POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master of Science in Data Science and Engineering

Master Thesis

Approaches for Domain Adaptive Object Detection in Production Environments

Supervisor
prof. Paolo Garza

Candidate
Gabriele Degola

Company advisors
Neovision
dr. Etienne Balit
ing. Valérian Gonnot

April 2022

Abstract

Nowadays, computer vision is built almost exclusively on deep learning methods. Research in recent years has improved more and more the performances of convolutional neural networks on several popular key tasks, such as image classification, detection and segmentation. However, these methods require large amounts of annotated data to be trained effectively, which are not always available.

For this reason, domain adaptation methods have been widely studied and developed. The goal is to transfer the knowledge learned by a machine learning model on annotated data to data from other domains for which annotations are not available or only partially. A typical case is the transfer from synthetic data (easy to generate in large quantities) to real data (expensive to collect and annotate).

This work focuses on domain adaptation for an object detection task, with the goal of identifying methods that can be deployed in production environments. It starts from a review of both the fundamental and some recently proposed methods and explores a simple approach to domain adaptation, without the need for complex modifications to the most commonly used network architectures. At the same time, it attempts to address some issues of more popular domain adaptation methods for object detection, that may hinder their use in actual applications.

The performed experiments show that the method can be easily applied by engineers for satisfactory results in real world projects, when it is complex to obtain image annotations.

Acknowledgements

Alla mia famiglia
Ad Alba
Ai miei nonni

Merci à Valérian, Etienne, Olivier, Taiamiti et à tout le monde chez Neovision, pour m'avoir accueilli comme si je faisais partie de l'équipe et pour m'avoir suivi minutieusement et patiemment tout au long de mon stage.

I would like to sincerely thank the supervisor of this work, professor Paolo Garza, for his thorough revision and for the useful comments and advice.

Thanks to the great people I have come across over these years. It has been a fun journey.

Once you free your mind about a concept of
Harmony and of music being "correct",
You can do whatever you want.
So, nobody told me what to do
And there was no preconception of what to do.

— Giorgio Moroder

Contents

Introduction

Over the past few decades, machine learning and deep learning techniques have achieved impressive results in a variety of tasks, of which computer vision and language processing are probably the best known. However, they generally assume that training and inference data are drawn from the same distribution, which is not always the case. Furthermore, data collection and annotation is an expensive and time-consuming task. It may therefore be desirable to apply a model trained on annotated data to another domain for which annotations are not available or only partially available. Several methods have been proposed to reduce the domain shift between the two data distributions. They have been widely studied and are presented in the following sections.

The starting point of this Master Thesis is a six-months internship carried out at Neovision[1], an engineering company based in Grenoble, France, and specialized in artificial intelligence. Transferring knowledge between different domains in an object detection scenario is a captivating but challenging task with applications in a variety of domains of great commercial interest. In addition, the shortage of available annotated data for the use case of interest is often one of the biggest obstacles for machine learning projects: engineers have to spend several hours gathering and possibly annotating related data, rather than focusing on algorithms and optimization. On the other hand, synthetic data can be easily generated in large quantities.

In addition, the tasks at hand can frequently be similar, but with data collected under different conditions of brightness, light spectrum, camera angle, and more. Previously trained models cannot be reused, and long training processes are therefore required. This is why Neovision is interested in tackling the domain adaptation problem, in order to offer the acquired knowledge and methods to its numerous customers. For this reason, the research documented in this Master Thesis proposes and shows that simple approaches for domain adaptive object detection, independent of specific employed network architectures and without introducing complex architecture modifications, can obtain satisfactory results and compete with more

[1] `https://neovision.fr/en/`

challenging methods previously proposed in the literature.

Chapter 1

Problem definition

1.1 Neural networks and computer vision

Thanks to their achievements in computer vision, deep learning methods based on convolutional neural networks (CNNs) have aroused a lot of interest over the last decade and have been used in many other branches of artificial intelligence. CNNs are a class of artificial neural networks, designed for the processing of structured arrays of data such as images.

