembles the adaptation phase.

2.4 Data Augmentation Transformations

The OSHOT adaptation phase is carried out on samples that come from different domains than the one used during the pre-training phase. In fact, the goal is to evaluate the degree of generalization acquired by the model, exploiting information about the target samples only before the final prediction.

The reflection that arises spontaneously is: why not adopt a strategy, in the pre-training phase, in order to allow the network to analyze also samples slightly different from those belonging to the source domain?
We can choose 2 main roads:

- the first is to modify the source images to reflect the style of the target domains, while preserving their content. These techniques belong to the field of Neural Style Transfer[34] and many methods, that belong to this field, use GANs[18] methods to perform style transformations. The problem with these approaches is that they require a lot of time to translate a single image and it is also necessary to know in advance the target domains;

- the second is to apply particular data augmentation transformations on the source images that modify their visual appearance, in order to move the images away from the domain they belong to. Images, obviously, cannot be completely transformed, but they can be edited to obtain altered brightness and clarity, swapped colors, etc.

In this master thesis project we will use simple data augmentation transformations. The reason for this choice is first of all linked to the setting in which we are, i.e. we have no information about the target images in the pre-training phase. Another reason is to keep the final algorithm as simple as possible and therefore we do not want to add additional components to the network that would make the training much slower and more computationally intensive.

2.4.1 Data Augmentation Overview

It is commonly known that to train machine learning models we need large amounts of data, because these models are made up of millions of parameters that must be initialized well, in order to obtain good performance on the tasks to be performed.

Additionally, in the real world scenario we may have dataset of images taken in a limited set of conditions but, our target application may exist in a variety of states, such as different orientation, location, scale, brightness, style etc. In order to cope with this kind of situations, we need to train our models with data that reflect also this variability.

Data Augumentation is a technique that allows us to increase the number of data available in the training phase, so as to cover more cases that can be encountered in the test phase. It is a widely used method because it acts as a regularizer and helps reduce overfitting when training a machine learning model. In general, it consists in applying transformations to images available in the training phase, in order to create new samples that are slightly modified copies of already existing samples. A practical example is just

flipping horizontally all the samples, so that we increase the size of the dataset by a factor of 2 and we analyze also images in different orientations.
There are two ways to apply this technique:

- **offline augmentation**, where transformations are applied before starting the training of the model. In this way, we increase the size of the dataset by a factor equal to the number of transformation used;

- **online augmentation**, where transformations are applied on the mini-batches that we feed into our model. Thus, we do not increase the number of samples of the dataset, but we apply the transformations directly to the images without creating new samples. However, the important thing is not only to have more samples, but to obtain different versions of the inputs, so that the network can learn characteristics and features useful for target tasks.

There are different transformations that can modify some characteristics of the input, such as orientation, brightness, scale, aspect-ratio, viewpoint, degree of rotation, etc. But, there are also transformations that modify the visual appearance of an image such as jittering, grayscale, blurring, inversion of colors, change of channels, etc .

2.4.2 Transformations of the Visual Appearance

In this master thesis project we will consider different transformations that we will apply at each iteration of the pre-training phase. They are transformations designed to change the way in which images appears to the model, so as to be different from the usual source domain images. This is done because, during the adaptation phase, the target samples can belong to different domains, which also means different styles and visual appearances of the various objects represented. Since we want to perfectly simulate this situation, we will use in the pre-training phase transformations that modify the visual appearance of the image, in order to help the network in the *learning to adapt* process.
Below we will describe briefly the main transformations used with same visual examples.

Color-Jitter transformation

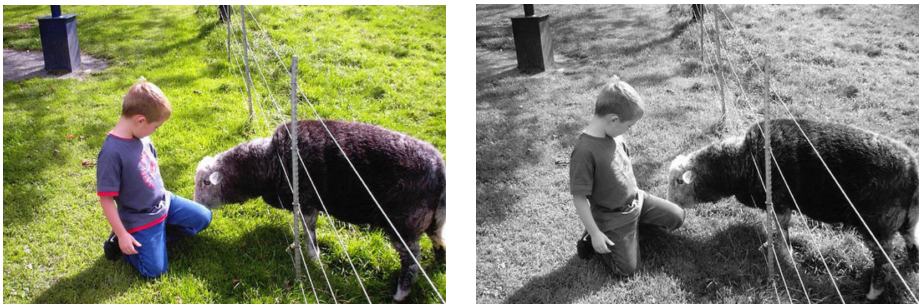
It is a transformation that randomly change the brightness, contrast, saturation and hue of an image (Figure 2.2). For our experiments we use the function inside torchvision[35] that accepts 4 parameters, one for each of the four property of the image defined above. Each parameter is used to define the range from which the function will choose randomly the jitter factor to be applied to the property.

Grayscale transformation

A simple transformation that convert an image into grayscale (Figure 2.3). For our experiment we use the function implemented in torchvision[35].

(a) Original image (b) Color-jittered image

Figure 2.2: Example of color-jitter transformation on a VOC 2012's sample, where the 4 parameters are sets to 0.4.



(a) Original image (b) Grayscale image

Figure 2.3: Example of grayscale transformation on a VOC 2012's sample.

Gaussian Blur transformation

This transformation consists in applying a mathematical function, a Gaussian function, to blur an image (Figure 2.4). The visual effect of this blurring technique is like viewing an image through a translucent parchment. For our experiments we use the function inside the Python Imaging Library(PIL), that accept one parameter, called radius, that define the intensity of the blurring effect, where higher value means higher blur.

Color-Inversion transformation

This transformation consists in inverting each pixel value of an image. The transformed image appears as if it was converted to a negative (Figure 2.5). The dark areas of the picture become bright and bright areas become dark, more specifically the black becomes white, the orange becomes blue, the white becomes black and so on. If we apply 2 times this transformation to an image, we obtain again the same image. For our experiment we use the function inside the PIL library.

(a) Original image (b) Blurred image

Figure 2.4: Example of Gaussian Blur transformation on a Cityscapes's sample, where the radius parameter is set to 5.

(a) Original image (b) Inverted image

Figure 2.5: Example of color inversion transformation on a Cityscapes's sample.

2.4.3 FULL META OSHOT

Now, we can reformulate the pre-training procedure of the OSHOT in a way even closer to the adaptation phase. In fact, with the help of the transformations, the model can analyze samples slightly different from the data distribution of the source domain, so as to simulate new target images.

We start from N annotated samples of the source domain $S = \{x_i^s, y_i^s\}_{i=1}^N$ and a set of transformations $Transf = \{Transf_k\}_{k=1}^J$. At each iteration one sample x_i^s is loaded from S and the transformations are applied one by one, in order to create J different instances of the input:

$$x_k^{tr} = Transf_k(x_i^s).$$

At this point, for each x_k^{tr} we perform γ steps of the rotation classifier G_r on the rotated version of it, called $R(x_k^{tr})$ with ground-truth q_k^{tr}, chosen randomly. Note that, in the rotation meta-task we have decided to use the **cross-task pseudo labelling** procedure of the OSHOT, because, if we want to be more adaptation-phase compliant, we have to take in mind that we do not have labels of the original image and so we cannot exploit, in the auxiliary self-supervised task, the ground-truth bounding boxes of the input, as the OSHOT does in the pre-training phase.

This means that, for each of the γ steps, first of all we produce the pseudo-labels from the detector G_d and after that we take only the bounding boxes of the prediction, to

consider only the relevant objects inside the features map of the rotated image. In this way, the rotation classifier have to look only to parts of the input that the detector predict as objects, so that irrelevant information of the background are filtered out. The advantage is that the final task of predicting the orientation become more difficult to perform, because the network have to infer the orientation only from crops of the original image that maybe contain objects.

Thus, from the model parametrized by $\theta = (\theta_f, \theta_r, \theta_d)$ we obtain the following objective function, for each of the transformed images:

$$\arg\min_{\theta_f, \theta_r} \lambda \mathcal{L}_r(G_r(G_f(R(x_k^{tr})|\theta_f)|\theta_r), q_k^{tr}), \tag{2.8}$$

where λ is used to adjust the importance of the auxiliary self-supervised rotation task with respect to the object-detection one.
This process involves only the backbone of the network and the rotation classifier and minimizing this equation we obtain:

$$\theta_f^k, \theta_r^k = (\theta_f, \theta_r) - \alpha \nabla_{\theta_f, \theta_r} \mathcal{L}_r(G_r(G_f(R(x_k^{tr})|\theta_f)|\theta_r), q_k^{tr}), \tag{2.9}$$

that are the update parameters of the model after the fine-tuning steps performed by the rotation classifier on the transformed image, using the **SGD** gradient-based learning rule. This is our **inner-loop optimization** and it is performed γ times for each of the transformed images.

Note that, with the addition of the different versions of the input image, we can think that each sample is a meta-task to be performed. Thus, we have different source domains from which we can adapt our network, so that at inference time the model is able to perform better on different target domains. In this way our implementation follow strictly the guidelines of **MAML**, where the authors[5] use different tasks during the meta-training procedure, structured related to the ones that the model have to face at meta-test time.

Thanks to this new way of performing the inner-loop optimization, we obtain different θ_f, each one corresponding to the fine-tuning steps on the *k-th* transformed image. Now, following the same approach of the adaptation phase, we need to compute the final prediction on each of the x_k^{tr}. Thus, we pass to G_d the features extracted from the transformed image x_k^{tr} by the backbone G_f parametrized by θ_f^k. In this way, following the idea of *Finn et al.*[5], we want to asses the performance of the model on each of the just introduced "*meta-tasks*", after one or more gradient steps performed on the backbone by the rotation classifier. Thus, our **meta-objective** is:

$$\arg\min_{\theta_f, \theta_d} \sum_{k=1}^{J} \mathcal{L}_d(G_d(G_f(x_k^{tr}|\theta_f^k)|\theta_d), y_i^s), \tag{2.10}$$

where each loss is computed from the prediction of the detector on the transformed image using the corresponding fine-tuned parameters θ_f^k. Note that y_i^s is equal for all the x_k^{tr}, because all the transformations modify only the visual appearance of the original image x_i^s, while the content is preserved. As already explained in the formulation of the **META OSHOT**, the meta-objective wants to minimize the initial model parameters (θ_f, θ_d). In

fact our objective is to push the network to *learn how to adapt* and we use the fine-tuned parameters of the backbone as a starting point to produce a test-error on the localization of the objects, that serve as the training error of the meta-learning process. At the end, the meta-optimization is performed with the **SGD** gradient-based learning rule and the result is:

$$\theta_f^*, \theta_d^* = (\theta_f, \theta_d) - \alpha \nabla_{\theta_f, \theta_d} \sum_{k=1}^{J} \mathcal{L}_d(G_d(G_f(x_k^{tr}|\theta_f^k)|\theta_d), y_i^s). \tag{2.11}$$

This is our **outer-loop optimization** of the initial model parameters (θ_f, θ_d), where we apply a first order approximation imposing $\frac{d\theta_f^k}{d\theta_f}$ as identity in the back-propagation, following MAML strategy.

To sum-up, in the previous sections we introduced the META-OSHOT, which consists in modifying the pre-training phase of the OSHOT to make it able to *learn to adapt* to the various samples.

Now, with this new reformulation we have introduced additional changes to the OSHOT, adding two fundamental components:

1. the transformations of the source images, which allow us to analyze samples that differ from the single domain available in the pre-training phase. This make the model even closer to the characteristics of the OSHOT adaptation phase, where each target sample can belong to a different domain;

2. the creation of "*meta-tasks*", which allow us to make the algorithm even closer to the way in which MAML carries out its meta-training procedure. Our meta tasks are a bit different from those used by MAML, because we don't have a train and test part but they coincide. We made this choice because it is what happens in the adaptation phase of the OSHOT, that is fine-tuning and then prediction on the same target image;

thus, we added the missing pieces of both sides. This new version (Figure 2.6) is called **FU**LL **M**ETA-**O**SHOT(**FUMO**) and can be considered the most complete and closest algorithm to both MAML and OSHOT adaptation phase, which was our main goal.

Implementation details

FUMO involves two important problems:

1. since we use more than one image at each iteration, the demand for resources is higher, especially for the GPU;

2. with the use of the cross-task pseudo labeling procedure, the model have to predict the bounding boxes on the transformed image before carrying out the rotation task.

To overcome the first problem, we have imposed a limit on the maximum number of transformations that can be performed at each iteration. The choice is to use a maximum of 3 variants of the source image, for our experiments.

The second problem is more complex because, in order to predict good bounding-boxes from the images, the model have to be trained for a certain number of iterations, otherwise it happens that the detector either does not find any bounding-boxes from the various images, canceling the effect of the rotation task, or fails to locate objects by considering many background scenes.

Thus, we have decided to do in the following way:

- the first phase will be dedicated to training the OSHOT for a certain number of iterations. At the end of this procedure we will have a good pre-trained model that will allow us to use the detector in the cross task pseudo labeling procedure;

- the second phase will use the previously trained model as a starting point for the FULL META-OSHOT training procedure. Since our algorithm is an extension of the OSHOT, it is easy to start from its trained model and apply the new algorithm on top.

This way of training not only brings us benefits to ensure the effectiveness of the rotation task, but also allows us to decrease the execution time of the algorithm, which, if used from the beginning, takes a long time to complete the pre-training phase.

This second problem also exists in the META-OSHOT, which does not use transformations but performs the cross-task pseudo labeling procedure. Thus, also in this case we have decided to operate as FUMO.

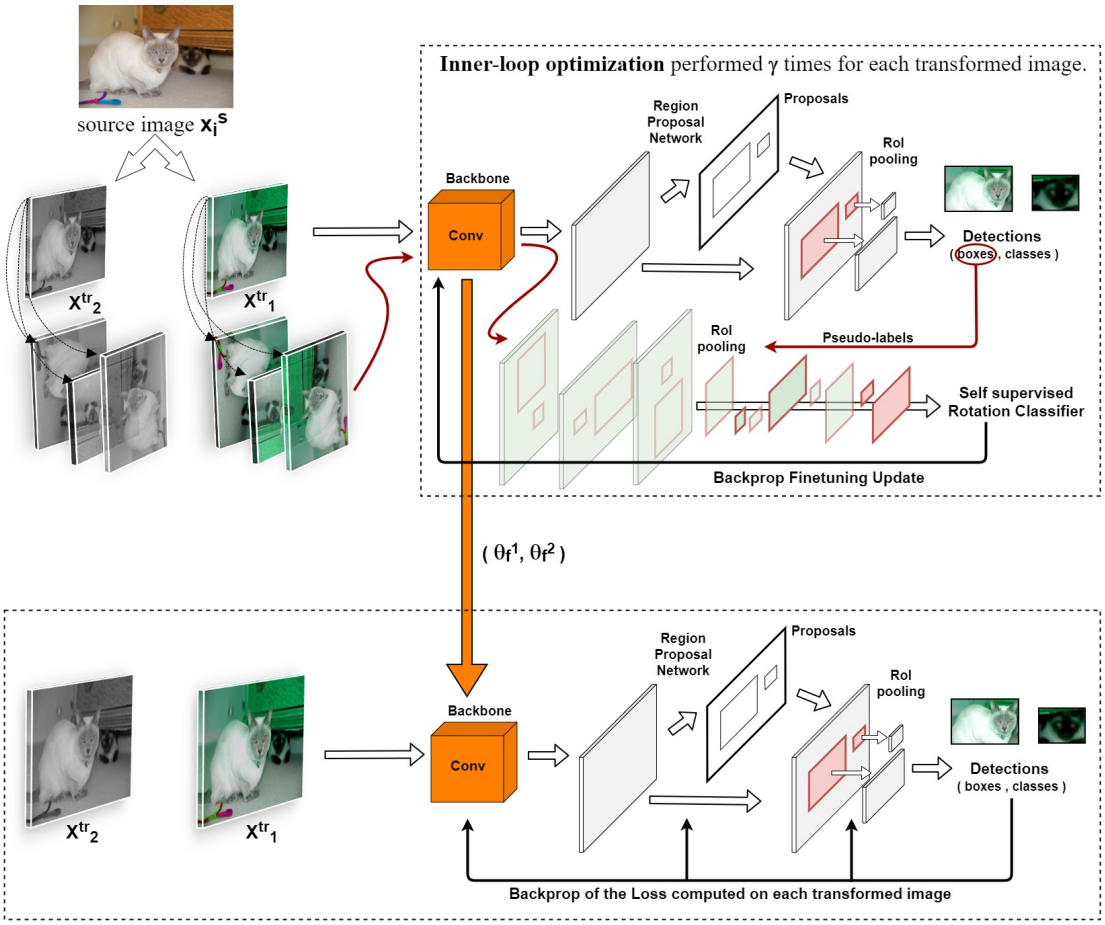


Figure 2.6: Pre-training phase of the FULL META OSHOT.

Chapter 3

Dataset and Evaluation Metrics

3.1 Dataset

This section will show different benchmarks used to compare this thesis work with other methods on the visual object detection task. Each subsection represents different benchmarks, where the proposed methods are trained on a single dataset and tested on the others. Note that, every dataset is a representation of a unique domain, except a new one.

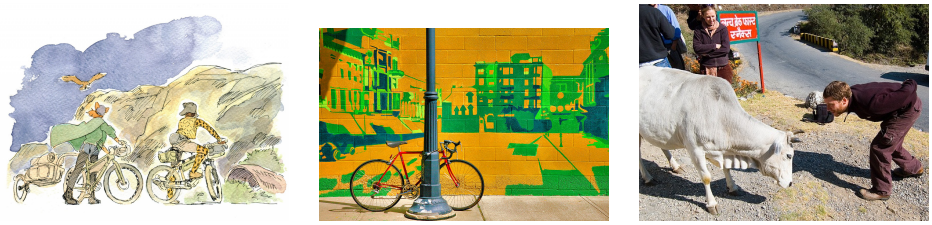
3.1.1 Pascal VOC, Artistic Media dataset and Social Bikes

This benchmark consider 5 object detection dataset that are different in terms of visual appearance, in fact they belong to a different domain, but they share the same set/subset of categories.

PASCAL VOC 2007 and 2012 sets. This dataset was born on the Pascal Visual Object Classes (**VOC**)[36] challenge, to provide the machine learning communities with a standard dataset of photos and annotation to set goals for methods and allow comparison of their performance. This challenge is organised annually, since 2005, and in this thesis the focus is on the data collected in the 2007 and 2012 editions, because they consider the same set of classes and it is trivial to join them in an unique dataset. VOC 2007 contains 5011 images in the train-val split and 4952 images in the test split, while VOC 2012 contains 11540 images in the train-val split. The number of total object category is 20 and each object is fully annotated with its class label and bounding box coordinates. Images, sometimes called **natural images**, are collected from the flickr photo-sharing web-site and represent real object obtained via different cameras, with significant variability in terms of object size, orientation, position etc., to better simulate the mutability of the world.

Clipart1k, Comic2k and Watercolor2k. This 3 datasets, also known as *Artistic Media Datasets* (AMDs), were created by *N. Inoue at al.*[37] for their analysis in cross-domain weakly supervised object detection. The objective of the authors was to create a model able to automatically detect and annotate each object inside an image, in order to make

(a) Clipart1k's sample (b) Comic2k's sample (c) Watercolor2k's sample



(d) Social Bikes' sample (e) Social Bikes' sample(2) (f) VOC 2012's sample

Figure 3.1: Example of images belonging to the 5 datasets. As pointed out, Social Bikes contains different domains

the process of creating labelled sample for the object detection task easy to perform and less human centred. In fact, these datasets with instance-level annotations was created to evaluate the performance of their framework and, as we can see from the name, each of them represents a different domain where the number of samples are limited. **Clipart1k** contains 20 categories, the same of VOC, but **Watercolor2k** and **Comic2K** contain only 6 category, which are a subset of the VOC ones.

Social Bikes. As already mentioned in the Related Work Chapter (2.2), this thesis is strictly related to the work of *D'Innocente at al.*[2] that also introduce a new concept-dataset containing scenes with only person and bicycles, collected from Twitter, Instagram and Facebook by searching for the tags *#bike*. This dataset is called **Social Bikes** and shows different style properties, because images were acquired randomly from social feeds and it is difficult to group them under a single domain. Thus, this is the only dataset inside this thesis that is composed of different domains, but the only categories of interest are: *bike* and *person*, that are a subset of the 20 categories of VOC.

The experiments carried out on this benchmark are structured as follows: the model will be trained only on Pascal VOC (**source domain**), considering all its categories, and then it will be tested on each of the remaining 4 dataset (**target domains**), in order to study the degree of generalization produced by Meta-Learning on data never seen before, exploiting the **One-Sample Adaptation** strategy of [2]. Even if the categories of the target domains are a subset of the source one, this does not affect the analysis because, in any case, they have been considered in the training phase. Therefore, the important thing is that the test dataset still contain the same or a subset of categories of the source domain to assess the performance achieved.

3.1.2 Cityscapes, Foggy Cityscapes and KITTI

These benchmarks deal with urban scenes in order to create object detection models useful for autonomous-driving scenario.

The Cityscapes dataset. Despite significant advance in the object-detection field of research, *visual scene understanding* remain challenging. Most of the work are related to autonomous-driving, where there is the need of reliably detect objects in urban areas. Due to the greater attention to this topic, different dataset were build to address this problem[38][39], but most of them are often much smaller than dataset addressing more general settings. For this reasons, in 2016 the **Cityscapes** dataset[40] has been presented, together with a benchmark, to capture the variability and complexity of real-world inner-city traffic scenes. This dataset was acquired from a moving vehicle during the span of several month and seasons in 50 cities, most of them located in Germany. After the video-recording, the authors created both images and instance-level/pixel-level annotations of the objects inside them manually and provided a dataset with 5000 samples, 2975 in the train-set, 500 in the validation set and 1525 in the test set. The total number of classes are 30 divided in 8 categories, but in this thesis work we will consider only a subset of them composed by 8 classes.

Thanks to its completeness and complexity, this dataset is often used for most of the object detection tasks and especially in works related to autonomous driving in urban environments.

(a) Cityscapes' sample (b) Foggy Cityscapes' sample

Figure 3.2: Comparison of the same image with and without fog.

The Foggy Cityscapes dataset. It is natural to think that models created to identify urban scenes are able to deal with any weather condition. However, most of the dataset, as well as Cityscapes, contain images collected in optimal weather conditions and this involves a degradation of performance when such models are used in real applications, where everything is unpredictable. For this reason **Foggy Cityscape**[41] has been presented as a benchmark for semantic foggy scene understanding (**SFSU**), with the aim of having a dataset capable of training/testing models in unfavourable weather conditions like fog. The authors show an automatic pipeline to add synthetic yet realistic fog to urban scenes of the Cityscapes dataset, creating 550 foggy images with fine semantic annotation directly inherited from Cityscapes. Most of the works use both dataset to create systems, like

autonomous-driving vehicles, able to tackle adversarial whether condition and guarantee the safety of the passengers.

The KITTI dataset[42]. This dataset was presented in 2013 with the main purpose of pushing forward the development of computer vision and robotic algorithms targeted at autonomous driving. Images have been recorded from a moving vehicle while driving in and around a city in Germany, to capture real-world traffic situations, ranging from freeways over rural areas to inner-city scenes. The authors provide bounding boxes and labels of the objects inside the images. It is composed by 7481 images belonging to the trainval split and the total number of classes are 8, but the most predominant are *Car* and *Pedestrian*. In this thesis work we will consider only the class *Car*.

Figure 3.3: Sample from KITTI dataset.

The experiments will be carried out in the following way: the model will be trained on Cityscapes and tested, following the One-Sample Adaptation strategy, on Foggy Cityscapes and KITTI, in order to evaluate the domain generalization performance achieved by the model. The only difference in the evaluation process of Foggy Cityscapes and KITTI is related to the number of classes: for the former we will consider the same category of Cityscapes and for the latter we will consider only the *Car* class, following common practise on different benchmarks. Afterwards, the model will be trained from scratch on KITTI, considering all the related classes, and tested on Cityscapes, where again only the class *Car* will be considered to evaluate the results.