1.1.1 Machine learning

The goal of machine learning is to extract patterns from raw input data, making a computer program build its own knowledge without relying on any hard-coded expertise. This knowledge is built from some representative data for the task of interest (the *training set*) in order to make predictions or take decisions on new data that may only be available over time (the *test set*). Machine learning approaches are often organized in three broad categories:

- **Supervised learning**, in which training data are provided as input-output pairs and the goal is to learn a rule that maps the inputs to the outputs. Common supervised learning problems are classification, when the response variable belongs to a discrete set, and regression, in which the objective is to estimate a continuous mapping function.
- **Unsupervised learning**, in which patterns and similarities in the data are sought without the need of data labels.
- **Reinforcement learning**, in which the computer program (called *agent*) interacts with an unknown environment and records the rewards associated to the actions it takes, progressively discovering the environment. The objective is to learn sequences of actions that lead to reward maximization.

Popular supervised machine learning algorithms, such as Support Vector Machines (SVM) and K-Nearest Neighbors (KNN), have strong theoretical properties and have matched human performances on numerous tasks.

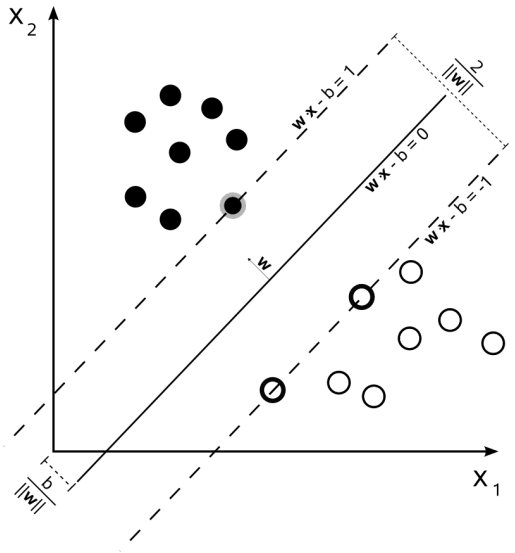


Figure 1.1: SVM for binary classification

1.1.2 Deep learning

Performances achieved by traditional machine learning algorithms heavily depend on the representation of the data with which they are fed. Indeed, they cannot analyze the full world, but only some pieces of relevant information that are available and used for learning. Each piece of information in the training examples is called a *feature*. Traditional machine learning algorithms learn from these features, but cannot influence how they are defined and extracted from the real world data.

Plenty of tasks can be solved by designing an appropriate set of features to extract, that will be forwarded to a machine learning algorithm. This requires previous knowledge, which is not always available, and intensive work and research from human actors. In many cases, it is extremely difficult to know a priori which features should be extracted. A clear example is dealing with images: humans can immediately distinguish a dog from a cat, but the difference in, e.g., fur and shape is not easy to describe in mathematical terms. A dog or a cat can take several poses and be pictured in various weather and lightning conditions. In addition, dogs or cat of different breeds can have very different appearance (high *intra-class variation*) and sometimes have common traits with other animals (low *inter-class variation*), but humans have no difficulty in classifying them as dogs or cats nevertheless. This makes the task complicated for machines.

Figure 1.2: A popular example of *intra-class variation*: even though they look different, humans do not have any trouble in saying all the images contain chairs.

To solve the problem, it is possible to use machine learning to learn an effective data representation in addition to the mapping from inputs to outputs, an approach known as **representation learning**. This increases performances with respect to hand-crafted features and enables the algorithm to easily adapt to new tasks. However, not all extracted features may be useful to understand which animal is present in an image, while some of them can be sound only if considered together with other features.

Deep learning has been a fundamental milestone in representation learning. In deep learning, representations are expressed in terms of other simpler representations that were previously extracted from the input data. In the case of dog images, a *deep neural network* can represent the concept of dog in terms of simpler concepts, such as corners and contours, which are in turn defined in terms of edges.

Different network architectures have been proposed for different tasks. The basic model is the *feedforward neural network* or *multi-layer perceptron* (MLP), whose schema is reported in Figure 1.3. Each node of the directed acyclic graph represents a neuron (from which the name *neural network*): the first network layer is the input layer, with as many neurons as input features, while the last layer is the output layer, with one node for any class the network is designed to predict (or a single node in the case of binary classification). The intermediate layers are the hidden layers, in which every neuron computes the value of a linear function of the features returned by the previous layer, followed by a scalar non-linearity. Each output serves as input for all nodes in the next layer.