3.2 Evaluation Metrics

The output of an object-detector is more complex than the one produced, for example, by an image classifier. This means that we need metrics that go beyond the accuracy, in order to measure the performance of a detector. In fact, we need to considerate that the produced output varies from sample to sample, due to the fact that a different number of objects can be present inside. In addition, the model need to predict correctly both the bounding box that contains the object and the label that identifies the class to which the object belongs to.

The principal metric used to evaluate object detectors is the ***m**ean **A**verage **P**recision*, but many popular challenges use it with some variations in definitions and implementations. In this master thesis project we will refer to the metrics defined by the *PASCAL VOC Challenge*[36] and in particular to the definition of **mAP** and to the code used to obtain it.

Before introducing this fundamental metric, we will show some basic definitions that help us to understand it better.

3.2.1 Confidence Score and Intersection over Union

The *confidence score* is the probability that a bounding box contains an object. It is usually predicted by the classifier that outputs the class of the object that is inside the detected bounding box.

The *Intersection over Union*(**IoU**) is defined as the area of the intersection divided by the area of the union of a predicted bounding box(Box_p) and a ground-truth bounding box(Box_{gt}):

$$IoU = \frac{\text{Area}(Box_{gt} \cap Box_p)}{\text{Area}(Box_{gt} \cup Box_p)}$$

It assumes values between 0 and 1, where 0 means no overlap and 1 means perfect overlap between the predicted box and the real one. In general, it is necessary to fix a threshold that is used to distinguish a valid detection from the others. In our case we assume 0.5 as threshold, following [36].

Both confidence score and IoU are used as criteria that determine whether a detection can be considered correct or not.

3.2.2 Precision and Recall

Before entering into the details of these two metrics, we need to redefine some definitions for the object-detection field:

- **True Positive** (TP): a correct detection, where the predicted class matches the class of a ground truth and the predicted bounding box has an IoU greater than a threshold (in our case 0.5) with the ground-truth box. The PASCAL VOC Challenge[36] includes an additional rule to define true positives: in case multiple predictions correspond to the same ground-truth, only the one with the highest IoU counts as a true positive, while the remaining ones are considered false positives.

- **False Positive** (FP): an incorrect detection, where the predicted class does not match the class of a ground truth or the predicted bounding box has an IoU lower than a threshold (in our case 0.5) with the ground-truth box.
- **True Negative** (TN): in the object-detection task it corresponds to the correct detection of the background, but we usually don't care about these kind of detection because we are not interesting in localize these areas of the image.
- **False Negative** (FN): ground truth bounding box with no matching detections.

Precision is the proportion of true positive detections for a given class:

$$precision = \frac{\text{TP}}{\text{TP + FP}} = \frac{\text{TP}}{all\ detections},$$

in practise it measures the ability of a detector to identify only relevant objects.

Recall, on the other hand, is the proportion of true positives detected among all ground-truths for a given class:

$$recall = \frac{\text{TP}}{\text{TP + FN}} = \frac{\text{TP}}{all\ ground\text{-}truths},$$

in practise it measures the ability of the model to find all relevant cases (all ground-truths).

A good model is the one that can identify most ground-truth objects (high recall) while only finding the relevant cases (high precision). Note that these 2 metrics can be computed only for one class at the time.

3.2.3 Precision-Recall Curve

We can draw a *precision-recall curve* with recall on the x-axis and precision on the y-axis, by setting the threshold for *confidence score* at different levels, with the idea that both precision and recall should remain high, even if the confidence threshold varies. What we get is a decreasing monotonic function, meaning that increasing the value of the recall decrease the precision and vice-versa. This trend occurs because:

- if the precision is high, this means that the FP is low, but this can results in high value of FN(low recall) because many object instances are missed;
- if the recall is high, this means that FN is low, but also that FP may increase(low precision) due to wrong detections;

thus, this curve shows trade-off between the two metrics for varying confidence values.
In general this curve is used to evaluate the performance of binary classification algorithms, but, with our reformulation of precision and recall, we can use it also to asses the performance of object-detection's models, considering one class at the time. If we plot curves of multiple models in the same graph, the one with the curve above the others has a better performance level.

3.2.4 mean Average Precision

Sometimes precision-recall curve can be a zigzag curve frequently going up and down due to lack of data and comparisons with multiple models can be difficult, if curves tend to cross each other frequently. Thus, the solution to this problem is to use a single value to compare different models and for this reason it is common practise to use the *Area Under the Precision-Recall Curve* (**AUC-PR**) also known as Average Precision (**AP**):

$$AP_\alpha = \int_0^1 p(r)dr,$$

where α is the IoU threshold used to define TP, FP, FN, and $p(r)$ is the precision-recall curve, a function that outputs a value of precision given a specific point of recall.

For the purpose of comparing different models we follow this procedure:

1. each model is tested on the same dataset where the AP is computed for each class, starting from from the model predictions and the ground-truth labels of the dataset;
2. the AP of each class is averaged in order to create the *mean Average Precision* (**mAP**) of the model:
$$mAP = \frac{\sum_{i=1}^{k} AP_i}{K} \quad for\ K\ classes; \tag{3.1}$$
3. at the end, the model with the highest value of **mAP** is the one that best perform.

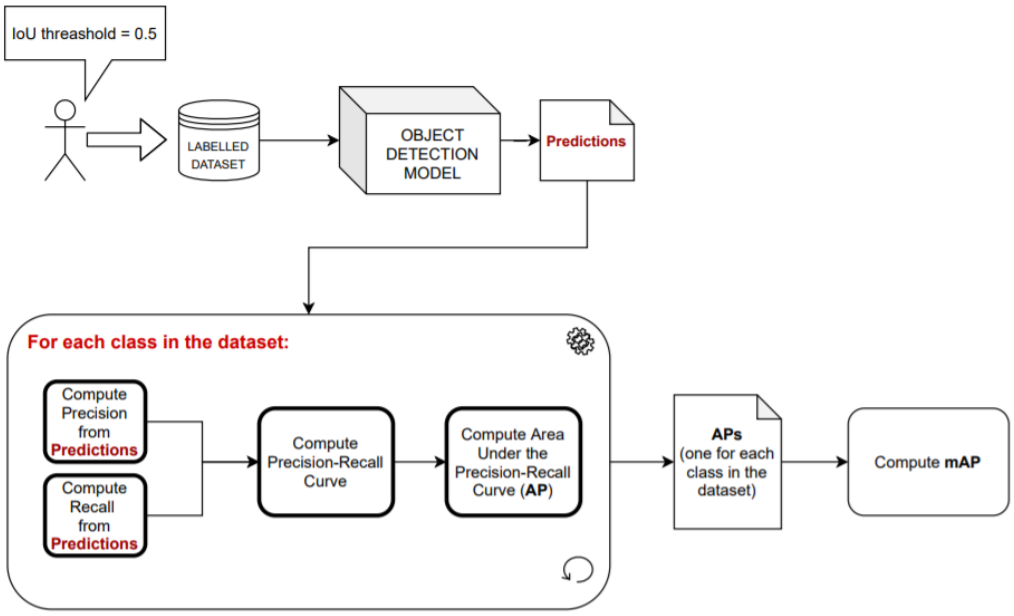


Figure 3.4: Complete process to obtain the **mAP** metric.

Chapter 4

Experiments and results

4.1 Introduction

Cross-domain analysis is typically performed with one of the following settings: domain adaptation or domain generalization.

The Domain Generalization(**DG**) setting, in the case of the object-detection task, is defined as follow:

- we have S domains available, each one composed by N labeled images. Note that each domain must share the same set/subset of categories;
- each label is composed of a set of *instance annotations*, each one consisting of a class label, representing the type of object, and 4 coordinates, representing the bounding box that contains the object in question;
- S-1 domains, called **source domains**, are used during the training phase to allow the model to obtain a certain degree of generalization;
- the left out domain, called **target domain**, is used during the testing phase, in order to evaluate the model's performance on a data distribution not seen during training. Note that the target domain may contains different domains inside.

DG is a zero-shot problem because performances are immediately evaluated on the target domain with no further learning or fine-tuning steps on target samples.

The Domain Adaptation setting(**DA**) is a little different:

- in general we consider only one source domain, but there are also variants that use multiple source domains, if they are available;
- the target domain is known a priori;
- during the training phase the model is trained both on the source domain, in a supervised way, and on part of the target domain, in a supervised or unsupervised way;

- during the test phase the model performances are evaluated on the target domain.

In the case of DA it is important that the model, in the test phase, performs well only on the target domain, therefore the performances it gets on the source or on other domains are not taken into account.

These two settings have been explained because, in our experimental analysis, we will follow something that is not strictly part of DG or DA, but contains characteristics of both: the **One-Sample Adaptation** setting, formulated and used by *D'Innocente at al.*[2] in the OSHOT(Section 2.2.2).

4.2 One-Sample Adaptation setting

This new setting has its own characteristics that have to be followed in order to make fair comparisons between the various models:

- there is no need to have more than one source domain;
- target domains are not known a priori, but they have to contains the same set or a subset of categories of the source one;
- during the training phase, the model is trained exclusively on the source domain;
- the test phase, or also called adaptation phase, is performed on one target image at the time;
- for each unlabelled target sample, fine-tuning steps are carried out on the model parameters and then the final prediction is calculated, which in the case of object-detection is a set of *instance annotations* that include the bounding boxes and class labels of each object contained in the image.

We use the auxiliary rotation task to update the network parameters on the target sample. After the final prediction is calculated, the initial parameters of the model will be restored to those coming from the training phase, so that again the network is ready to adapt to a new target sample. The One-Sample Adaptation procedure is completely transparent to the end user, in fact we can simply refer to it as the testing phase because what matters to the end user is only the prediction on the image passed to the object detector.

The One-Sample Adaptation setting is very similar to the DG one, in fact during training there is no trace of target samples, but, unlike DG, in the test phase the model exploits the target sample in order to extract information that can be helpful for the final prediction. On the other hand, this new setting is further away from DA, but follows the concept of adapting to the target samples that is used only in the adaptation phase.

The main purpose of this master thesis work is to improve OSHOT in cross-domain analysis. Since OSHOT already uses the One-Sample Adaptation setting, it is easy to make a fair comparison between our methods and it in the same benchmarks. The problem is instead when we want to compare our methods with other algorithms that use different settings or belonging to DG or belonging to DA.

For this reason, in this section we will show the performances obtained in various benchmarks by our algorithms and the OSHOT, using the One-Sample Adaptation setting. In the last section, we will show the comparison with 2 models used in the DA scenario, highlighting the related problems and the changes made to make the final comparison as fair as possible.

4.2.1 Description of the methods

Before going into the details of the various experiments, we will briefly introduce the methods used for comparisons on the various benchmarks.

OSHOT methods

Baseline The baseline model is the simple **OSHOT**[2], discussed in the 2.2 Method Section. Here we will use its original implementation to assess whether our methods really exceed its performance.

OSHOT-TRANSF This method is a reformulated version of the baseline. We created this version to make sure that our meta-leraning procedure applied to OSHOT really does its job, which is to make the network able to *learn to adapt*. In fact, we applied the same strategy of transformations, used in our new methods, even in the basic version of the OSHOT, in order to understand if the advantages we obtain on the final prediction are due to the new introduced meta-learning procedure or only to the introduction of this form of data augmentation. In this way, we can also evaluate the performance of this new version of the baseline and understand whether it is a valid alternative to the basic OSHOT.

Thus, this method differs from OSHOT only in the pre-training phase, where:

- for each iteration we apply a set of transformations on the input, creating multiple versions of the source image;
- each transformed image is passed to the model, which perform a prediction and calculate a loss function on it;
- all the loss functions of the transformed images are added together and a single backward and step is performed on the sum.

Note that the model is the same of that used by the OSHOT(Section 2.2) on each source image. To sum-up, in this method we do not use a meta-learning procedure, but we introduce, at each iteration of the baseline, only transformations of the visual appearance, leaving everything else unchanged.

META OSHOT methods

META OSHOT This method is the first meta version of OSHOT, described in the 2.3.2 Method Section. Here transformations are not yet present, but it is the first approach that follows the meta-learning procedure used by *Finn et al.*[5].

FUMO Finally, we present the most complete meta version of the OSHOT, the **FU**LL **M**ETA **O**SHOT (**FUMO**), described in the 2.4.3 Method Section. This method, as already explained, uses a set of transformations that it applies at each iteration on the source image, in order to create "*meta-tasks*" used to update the parameters of the model in the pre-training procedure. This strategy perfectly simulates what OSHOT will face in the adaptation phase.

4.2.2 Experimental Setup

Since all the methods described above are changes made to the pre-training phase of the OSHOT, first we will describe the configurations used for it and then the changes made to the other methods.

OSHOT

We used the implementation of the OSHOT provided by the same authors of the paper[2], written in PyTorch[43] and made public[44]. This implementation includes two macro blocks: the Faster R-CNN[3] and the rotation classifier(Related Works Section 1.3.1). In addition, as already described in the Method Section, we have an initial pre-training phase of the model and subsequently an adaptation phase.

Faster R-CNN. The implementation used for this object detector is the one developed by Facebook[45] in Pytorch and made public. Our configurations are:

- ResNet-50[46] backbone pre-trained on ImageNet;
- anchors of three scales (128, 256, 512) and three aspect ratios (1:1, 1:2, 2:1), so for each point of the features map, produced by the backbone, we have 9 different bounding boxes candidates;
- RPN produces 300 top proposals after non-maximum-suppression(NMS);
- IoU threshold is set at 0.5 for the mAP results.

D'Innocente at al.[2] followed these settings in order to fairly compare their methods with others on the same task.

Auxiliary Self-Supervised Rotation Classifier. The OSHOT authors added this classifier into the Faster R-CNN framework, specifically as a second branch starting from the backbone output. In this way, they train the net in a single-stage with a multi-task loss. The configurations are as follows:

- the weight of the auxiliary task is set to $\lambda = 0.05$;
- for each ground-truth/detected object inside the image is considered a random rotated version of it. The possible degrees of rotation are (0°, 90°, 180°, 270°) and the chosen one becomes the ground-truth label that the rotation classifier have to predict.

Pre-training phase. Before passing the image into the network, it is resized so that the shorter side is equal to 600 pixels and a random horizontal flip transformation is performed on it. Other configurations are:

- the first 2 blocks of the ResNet50 backbone are freezed;
- the batch size is equal to one;
- batch normalization layers are kept fixed for both pretraining and adaptation phase.

The authors train the network using SGD with momentum set at 0.9. Unless differently specified, the network is trained for 70k iterations, with an initial learning rate of 0.001 and a decay-step after 50k iterations.

Adaptation phase. In this procedure finetuning steps are carried out on the model coming from the pre-training phase using only the self-supervised rotation task, which works on rotated versions of the target objects. The batch size is equal to 1 and, unlike training, a dropout with probability $p = 0.5$ is added before the rotation classifier, because, as the authors say, this can be helpful to prevent overfitting on the self-supervised task. The weight of the auxiliary task is increased to $\lambda = 0.2$ to speed up the adaptation process. All the other hyperparameters and settings are the same used during the pre-training procedure. The official total number of fine-tuning steps on the single target image is 30, but for our experiments we use 10 as maximum number of steps, because we want to show that with our procedure the network is already able to achieve better results in fewer iterations.

OSHOT-TRANSF

This method differs slightly from the **baseline** and the only changes made are in the pre-training phase.

Pre-training phase The first 60k iterations are performed following the basic OSHOT algorithm and using its hyperparameters. Only in the last 10k iterations of the pre-training phase we apply 2 transformations to the original image, to make a fair comparison with the meta versions of the OSHOT: Color-Jitter, with all the parameters set to 0.4, and Greyscale. This results in a batch size of 2 in the last 10k iterations, where we perform a loop over all the transformed versions of the source image. When the losses have been

calculated for all the elements of the batch, then a single backward and update of the internal parameters of the model is made, starting from the sum of all the single losses of the transformed images. Note that, the only difference from the baseline is that the network sees multiple images with a different visual appearance in one iteration, but from the implementation point of view everything else is the same.

META OSHOT

This method has substantially modified only the pre-training stage of the OSHOT, leaving the adaptation phase unchanged.

Pre-training phase In this phase for 60k iterations the model is trained following the OSHOT baseline training procedure. In the last 10k iterations the META OSHOT algorithm is executed. The inner-loop optimization is done 5 times per iteration, so $\gamma = 5$, with $\lambda = 0.05$ and batch size of 1. The other hyperparameters remain unchanged, following all the settings of the OSHOT.

FUMO

This method is a variant of the META-OSHOT training procedure, so we show only the differences that are present in the pre-training phase.

Pre-training phase Here we follow the same training strategy of the META OSHOT: 60k iterations of the baseline and the last 10k, instead, following the FULL META OSHOT algorithm. The main difference is that in each of the last 10k iterations we apply two transformations to the original image: Color-Jitter, with all the parameters set to 0.4, and Greyscale. Thus, the batch size of the last 10k iterations is 2, but the other hyperparameters remain the same of the META OSHOT and the OSHOT.

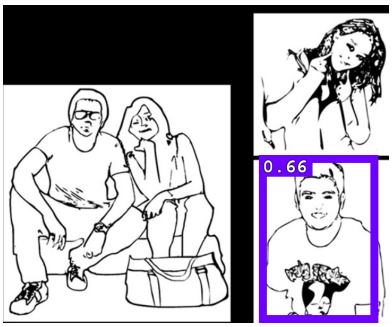
4.2.3 Detailed results and analysis

The metric used to evaluate the results is the **mAP** and we will consider the cases in which, during the adaptation phase, the network performs no adaptation, which means that a simple test on the target sample is performed without passing through the rotation classifier, 5 iterations of adaptation, *i.e.* $\gamma = 5$, and 10 iterations of adaptation. Each of the experiments are repeated 3 times in order to obtain more reliable results.

VOC to AMDs and Social Bikes

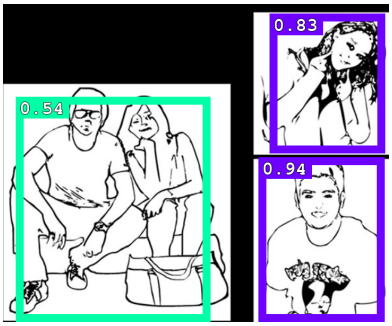
The dataset and the proposed benchmark are described in the 3.1.1 Section. We investigate this setting by training the source detector on the *trainval* split of Pascal VOC 2007 and 2012 sets. In total there are 16551 annotated images for this domain. To evaluate the performances of our models, we use for all the AMDs the *test* split composed by half of the total samples. In the case of Social Bikes, we will use all the 530 samples to evaluate the performance of the trained models, considering only the *bike* and *person* classes, as mentioned in the introduction of this dataset.

(a) Comic2k gt annotated image

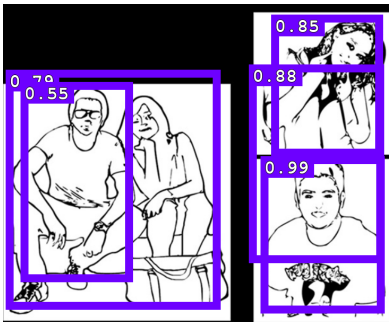


(b) OSHOT test

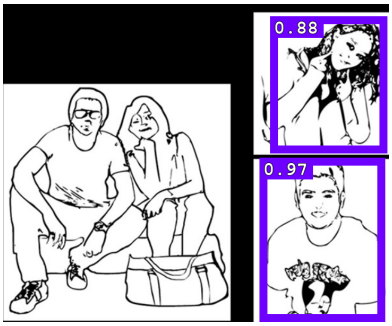
(c) FUMO test



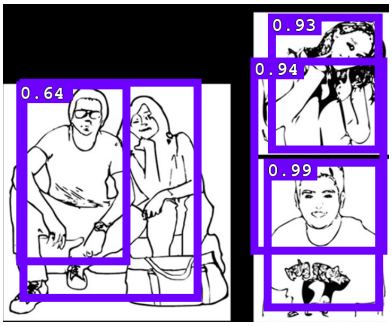
(d) OSHOT $\gamma = 5$



(e) FUMO $\gamma = 5$



(f) OSHOT $\gamma = 10$



(g) FUMO $\gamma = 10$

Figure 4.1: Example of object-detection results on a Comic2k image. The two methods are compared by considering how many fine-tuning steps they do during the adaptation phase.

VOC to AMDs results. In this setting the detector is trained only to analyze realistic images, instead in the test phase it has to predict on samples that come from different styles, shapes, colors, effects that can greatly disturb its performance. For this reason, artistic images are a very complex field, if they are not exploited during the training phase.

Method	Clipart			Comic			Watercolor			**Average mAP**		
	Test	$\gamma = 5$	$\gamma = 10$	Test	$\gamma = 5$	$\gamma = 10$	Test	$\gamma = 5$	$\gamma = 10$	Test	$\gamma = 5$	$\gamma = 10$
OSHOT	27.50	30.93	**31.77**	18.08	22.55	25.36	45.74	48.28	48.46	30.44	33.92	35.20
OSHOT-TRANSF	28.61	30.51	31.37	20.09	24.93	27.91	45.37	47.72	48.67	31.36	34.39	35.98
META OSHOT	**29,35**	31.39	31.52	20.18	24.77	26.67	45.78	**49.04**	48.80	31.77	35.06	35.66
FUMO	28.62	**31.66**	31.74	**21.08**	**25.20**	**28.60**	**46.36**	48.94	**49.83**	**32.02**	**35.26**	**36.72**

Table 4.1: Summary of the performances on Clipart1k, Comic2k and Watercolor2k dataset.

The results are shown in Table 4.1. The most important section is the **Average mAP**, which represents, as the name suggests, the average of the mAPs over the 3 dataset. These values make us understand how much a model performs on multiple domains and who is the best. The **AP**s of each method, for the classes inside these dataset, are reported in the Appendix A.

As we can see from the results, the OSHOT-TRANSF already exceeds the baseline, even if the differences are less than 1%. This means that the transformations of the visual appearance, introduced during the training phase, help the network to obtain more domains-invariant features. The META OSHOT, on the other hand, raises the mAP even more, making us understand that the meta-learning procedure helps to generalize well on multiple domains, even more than the use of the transformations. An exception is in the case of 10 iterations of adaptation, where the result is less than the OSHOT-TRANSF. One reason may be because the network, during the training phase, performs the inner-loop optimization only 5 times, and therefore learns well to adapt on 5 iterations, but this is not always the case. In fact, when we look to FUMO, all the results exceed the OSHOT by about 1.5% and the version with the transformations by about 1%. This is the confirmation that the two components combined together, *i.e.* the transformations of the visual appearance and the meta-learning procedure, give the best result if used during training. In fact, they perfectly simulate what the network actually faces in the test phase, that is samples with a different visual appearance where the network have to perform some iterations of adaption before producing the final result. The Figure 4.1 compares the adaptation phase of the OSHOT and FUMO, starting from a Comic2k sample with ground-truth bboxes that represent 2 people highlighted with the violet color. As we can see from the images, the two methods obtain the same results only in the Test phase. In fact, by comparing the bounding boxes predicted by the two models and those present in the ground-truth image, FUMO adapts faster and obtains better performances when $\gamma = 5{,}10$. This result confirms that the meta-learning and the transformations, used during the pre-training phase, push the network to adapt faster to new target samples.

Looking at Table 4.2, we can see different transformations applied to FUMO during the pre-training phase. For example, when we use Gaussian blur and inversion, we obtain transformed images that do not reflect at all those belonging to the target domains and

so in the adaptation phase we achieve lower results. Therefore, even the choice of transformations have to be examined well, in order to have images that can reflect domains that hypothetically can appear in inference phase.

Transformations applied to **FUMO** in the pre-training phase:	**Average mAP**		
	Test	$\gamma = 5$	$\gamma = 10$
Color Jitter(0.1) & Greyscale	31.37	35.18	36.26
Color Jitter(0.4) & Greyscale	**32.02**	**35.26**	**36.72**
Color Jitter(0.1) & Grayscale & Inversion	30.91	35.03	36.07
Inversion & Gaussian Blur(r=5)	29.95	33.51	34.50

Table 4.2: Results of the inference on AMDs, using different types of transformations in the pre-training phase of the FULL META OSHOT.