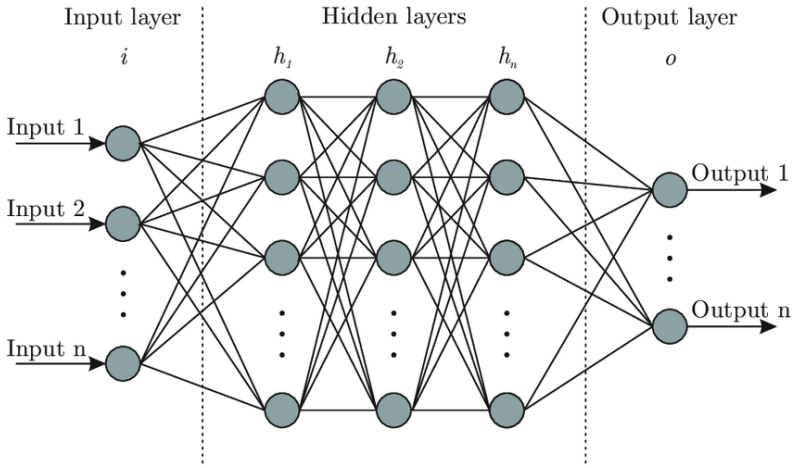


Figure 1.3: Example of feedforward neural network

More formally, the output of the j^{th} node of the first hidden layer h_1, when receiving an observation x from the n input nodes, is the composition of:

- a dot product a_j between x and the node's weight vector w_j, considering its bias term b_j:

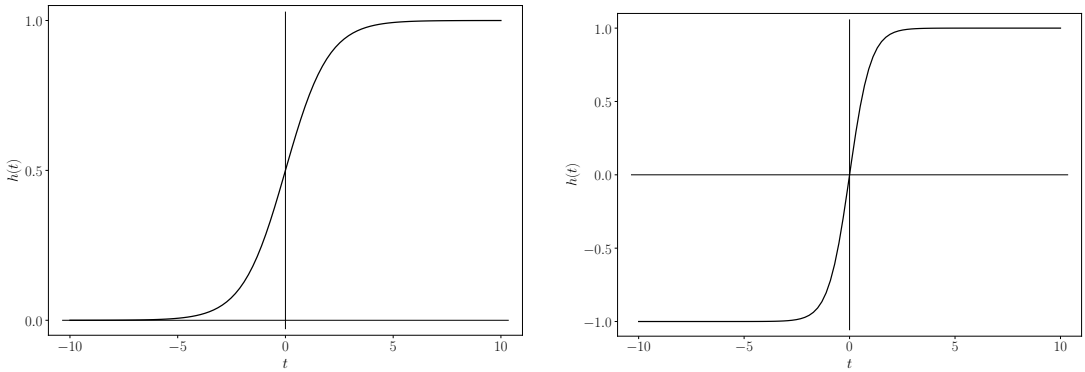
$$a_j = b_j + \sum_{i=0}^{n} w_{j,i} x_i$$

- and a differentiable activation function h, such that:

$$z_j = h(a_j).$$

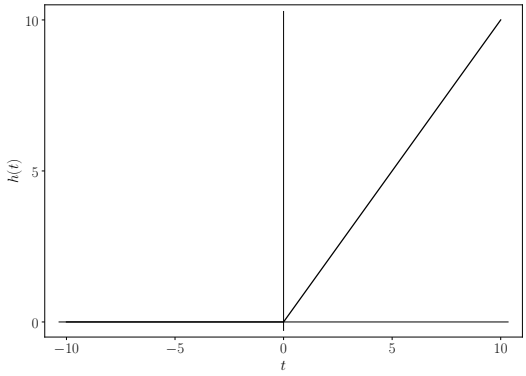
This is incrementally repeated for all neurons of every network layer, until an output is produced by the neurons of the output layer. The network learns the weights w associated to all neurons as well as their biases b, which define the output of the nodes when no input is provided[2].

Several activation functions have been proposed over the years to build deep neural networks and effectively exploit their depth. *Sigmoid* and *tan-h*, respectively defined as $h(t) = 1/1+e^{-t}$ and $h(t) = \tanh(t) = e^{t}-e^{-t}/e^{t}+e^{-t}$, have often been used as activation functions, as they squeeze input values to finite intervals ([0,1] and $[-1,1]$ respectively, as represented in Figures 1.4a and 1.4b) and are biologically inspired. In modern deep learning, the default recommendation is to use the rectified linear unit (ReLU) function, defined as $h(t) = \max(0, t)$. With respect to sigmoid and tan-h, the ReLU does not saturate (mitigating the risk of *vanishing gradient*) and has faster convergence. An alternative is the *leaky ReLU*, defined as $h(t) = \max(\alpha t, t),\ \alpha \in [0,1]$, which in turn mitigates the risk of "dying" neurons, due to the fact that ReLU sets all negative values to 0.