VOC to Social Bikes results. Social Bikes was introduced in the 3.1.1 Section and its peculiarity is that it contains images that come from different styles, poses, angle, brightness, capture methods, etc., because they are uploaded on social media by different people. Thus, the difficulty for an object detector, trained only on real images coming from the same distribution, is higher.

The results are reported in the Table 4.3. Our algorithms have been designed to improve the adaptation phase of the OSHOT, so the results we most take into account are those that do not belong to the **Test** column, which means no adaptation. In this dataset we consider only the Average Precision(**AP**) for the *Bike* and *Person* class, which are the most frequent objects inside the dataset, and for this reason we have decided to also show how the various methods perform on the individual classes.

Method	**Social Bikes**								
	Bike			Person			**mAP**		
	Test	$\gamma = 5$	$\gamma = 10$	Test	$\gamma = 5$	$\gamma = 10$	Test	$\gamma = 5$	$\gamma = 10$
OSHOT	73.95	74.81	74.96	68.60	71.64	73.26	71.27	73.23	74.11
OSHOT-TRANSF	74.00	74.70	74.62	71.63	73.04	**74.42**	72.81	73.87	74.52
META OSHOT	73.47	74.54	74.83	69.45	72.62	73.13	71.46	73.58	73.98
FUMO	**74.28**	**75.09**	**75.22**	**71.66**	**73.34**	74.37	**72.97**	**74.22**	**74.80**

Table 4.3: Results of the experiments using *Social Bikes* as target domain

Looking at the **mAP** section, simply adding transformations to the OSHOT improves the performance by about 0.5%. META-OSHOT, instead, gets a lower mAP than the OSHOT-TRANSF, in 5 and 10 iterations of adaptation. This result wants to show that the dataset has too many different domains within it that the meta-learning procedure alone is not enough to guarantee good performance, because it does not completely simulate what happens during the adaptation phase. In fact, the model *learns to adapt* only from one single domain and it is not enough to deal with the variety of other domains present in this dataset. FULL META OSHOT, instead, is the best method with performances that

exceed the baseline by about 1% and the OSHOT-TRANSF by 0.5%, confirming that if the adaptation phase is simulated perfectly, always it brings advantages to the model when used on the target samples.

Cityscapes, Foggy Cityscapes and KITTI

In this benchmark, described in the 3.1.2 Section, we analyze how our methods perform on urban and suburban scenes, even under different weather conditions. The analysis is conducted separately due to the different training configurations.

Cityscapes to KITTI & KITTI to Cityscapes results. These two dataset represent real-world inner-city traffic scenes collected in favorable weather conditions. Images are captured in different cities and with different cameras in these 2 sets. Thus, the dataset intrinsically are biased and training a model on one of them may not perform well on the other, despite the visual style homogeneity.
We investigate this benchmark in 2 directions:

1. we train the source detector on the *train* split of Cityscapes, composed by 2975 annotated images, and we evaluate the performances of our models on the 7481 images belonging to the *trainval* split of KITTI;

2. we train the source detector on the *trainval* split of KITTI and we evaluate the performances on the *validation* split of Cityscapes, composed by 500 samples.

In both the experiments we consider only the label *car* to asses the performance achieved, following standard practices in cross-domain evaluation, and for this reason the Average Precision (**AP**) metric has been used.

Method	**Cityscapes to KITTI**			**KITTI to Cityscapes**		
	AP Car			**AP** Car		
	Test	$\gamma = 5$	$\gamma = 10$	Test	$\gamma = 5$	$\gamma = 10$
OSHOT	75.01	74.99	74.98	31.22	34.08	34.46
OSHOT-TRANSF	**75.59**	**75.58**	**75.51**	**43.06**	**43.06**	**43.04**
META OSHOT	75.21	75.23	75.18	33.63	34.05	34.17
FUMO	75.43	75.41	75.34	42.88	42.95	43.01

Table 4.4: Summary of the experiments in the 2 directions.

The results are reported in Table 4.4 and in both settings the best performing method is OSHOT-TRANSF.

In the case of the *Cityscapes to KITTI* setting, the results on the *car* class are higher than those obtained in the opposite setting. This is maybe because the diversity among car instances in Cityscapes is high enough for learning a good car detection model. However the adaptation phase is not helpful, maybe because the two domains are not so different and therefore the network does not find any advantage in adapting to the various target samples. This is a problem that OSHOT already has and persists even in our newly created

methods. OSHOT-TRANSF grants a benefit over the baseline of about 0.5%, which is not too low since it is applied only in the last 10k iterations by simply adding transformations to the original OSHOT. The other two meta-learning methods are also better than the baseline, however the network does not *learn to adapt* and therefore this procedure is not so useful.

In the case of *KITTI to Cityscapes*, OSHOT-TRANSF exceed the baseline of about 12% in the test phase and of about 9% in the adaptation phase, reaching **state of the art** results in this benchmark. However, it should be notice that the adaptation phase do not work well because the results are almost the same as the test ones. FUMO also performs very well in this setting, obtaining respectively an 11% and 8% plus in the test and adaptation phase. However, the meta-learning training procedure is not sufficient to increase the performance in case of $\gamma = 5{,}10$, in fact the results are very similar to the simple test. This means that the network does not *learn to adapt*and this consideration is also confirmed by the results of the META OSHOT that are lower than the baseline.

To sum-up, this two dataset are very similar, but they are created from images captured with different cameras and in different cities. Thus, our meta-learning training procedures is not so useful because the network is already trained on a domain close to the target one and therefore it is ready to make predictions on the target samples. Instead, the addition of transformations, especially in the case of *Cityscapes to KITTI*, brings many advantages already in the test phase. This result further confirms that simple transformations, like Color-Jitter and Greyscale, help to decrease even more the *domain shift* between source and target domain.

Cityscapes to Foggy Cityscapes results. Cityscapes is a dataset collected in different seasons of the year, but with favorable weather conditions. Some special climatic events, such as fog, can be ignored in the source data acquisition, but the adaptation to these conditions is crucial for real-world applications. For this reason we want to create models that, trained only on Cityscapes, also adapt well to Foggy Cityscapes. The following analysis has different training settings than the other benchmarks, but everything is done to make fair comparison between the results:

- in the case of the OSHOT, following [2], we train the base detector for 30k iterations on the 2975 images of the *train* split of Cityscapes, using learning-rate equal to 0.001 without step-decay. Every 1000 iterations we perform a validation phase on the Cityscapes *validation* split, composed of 500 samples, to choose the best performing model that we will use for the adaptation phase;

- in our 3 methods, instead, first we train the OSHOT on Cityscapes for 25k iteration, with the same procedure described just above. Then, starting from the best model of the baseline on 25k iterations, we apply the three different algorithms for other 5k iteration, using always the train split of Cityscapes as *source* domain, without choosing the best performing model.

We have chosen this training strategy because, in this way, our methods carry out the same number of iterations of the OSHOT and we can make a fair comparison between the

results that are shown in the Table 4.5. The per-class results of each method, considering also the different number of adaptation iterations, are reported in the Table A.4 of the Appendices. In all the experiments we use 500 samples of the *validation* split of Foggy Cityscapes with the maximum degree of synthetic fog inside images, making the task even more complicated.

Method	**mAP**		
	Test	$\gamma = 5$	$\gamma = 10$
OSHOT	27.95	28.90	**29.23**
OSHOT-TRANSF	27.96	28.49	28.37
META OSHOT	25.10	27.05	27.91
FUMO	**28.77**	**29.01**	29.20

Table 4.5: Summary of the experiments on Foggy Cityscapes dataset.

FUMO is the method that best performs in this setting, even if, in the adaptation phase, it slightly exceeds the baseline or assumes similar values. The OSHOT-TRASF and META-OSHOT, instead, do not obtain good results, assuming values below the OSHOT. This makes us understand that, in most cases, only considering transformations or the meta learning procedure is not enough to create a model capable of adapting well to the target samples. FUMO, on the other hand, thanks to its training procedure, boost the model *to learn to adapt* to each single sample in the last 5k iterations. This perfect match between training and test conditions[6] guarantees the creation of a model that can only be improved.

In this setting we have tried different transformations in the FULL META OSHOT, reported in the Table 4.6. In particular, we thought that the Gaussian Blur transformation could help the algorithm to perform better on adverse weather conditions, because blur can somehow be considered close to the case in which there is fog in the image. However, the model is only disturbed by the use of this single transformation, concluding that the blur cannot be assimilated to a hypothetical fog condition. We also have used Blur in conjunction with other transformations, such as Color Jitter or Inversion, but, even there, the results are not the best.

Transformations applied to **FUMO** in the pre-training phase:	**mAP**		
	Test	$\gamma = 5$	$\gamma = 10$
Color Jitter(0.1) & Greyscale	27.65	28.42	**29.27**
Color Jitter(0.4) & Greyscale	**28.77**	**29.01**	29.20
Color Jitter(0.1) & Gaussian Blur(r=5)	26.31	27.13	28.00
Inversion & Gaussian Blur(r=5)	28.51	28.92	28.77
Gaussian Blur(r=5)	24.10	23.65	24.07

Table 4.6: Results of the inference on Foggy Cityscapes, using different types of transformations in the pre-training phase of the FULL META OSHOT.

4.2.4 Detection error analysis

The metric used, in most benchmarks, to compare object detection models is the **mAP**. The problem with this metric is that it does not explain *why* one method is better than another, but produces a single value that the higher the model performs better. Not only that, it does not even provide us with indications on what produces the most errors in the prediction, in order to improve the performance of the algorithms. For these reasons, the authors of [47] have carried out an extensive analysis to understand the frequency and impact of different types of false positives and have provided guidelines in order to better categorize them.

Our goal is to analyse the detection errors to investigate the positive impact of our meta-learning method in the One-Sample Adaptation phase, comparing the results obtained by FUMO and OSHOT in the 5 and 10 adaptation steps of the rotation task. In fact, for all the benchmarks analysed in the previous sections, we categorize the detection outputs into three groups, following [47]:

- **correct detection**, where the prediction has the same class of the ground-truth object and Intersection over Union(**IoU**) grater or equal than 0.5;
- **mislocalization error**, where the prediction has the same class of the ground-truth object and IoU between 0.3, included, and 0.5;
- **background error**, where either the prediction has wrong class with the ground-truth object or it has correct class and IoU less than 0.3.

The analysis is not carried out on all the predictions of the model, but on a subset of the most confidence detections, following [2]: for VOC to Clipart we have chosen the 1k most confidence detections, 2k for VOC to Comic and VOC to Watercolor, and 1060 for VOC to Social Bikes. For the benchmarks that deal with urban scenes we have chosen: 6k most confidence predictions for Cityscapes to Foggy Cityscapes, 1.5k for KITTI to Cityscapes and 20k for Cityscapes to KITTI.

From Figure 4.2 we can state that for Comic, Clipart and Social Bikes, FUMO achieves better results than the OSHOT, both in the case of 5 and 10 iterations of adaptation, with a decreasing trend in false positives classified as background from 5 to 10 iterations. In the case of Watercolor, on the other hand, OSHOT obtains the best results and FUMO, instead, achieves similar performances in the two different number of iterations, increasing the background error. Overall, from 5 to 10 iterations of adaptation, the ratio between the mislocalization error and correct localization either decreases or remains stable in both methods, while the background errors decrease. A similar behavior also occurs in the first two rows of Figure 4.3, where, when testing on Foggy Cityscapes and Cityscapes, FUMO is again the best method. For KITTI, instead, the two models achieve similar performances with low errors, confirming the results previously obtained in this setting.

To sum-up, FUMO works better than OSHOT in most of these dataset, achieving a lower margin of error and higher localization accuracy, confirming that the meta-learning procedure has a positive impact in the model adaptation phase.