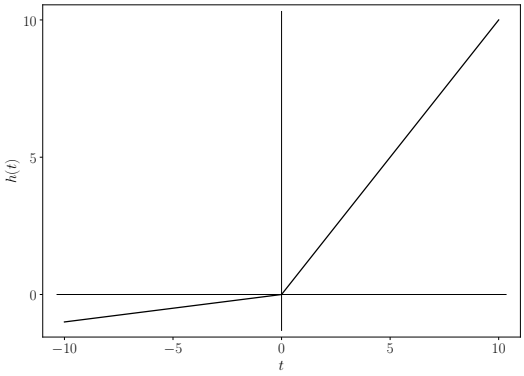


(a) Sigmoid

(b) tan-h



(c) ReLU



(d) Leaky ReLU ($\alpha = 0.1$)

Figure 1.4: Commonly used activation functions for neural networks

Network weights are learned through gradient based methods. In order to do so, a fundamental step is the definition of a *loss* or *cost* function. Loss functions are mathematical functions that map an event or the values of one or more variables to a real number, which intuitively represents a certain cost associated with these values. Therefore, each training of deep neural networks aims at minimizing or maximizing one or more loss functions that depend on the problem, in order to make network weights evolve in the desired directions.

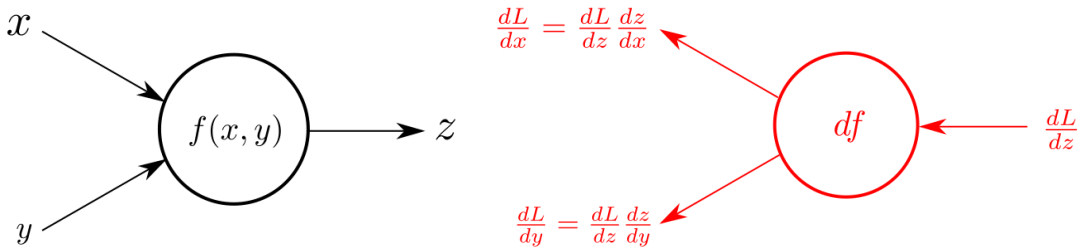
In a multi-class classification scenario, the output of each output neuron is mapped to the posterior probability of the corresponding class. This is usually done through the *softmax* operation. If the k^{th} node of the output layer returns the value y_k, it is computed as:

$$p(c_k|x) = \frac{\exp(y_k)}{\sum_{i=0}^{n} \exp(y_i)}$$

which is therefore the probability that observation x belongs to class k. This is computed for all classes and x is usually assigned to the class with the highest

probability.

According to the *backpropagation* algorithm, classification error or loss is estimated at the output layer, considering incorrectly classified examples, and network weights are updated accordingly, from the output to the input layer. This is done by computing the gradient of the loss function and distributing it backwards on the network neurons, by recursively applying the chain rule of calculus[14]. A general schema is reported in Figure 1.5.



(a) Forward step (b) Backward step

Figure 1.5: Propagation and backpropagation schema

1.1.3 Convolutional neural networks

Convolutional neural networks (CNNs) are a special class of feedworward neural networks, designed to work with inputs that are organized as grids. Hidden units are organized into grids as well. Indeed, if image pixels were directly fed to a feedforward neural network, the model could not extract meaningful relations between adjacent pixels, possibly taking into account the different color channels.

CNNs are usually composed of two main sections: the first one deals with feature extraction, while the second addresses classification or regression, depending on the desired task. A simple CNN architecture is reported in Figure 1.6.

For feature extraction, following the same principle of MLPs, increasingly complex non-linear relationships between adjacent pixels are extracted from input images, by stacking convolutional layers with non-linear activation functions. Linear mappings from one layer to the next one take the form of convolution operations with three-dimensional convolutional filters of fixed size. *Pooling* layers can then be added in order to reduce the size of the extracted feature maps. The goal of pooling is to replace the output of a convolutional layer with a summary of its local statistics, making it more robust to small differences in the input. Popular pooling operators are *max* and *average* pooling.