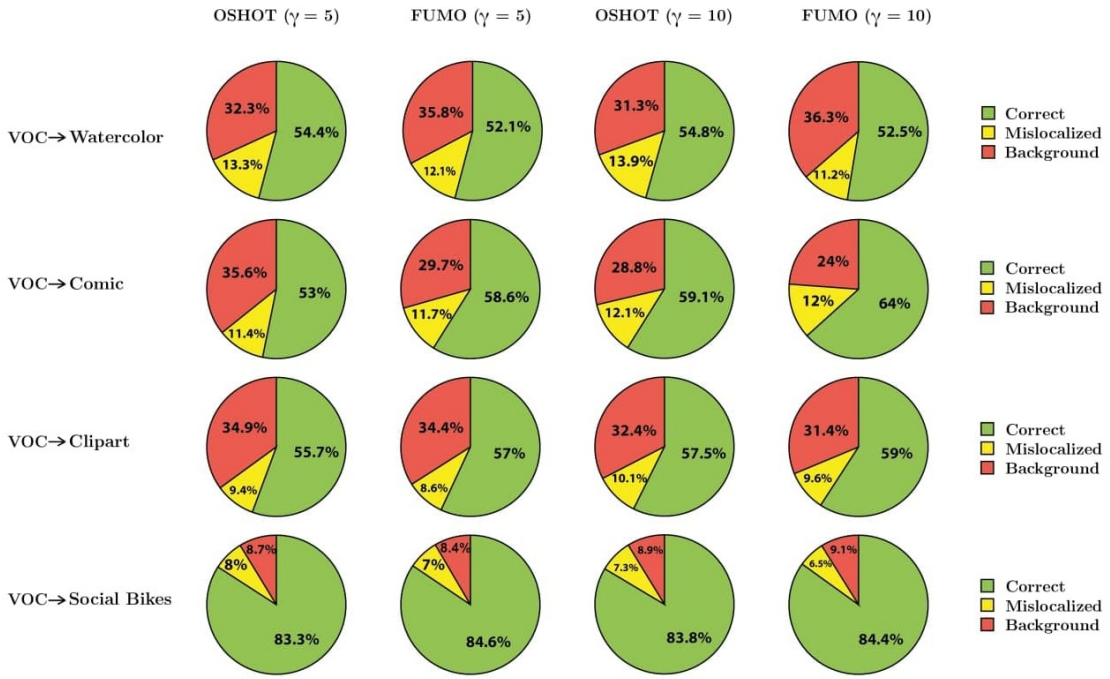


Figure 4.2: Detection error analysis on VOC to AMDs and Social Bikes.

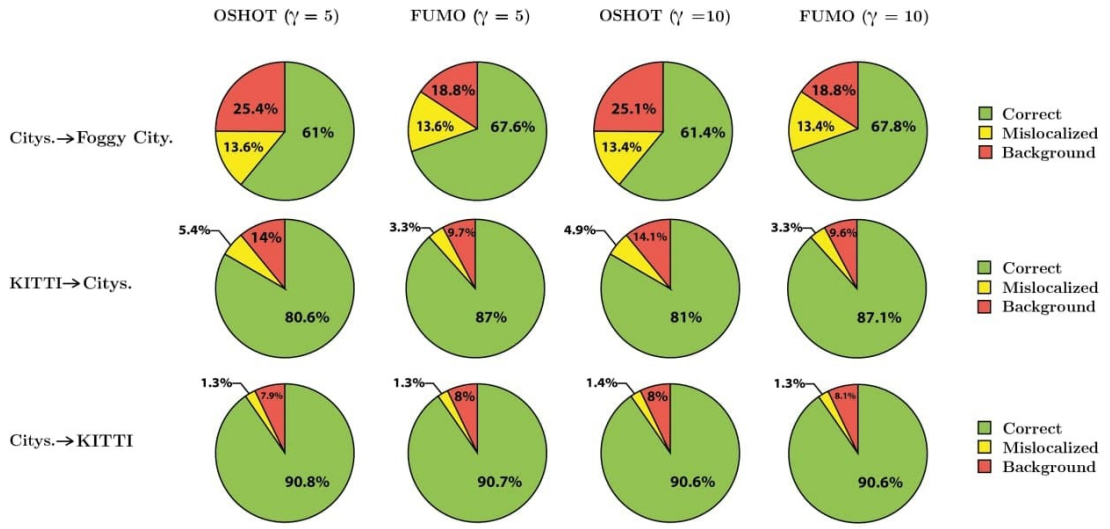


Figure 4.3: Detection error analysis on the three cases of urban scenes.

4.3 1% target adaptation setting

The One-Sample Adaptation is a new setting introduced by the OSHOT, so it is difficult to compare our methods with others on the same benchmarks, in the case of the cross-domain analysis.

It is true that *Domain Generalization* methods do not use the target domains during the training phase, but they impose that the trained model makes only the inference on the target samples during the test phase, without exploiting any finetuning steps on them. Thus, the comparison between our methods and those belonging to **DG** cannot be fair.

Domain Adaptation methods, instead, use part of the target samples during the training phase to adapt to the new domain and to be more efficient during the testing phase. In this case the procedure is far away from that followed by the One-Sample Adaptation, but we can exploit this form of adaptation. In fact, the main idea is to compare FUMO with **DA** methods, imposing that, during the training phase only a small batch of samples belonging to the target domain are available to carry out their training procedure. More specifically, the idea is to give access to only 1% of the target samples, since it is impossible to completely remove them, because the training procedure of the DA models is focused on exploiting these samples and there is not a separate procedure where these methods can extract information from the targets, like OSHOT does. In this way we do not really bring the DA methods into our setting, but we want to show how their performance decreases when they have a limited set of data available to adapt during the training phase. In the case of the One-Sample Adaptation strategy, instead, this limitation does not affects the final performances, because the model does not use target samples in the training phase, but it exploit one sample at a time to adapt itself in an unsupervised way only in the adaptation phase.

4.3.1 Description of the methods

Here we present the two DA methods used in the various tables. They will be compared with the FULL META OSHOT(**FUMO**), already presented in the One-Sample Adaptation setting.

CST-DA-Detection. A classic two-stage detector, like the Faster R-CNN[3], is composed of 3 main components: the backone, RPN and ROI Head. The backbone is the common part of the architecture from which 2 branches start, representing the other 2 components. This means that ROI Head and RPN train them-self independently, affecting only the backbone, but the RPN output is used by the ROI Head influencing its performance. Starting from these observations, CST-DA-Detection[7] is the first work that shows how these two components have a different transferability when dealing with a large domain gap. Therefore, it is not enough to align and adapt only the global features of the backbone to reduce the domain shift between source and target domain.
Their solution for unsupervised domain-adaptive object-detection is based on:

- a collaborative training between the RPN and the ROI Head, where the high confidence output of one is used to train the other and vice versa. The confidence output is calculated on the unlabelled samples coming from the target domain;
- a domain classifier that, starting from the backbone output, focuses only on anchors with high foreground confidence produced by the RPN. In this way, it obtain cross-domain features alignment and complement the collaborative training;
- a customized maximum discrepancy classifier (**MCD**), that uses low confidence RoIs to calculate foreground/background discrepancy between RPN and ROI Head. To perform this task, the backbone is trained to minimize the discrepancy and instead RPN and ROI Head try to maximize it. This technique is used to improve the transferability of the model.

SW-Faster-ICR-CCR

ICR-CCR. The main goal of domain adaptive object detection is to reduce the gap between source and target domain. Many methods try to alleviate the problem by performing a fully alignment of both the predicted instances contained in the images and the images themselves. The problem with these models is that they also consider parts of the images, such as the background, that are not easily transferable between domains, decreasing the generalization capability of the network. To overcome this problem, the authors of ICR-CCR[8] propose a categorical regularization framework that can be applied as a plug-and-play component to assist Domain Adaptive Faster R-CNN methods. Their idea is to focus on aligning the crucial regions and important instances across domains. Therefore, they introduce two new regularization modules into the object detector:

- image-level categorical regularization(**ICR**), where immediately after the backbone the authors introduce an image-level multi label classifier that is trained with categorical supervision from the source domain. This enables the backbone to learn concepts of the objects contained in the images, since it is not affected by any noise from the source background, allowing to align the crucial regions of both the source and target domain;
- categorical consistency regularization(**CCR**), where the authors take into account the consistency between the prediction of both the image-level multi label classifier and the instance-level classifier of the detector as a metric for the hardness of classifying a certain target sample. This metric will be used as a regularization factor to increase the weight of the hard aligned samples in the target domain during instance-level alignment.

SW-Faster-ICR-CCR. ICR-CCR is a plug-and-play framework so, in order to be used, it have to be integrated into some other models. The authors use Strong-Weak[48] as a baseline to integrate their algorithm. It is a method belonging to the Domain Adaptive Faster R-CNN series, which uses Gradient Reversal Layers (GRLs) and adversarial learning to obtain a strong alignment between lower level features and a weak alignment between higher level features.

4.3.2 Experimental Setup

The methods just described use settings that are different from those followed by FUMO. In order to fairly compare the results, we have modified their code to make their settings similar to ours. We have performed three runs of each method to get more reliable results.

CST-DA-Detection

Ganlong et al.[7] present their method using an implementation of the Faster RCNN with a VGG16 backbone. We have downloaded their code and rerun the experiments using a ResNet-50 backbone to align their results with our. All other hyperparameters are the same of their training procedure. For some benchmarks we have implemented whole pieces of code, because they were not completely present in the repository. To do this correctly, we were inspired by the parts already present, asking the authors for advice.

SW-Faster-ICR-CCR

Chang-Dong et al.[8] use a different backbone for the Faster R-CNN. More specifically, they use a VGG16 for some benchmarks and a ResNet101 for others. Thus, we downloaded their implementation, which is publicly available, and implemented a piece of code to add support for the ResNet50 backbone, in order to have a fair comparison with our method. All other settings and hyperparameters are the same of those described in the paper's experimental setup.

4.3.3 Detailed results and analysis

The analysis is carried out on 3 different experimental settings that are common to the three methods on which we will make the comparison. Note that, in the tables there are three sections: the **One-Shot** target section which refers to the One-sample Adaptation setting followed by our method, while **Ten-Shot** target and **Full** target refer to the case in which the DA methods respectively have access to ten or all the target samples during the training phase.

VOC to Clipart1k setting. In this case we use the *trainval* split of Pascal VOC 2007 and 2012 sets as source domain and we evaluate the results on the *test* split of Clipart1k. We compare the performances obtained by our method only with SW-Faster-ICR-CCR because CST-DA has no support for this setting and when we train the model with Full Target, we use all the 1000 images of Clipart1k, following the paper settings[8]. Looking at Table 4.7 we can see that FUMO, with 5 and 10 iterations of adaptation, far exceeds ICR-CCR both when it has access to only 10 target samples and when it has access to the entire target. Furthermore, ICR-CCR achieves 3% lower results when it deal with a small number of target samples during the training phase, underlining that many DA methods are not designed to tackle data scarcity conditions. Thus, we can conclude that the adaptation procedure followed by FUMO, both in the train and test phase, is a strategy that allows to further decrease the domain shift between source and target, without the need of prior knowledge on the test distribution. The performances per each class of Clipart1k are reported in the Table B.2 of the Appendices.

One-Shot Target	
Method	**mAP**
FUMO ($\gamma = 0$)	28.62
FUMO ($\gamma = 5$)	31.66
FUMO ($\gamma = 10$)	**31.74**
Ten-Shot Target	
SW-ICR-CCR[8]	27.09
Full Target	
SW-ICR-CCR[8]	30.25

Table 4.7: Results of the experiments using Pascal VOC as source domain and Clipart1k as target domain.

Cityscapes to Foggy Cityscapes setting. Here we use the *train* split of Cityscapes as source domain and we evaluate the results on the *validation* split of Foggy Cityscapes. When we train the 2 DA models with Full Target, we use all the *train* split of the target domain. The experiments are carried out always considering the Foggy Cityscapes splits with the highest percentage of synthetic fog inside the images. Looking at the 10-Shot target results in Table 4.8, our method, with $\gamma = 10$, slightly exceeds ICR-CCR, but the best results are achieved by CST-DA. This method assumes almost similar values with the case in which the entire target is available, which means that, with this algorithm and in this setting, 10 samples are enough to generalize well on the target domain. SW-ICR-CCR, instead, loses 7% of performance when it has access to only 10 target samples in the training phase, highlighting again that some DA methods are not optimal in conditions where the available data are limited. To conclude, the results obtained by FUMO are very promising, because it uses, only in the test phase, one target at a time to adapt itself, so it is much more disadvantaged than the other two methods, that use 9 more samples. The performances per each class of Foggy Cityscapes are in the Table B.1 of the Appendices.

One-Shot Target	
Method	**mAP**
FUMO ($\gamma = 0$)	28.78
FUMO ($\gamma = 5$)	29.01
FUMO ($\gamma = 10$)	29.20
Ten-Shot Target	
SW-ICR-CCR[8]	29.05
CST-DA-Det[7]	**33.27**
Full Target	
SW-ICR-CCR[8]	36.53
CST-DA-Det[7]	33.98

Table 4.8: Results of the experiments using Cityscapes as source domain and Foggy Cityscapes as target domain

KITTI to Cityscapes setting. The last setting considers the *trainval* split of KITTI as source domain and the results are calculated on the *validation* split of Cityscapes. Here the objective is to compare the degree of generalization obtained by different object-detectors on the *Car* category. When we train the 2 DA models with Full Target, we use the whole *train* split of Cityscapes. Looking at the results in Table 4.9, FUMO is the best model with AP equal to 43%, exceeding by 8% CST-DA and 3% ICR-CCR in the case of Ten-Shot target. If we consider also the case in which the two models have access to all the target samples, we can see than the performances are lower than FUMO. Thus, our algorithm, that combines meta-learning and the use of transformations, reaches an higher degree of generalization without any information of the target domain during the training phase, leading to **state-of-the-art** results in this setting.

One-Shot Target	
Method	**AP Car**
FUMO ($\gamma = 0$)	42.88
FUMO ($\gamma = 5$)	42.95
FUMO ($\gamma = 10$)	**43.01**
Ten-Shot Target	
SW-ICR-CCR[8]	39.7
CST-DA-Det[7]	35.25
Full Target	
SW-ICR-CCR[8]	39.79
CST-DA-Det[7]	37.28

Table 4.9: Results of the experiments using KITTI as source domain and Cityscapes as target domain

4.4 Final considerations

The results of the various experiments confirm that our meta-learning procedure, applied to the OSHOT, allows to adapt more quickly to the various samples.

We started from the idea of perfectly simulate the adaptation phase of the OSHOT during training, so we coded the logic of meta-learning within the self-supervised rotation task, creating the META OSHOT. This new method produces promising results in all benchmarks, but there is still something missed to be fully compliant with the One-Sample Adaptation setting. Thus, we introduced the transformation of the visual appearance on samples belonging to the source domain, in order to make the model aware that the data, on which it have to make the inference, may belong to a different distribution than the training one. The use of transformations has brought significant benefits also to the simple OSHOT, increasing the final performance in the different settings.

In the end we created the FULL META OSHOT by combining self-supervision, meta-learning and transformations of the visual appearance. It exceeds the performance of both the baseline and the two DA methods in most of the benchmarks, establishing the new **state-of-the-art** in detection for social media monitoring and autonomous driving scenarios.

Chapter 5

Conclusions

In this master thesis work we investigated the effectiveness of applying a meta-learning algorithm to an object-detector that works for cross-domain analysis. This detector[2] already exploited a self-supervised rotation task both in the training phase to extrapolate features that are domain-invariant and in the adaptation phase to exploit the characteristics contained in each new target sample in an unsupervised way.
Our new reformulation, on the other hand, pushes the network to adapt faster to new domains, because by implementing the logic of **MAML**[5] in the self-supervised task, we force the model to *learn to adapt* to one image at a time already in the training phase, perfectly reflecting what the model has to face in the adaptation phase. In fact, this new combination of meta-learning and self-supervision has achieved **state-of-the-art** results in almost all benchmarks, ensuring better performance with respect to **OSHOT**[2].

With various experiments we prove the effectiveness of the proposed algorithm also in the Domain Adaptation setting. In this case the performances of our model are better than those obtained by previous state-of-the-art Domain Adaptation algorithms[8][7], even if our method can look at an overall lower number of samples belonging to the target domain. This analysis shows us that DA methods lose a lot in performance, if they work on a small number of target samples during the training phase. However, the most important result is that **One-Sample Adaptation**, the new cross domain analysis research setting introduced by the authors of [2] and used in our method, is more general and useful than the Domain Adaptation setting. In fact, even if the model, during the training phase, does not have access to any information about the distribution of target samples, before the final prediction it can extrapolate relevant information from each single target image in an unsupervised way.

There are several features that can be improved in **future works**, for instance during the integration of meta-learning in[2], we had to make changes to the logic of MAML, because our setting does not include meta-tasks. Thus, we can investigate a new way to create a distribution of tasks from the source domain that can be integrated into our setting, in order to be as close as possible to the meta-learning principles and improving performances on new tasks. Another interesting thing to analyze is a new article[49] that proposes modifications to MAML that not only stabilize the overall architecture, but also

improve the generalization performance, convergence speed and computational overhead of the method of *Finn et al.*[5]. Referring instead to the use of transformations during meta-training, the next step can be to use more complex techniques, such as GAN[18] methods, to generate source images that more closely resemble the target domains because, in general, simple transformations of the visual appearance are not enough to reduce the *domain gap*. Finally, we can investigate new meta-learning methods[50][51] that try to solve the problems of the gradient-based meta-learning techniques, like MAML, which have practical difficulties when operating on high-dimensional parameter spaces in extreme low-data regimes.

Bibliography

[1] L. Jing and Y. Tian. "Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey". In: *TPAMI* (2020), pp. 1–1. DOI: 10.1109/TPAMI.2020.2992393.

[2] Antonio D'Innocente et al. "One-Shot Unsupervised Cross-Domain Detection". In: *Computer Vision – ECCV 2020* (2020). arXiv: 2005.11610.

[3] S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[4] Timothy Hospedales et al. "Meta-Learning in Neural Networks: A Survey". In: (2020). arXiv: 2004.05439.

[5] C. Finn, S. Levine, and P. Abbel. "Model-Agnostic Meta-learning For Fast Adaptation Of Deep Networks". In: *ICML* (2017). arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

[6] Oriol Vinyals et al. "Matching Networks for One Shot Learning". In: *NIPS* 29 (2016). Ed. by D. Lee et al., pp. 3630–3638. URL: https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf.

[7] Ganlong Zhao et al. "Collaborative Training between Region Proposal Localization and Classification for Domain Adaptive Object Detection". In: *European Conference on Computer Vision (ECCV)* (2020). arXiv: 2009.08119.

[8] C. -D. Xu et al. "Exploring Categorical Regularization for Domain Adaptive Object Detection". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 11721–11730. DOI: 10.1109/CVPR42600.2020.01174.

[9] Sha Qian. "Real-time Object Detection in Flutter". In: *Noteworthy The Journal Blog* (2019). URL: https://blog.usejournal.com/real-time-object-detection-in-flutter-b31c7ff9ef96.

[10] Jason Brownlee. "A Gentle Introduction to Object Recognition With Deep Learning!" In: (2019). URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/.

[11] R. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

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

[13] K. He et al. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[14] J. Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[16] Mancini and Morerio. "Transferring Knowledge Across Domains: an Introduction to Deep Domain Adaptation". In: *20th International Conference on IMAGE ANALYSIS AND PROCESSING* (2019).

[17] Y. Ganin et al. "Domain-Adversarial Training of Neural Networks". In: *Advances in Computer Vision and Pattern Recognition* (2017). DOI: 10.1007/978-3-319-58347-1_10.

[18] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *NIPS* (2014). arXiv: 1406.2661.

[19] Konstantinos Bousmalis et al. "Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pp. 95–104. DOI: 10.1109/CVPR.2017.18.

[20] Fabio Maria Carlucci et al. "Just DIAL: DomaIn Alignment Layers for Unsupervised Domain Adaptation". In: (2017). Ed. by Sebastiano Battiato et al., pp. 357–369. DOI: 10.1007/978-3-319-68560-1_32. URL: https://doi.org/10.1007/978-3-319-68560-1_32.

[21] Da Li et al. "Learning to Generalize: Meta-Learning for Domain Generalization". In: (2017). arXiv: 1710.03463 [cs.LG].

[22] M. Mancini et al. "Best Sources Forward: Domain Generalization through Source-Specific Nets". In: *25th IEEE International Conference on Image Processing (ICIP)* (2018). DOI: 10.1109/ICIP.2018.84513180.

[23] D. Li et al. "Deeper, Broader and Artier Domain Generalization". In: *IEEE International Conference on Computer Vision (ICCV)* (2017). DOI: 10.1109/ICCV.2017.591.

[24] F. M. Carlucci et al. "Domain Generalization by Solving Jigsaw Puzzles". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). DOI: 10.1109/CVPR.2019.00233.

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

[26] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. "A critical analysis of self-supervision, or what we can learn from a single image". In: *ICLR* (2020). arXiv: 1904.13132. URL: http://arxiv.org/abs/1904.13132.

[27] Richard Zhang, Phillip Isola, and Alexei A. Efros. "Colorful Image Colorization". In: ed. by Bastian Leibe et al. 2016, pp. 649–666. ISBN: 978-3-319-46487-9. DOI: 10.1007/978-3-319-46487-9_40.

[28] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised Representation Learning by Predicting Image Rotations". In: *ICLR* (2018). arXiv: 1803.07728. URL: http://arxiv.org/abs/1803.07728.

[29] Gary Marcus. "Deep Learning: A Critical Appraisal". In: *arXiv e-prints* (2018).

[30] C. Finn and S. Levine. "Meta-Learning: from Few-Shot Learning to Rapid Reinforcement Learning". In: *ICML 2019 Meta-Learning Tutorial* (2019). URL: https://sites.google.com/view/icml19metalearning.

[31] S. Thrun and L. Pratt. "Learning to Learn: Introduction and Overview". In: *Learning to Learn* (1998). DOI: 10.1007/978-1-4615-5529-2_1.

[32] G. Kosh, R. Zemel, and R. Salakhutdinov. "Siamese Neural Networks For One-shot Image Recognition". In: *ICML* (2015).

[33] Zeyi Huang et al. "Self-challenging Improves Cross-Domain Generalization". In: *ECCV 2020* (Nov. 2020), pp. 124–140. DOI: 10.1007/978-3-030-58536-5_8.

[34] Y. Jing et al. "Neural Style Transfer: A Review". In: *IEEE Transactions on Visualization and Computer Graphics* 26.11 (2020), pp. 3365–3385. DOI: 10.1109/TVCG.2019.2921336.

[35] "Torchvision transformations". In: *Pytorch project* (). URL: https://pytorch.org/vision/stable/transforms.html.

[36] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88 (2009), pp. 303–338. DOI: 10.1007/s11263-009-0275-4.

[37] Naoto Inoue et al. "Cross-Domain Weakly-Supervised Object Detection through Progressive Domain Adaptation". In: *CVPR 2018* abs/1803.11365 (2018). arXiv: 1803.11365. URL: http://arxiv.org/abs/1803.11365.

[38] G. Brostow, J. Fauqueur, and R. Cipolla. "Semantic object classes in video: A high-definition ground truth database". In: *Pattern Recognit. Lett.* 30 (2009), pp. 88–97.

[39] M. Enzweiler et al. "Efficient multi-cue scene segmentation". In: *GCPR* (2013).

[40] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *CVPR 2016* abs/1604.01685 (2016). arXiv: 1604.01685. URL: http://arxiv.org/abs/1604.01685.

[41] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. "Semantic Foggy Scene Understanding with Synthetic Data". In: *IJCV* abs/1708.07819 (2017). arXiv: 1708.07819. URL: http://arxiv.org/abs/1708.07819.

[42] A. Geiger et al. "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* (2013). DOI: 10.1177/0278364913491297.

[43] A. Paszke et al. "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop* (2017).

[44] Antonio D'Innocente et al. "Demo implementation of OSHOT: One SHOT unsupervised cross domain detection". In: (2020). URL: https://github.com/VeloDC/oshot_detection.

[45] F. Massa and R. Girshick. "Maskrcnn-Benchmark: Fast, Modular Reference Implementation of Instance Segmentation and Object Detection Algorithms in PyTorch". In: (2018). URL: https://github.com/facebookresearch/maskrcnn-benchmark..

[46] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CVPR* (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[47] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. "Diagnosing Error in Object Detectors". In: *Computer Vision – ECCV* (2012). Ed. by Andrew Fitzgibbon et al., pp. 340–353.

[48] K. Saito et al. "Strong-Weak Distribution Alignment for Adaptive Object Detection". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 6949–6958. DOI: 10.1109/CVPR.2019.00712.

[49] Antreas Antoniou, Harrison Edwards, and Amos J. Storkey. "How to train your MAML". In: *ICLR* (2019). arXiv: 1810.09502. URL: http://arxiv.org/abs/1810.09502.

[50] Sebastian Flennerhag et al. "Meta-Learning with Warped Gradient Descent". In: *International Conference on Learning Representations* (2020). URL: https://openreview.net/forum?id=rkeiQlBFPB.

[51] Andrei A. Rusu et al. "Meta-Learning with Latent Embedding Optimization". In: *International Conference on Learning Representations* (2019). URL: https://openreview.net/forum?id=BJgklhAcK7.

[52] W. Liu et al. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2.

[53] Julia Angwin at al. "Machine Bias". In: *ProPublica* (2016). URL: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

Appendices

Appendix A

One-Sample Adaptation: per-class results

Method	bike	bird	car	cat	dog	person	**mAP**
OSHOT ($\gamma = 0$)	27.18	11.40	18.54	10.10	13.49	27.77	18.08
OSHOT ($\gamma = 5$)	34.44	14.12	21.51	11.14	18.13	35.94	22.55
OSHOT ($\gamma = 10$)	37.40	13.28	25.37	13.74	21.44	40.93	25.36
OSHOT-TRANSF ($\gamma = 0$)	32.64	11.78	20.91	9.09	14.40	31.73	20.09
OSHOT-TRANSF($\gamma = 5$)	35.75	13.72	27.49	12.25	20.91	39.47	24.93
OSHOT-TRANSF ($\gamma = 10$)	39.52	13.00	29.84	**15.25**	24.66	45.20	27.91
META OSHOT ($\gamma = 0$)	33.38	11.24	25.51	10.13	12.91	27.89	20.18
META OSHOT ($\gamma = 5$)	40.15	13.16	27.29	11.15	17.87	39.01	24.77
META OSHOT ($\gamma = 10$)	37.48	14.67	31.06	12.77	19.76	44.25	26.67
FUMO ($\gamma = 0$)	32.21	12.75	23.35	10.85	14.22	33.11	**21.08**
FUMO ($\gamma = 5$)	33.69	13.84	29.94	12.93	20.09	40.72	**25.20**
FUMO ($\gamma = 10$)	**40.06**	**15.23**	**32.22**	13.10	**24.76**	**46.25**	**28.60**

Table A.1: Results of the experiments using VOC as source domain and Comic2k as target domain

Method	bike	bird	car	cat	dog	person	**mAP**
OSHOT ($\gamma = 0$)	70.20	45.57	45.91	30.36	25.76	56.63	45.74
OSHOT ($\gamma = 5$)	70.56	45.70	50.18	31.99	31.57	59.45	48.24
OSHOT ($\gamma = 10$)	74.33	42.85	53.23	30.18	30.39	59.79	48.46
OSHOT-TRANSF ($\gamma = 0$)	72.03	44.71	47.83	30.33	23.96	53.34	45.37
OSHOT-TRANSF($\gamma = 5$)	71.86	41.72	**53.78**	31.08	30.81	57.07	47.72
OSHOT-TRANSF ($\gamma = 10$)	**79.15**	40.89	52.55	29.76	29.90	59.73	48.67
META OSHOT ($\gamma = 0$)	66.40	46.80	47.32	33.17	26.07	54.88	45.78
META OSHOT ($\gamma = 5$)	69.34	**47.58**	51.85	**36.28**	30.86	58.32	**49.04**
META OSHOT ($\gamma = 10$)	68.29	46.68	50.10	35.40	**32.85**	59.49	48.80
FUMO ($\gamma = 0$)	71.92	45.52	48.15	34.26	22.78	55.53	**46.36**
FUMO ($\gamma = 5$)	74.50	45.60	50.67	34.55	28.55	59.74	48.93
FUMO ($\gamma = 10$)	77.18	45.05	52.28	32.09	31.62	**60.75**	**49.83**

Table A.2: Results of the experiments using VOC as source domain and Watercolor2k as target domain

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	**mAP**
OSHOT ($\gamma = 0$)	21.69	49.27	22.05	18.20	28.83	54.26	31.00	6.57	32.04	10.19	**29.00**	09.34	29.24	46.16	37.64	34.04	10.82	14.02	35.59	29.99	27.50
OSHOT ($\gamma = 5$)	25.73	53.32	24.70	20.33	33.31	53.73	34.11	06.26	34.98	12.78	26.27	07.74	33.30	60.28	44.82	40.98	14.62	16.52	39.71	35.10	30.93
OSHOT ($\gamma = 10$)	25.55	**54.33**	25.51	19.61	**34.91**	48.61	33.53	05.96	36.39	13.25	24.67	8.75	37.83	**75.78**	50.80	**42.74**	16.71	12.07	38.61	29.40	**31.75**
OSHOT-TRANSF ($\gamma = 0$)	22.35	49.27	23.83	17.97	31.41	**63.00**	28.71	**06.71**	32.76	13.54	26.93	09.09	29.61	51.41	39.97	34.03	12.45	12.84	34.18	32.47	28.63
OSHOT-TRANSF($\gamma = 5$)	22.77	48.37	26.12	17.62	33.40	55.60	30.78	04.29	**39.18**	14.48	25.06	09.09	35.57	62.72	49.05	34.95	16.13	11.16	39.28	34.55	30.51
OSHOT-TRANSF ($\gamma = 10$)	22.27	51.01	27.24	18.35	33.77	51.76	32.77	06.38	38.15	14.97	23.53	**12.01**	**40.73**	71.79	51.45	35.54	**17.35**	08.29	37.61	32.47	31.37
META OSHOT ($\gamma = 0$)	24.89	48.12	23.62	21.56	30.71	55.52	30.40	06.06	34.49	14.11	27.31	09.09	29.95	50.83	39.34	37.73	07.88	20.59	38.08	36.75	**29.35**
META OSHOT ($\gamma = 5$)	29.42	52.88	24.41	**22.94**	33.09	54.70	32.59	04.29	37.24	16.89	27.65	03.87	32.86	52.15	48.52	41.05	13.02	**20.88**	40.12	38.80	31.37
META OSHOT ($\gamma = 10$)	**32.72**	52.07	24.82	21.18	34.30	44.11	32.11	03.74	35.87	14.42	28.36	05.33	38.53	57.48	50.81	40.89	15.74	16.93	**41.34**	**39.64**	31.52
FUMO ($\gamma = 0$)	21.26	47.35	24.52	20.96	32.24	52.13	30.49	06.06	36.07	15.01	26.05	04.11	31.79	53.27	42.97	36.58	06.70	15.52	33.54	35.80	28.62
FUMO ($\gamma = 5$)	28.67	49.63	27.92	22.91	30.98	57.28	32.52	05.05	36.16	**19.76**	28.05	3.13	36.80	66.12	50.18	39.63	13.62	14.07	34.98	35.74	**31.66**
FUMO ($\gamma = 10$)	28.25	50.33	**28.61**	22.70	32.89	39.08	**34.42**	05.05	36.51	19.35	26.08	07.26	40.68	69.89	**53.22**	40.66	16.75	12.98	35.04	35.17	**31,75**

Table A.3: Results of the experiments using VOC as source domain and Clipart1k as target domain

Method	person	rider	car	truck	bus	train	mcycle	bicycle	**mAP**
OSHOT ($\gamma = 0$)	32.23	38.39	38.50	20.89	30.01	07.58	24.06	31.98	27.95
OSHOT ($\gamma = 5$)	33.40	38.48	40.69	**21.41**	32.22	08.56	**24.11**	32.35	28.91
OSHOT ($\gamma = 10$)	**34.39**	38.48	42.67	21.14	31.84	08.42	23.86	33.09	**29.24**
OSHOT-TRANSF ($\gamma = 0$)	30.45	37.45	42.66	16.87	29.49	**14.50**	21.89	30.37	27.96
OSHOT-TRANSF($\gamma = 5$)	30.89	38.49	42.95	17.52	32.10	13.93	21.56	30.49	28.49
OSHOT-TRANSF ($\gamma = 10$)	31.19	38.46	43.08	15.86	32.27	13.94	21.31	30.83	28.37
META OSHOT ($\gamma = 0$)	30.56	35.13	35.86	16.60	28.45	07.58	18.20	28.44	25.10
META OSHOT ($\gamma = 5$)	32.13	38.15	39.88	17.39	30.93	07.53	21.01	29.23	27.03
META OSHOT ($\gamma = 10$)	32.58	38.66	42.16	18.07	**33.59**	07.79	20.40	30.01	27.91
FUMO ($\gamma = 0$)	31.69	40.79	43.68	18.28	28.82	10.95	22.79	**33.25**	**28.78**
FUMO ($\gamma = 5$)	31.96	39.74	43.85	18.75	31.80	10.65	22.13	33.20	**29.01**
FUMO ($\gamma = 10$)	31.99	**39.96**	**43.97**	17.98	32.59	12.47	21.51	33.13	29.20

Table A.4: Results of the experiments using Cityscapes as source domain and Foggy Cityscapes as target domain

Appendix B

1% Target Adaptation: per-class results

Method	person	rider	car	truck	bus	train	mcycle	bicycle	**mAP**
One-Shot Target									
FUMO ($\gamma = 0$)	31.69	40.79	43.68	18.28	28.82	10.95	22.79	33.25	28.78
FUMO ($\gamma = 5$)	31.96	39.74	43.85	18.75	31.80	10.65	22.13	33.20	29.01
FUMO ($\gamma = 10$)	31.99	39.96	43.97	17.98	32.59	12.47	21.51	33.13	29.20
Ten-Shot Target									
SW-ICR-CCR[8]	29.64	40.76	39.64	20.53	32.81	11.06	23.96	34.03	29.05
CST-DA-Det[7]	31.08	43.75	44.82	22.44	41.38	**27.13**	23.24	32.32	**33.27**
Full Target									
SW-ICR-CCR[8]	**32.73**	**45.10**	**49.68**	**29.21**	**46.70**	25.95	**26.33**	**36.50**	36.53
CST-DA-Det[7]	31.73	45.09	46.55	21.13	41.26	29.52	23.51	33.00	33.98

Table B.1: Results of the experiments using Cityscapes as source domain and Foggy Cityscapes as target domain

Method	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	**mAP**
One-Shot Target																					
FUMO ($\gamma = 0$)	21.26	47.35	24.52	20.96	32.24	52.13	30.49	**06.06**	36.07	15.01	26.05	04.11	31.79	53.27	42.97	36.58	06.70	**15.52**	33.54	**35.80**	28.62
FUMO ($\gamma = 5$)	**28.67**	49.63	27.92	**22.91**	30.98	**57.28**	32.52	05.05	36.16	19.76	**28.05**	03.13	36.80	66.12	50.18	39.63	13.62	14.07	34.98	35.74	31.66
FUMO ($\gamma = 10$)	28.25	**50.33**	**28.61**	22.70	**32.89**	39.08	**34.42**	05.05	**36.51**	19.35	26.08	07.26	40.68	**69.89**	53.22	**40.66**	16.75	12.98	**35.04**	35.17	**31.75**
Ten-Shot Target																					
SW-ICR-CCR[8]	20.46	46.69	20.81	16.93	25.23	37.17	27.18	3.81	33.99	20.54	15.16	6.65	38.54	48.98	50.61	33.27	25.27	10.68	28.75	30.95	27.09
Full Target																					
SW-ICR-CCR[8]	16.10	47.90	24.13	21.48	29.58	33.12	32.10	3.90	34.12	**31.77**	17.55	**8.46**	**42.65**	61.21	**60.82**	37.14	**25.95**	15.34	30.58	31.10	30.25

Table B.2: Results of the experiments using VOC as source domain and Clipart1k as target domain