Different architectures can be used for this part of the network, which is typically referred to as *backbone*.

In the most simple case, extracted features are then forwarded to one or more *fully connected* layers, that can be seen as a standard MLP introduced in the previous section. The MLP assembles all local information into a global vectorial representation and outputs a prediction for the input image.

However, this part of the network depends on the task of interest and several architectures have been proposed over the years. These architectures can be complex and composed of different units for different sub-tasks. They always rely on the features extracted by the backbone, that are potentially shared among all the components of the architecture, as several high level tasks may need to access the same low level features

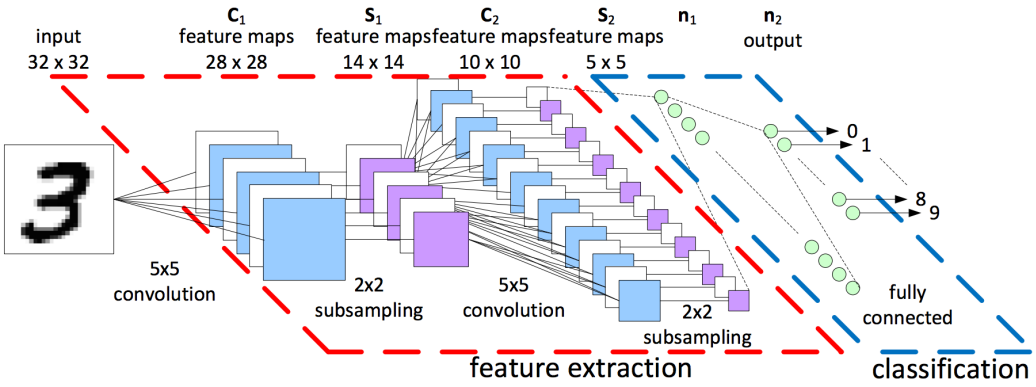


Figure 1.6: An example of CNN for handwritten digit recognition

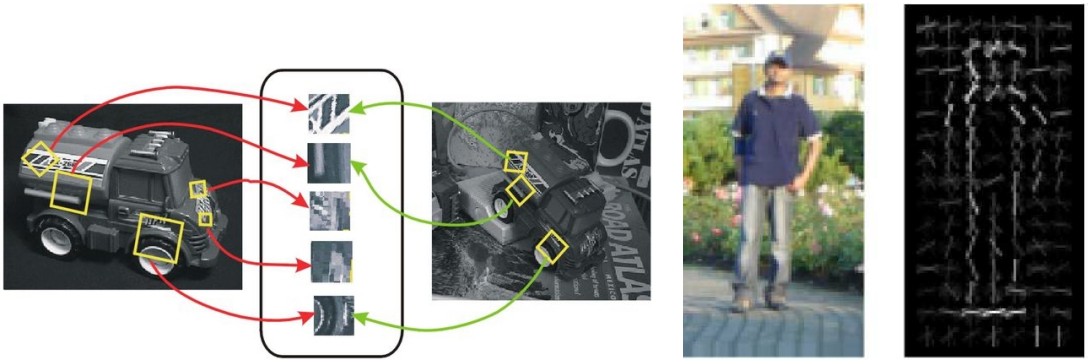
1.1.4 Computer vision

Before the rise of CNNs, computer vision researchers focused on studying effective algorithms to extract information related to the shapes appearing in the images. This is inspired by how animal and human brain works, first focusing on edges or lines, and only after on the full image.

Such algorithms include, but are not limited to, scale-invariant feature transform (SIFT) and histogram of oriented gradients (HOG). SIFT extracts local features from images, with the goal of describing different classes of objects by invariant features[28]. HOGs, instead, describe images by the number of occurrences of gradient orientations in different portions of the images[8]. Examples of the algorithms are reported in Figure 1.7.

As mentioned in previous sections, features extracted from available images are then used to train supervised or unsupervised machine learning algorithms, depending on the task of interest.

In recent years, CNN-based methods have outperformed previous approaches based on manually extracted features and classical machine learning algorithms,



(a) Scale-invariant feature transform[28]

(b) Histogram of oriented gradients[8]

Figure 1.7: Examples of hand-crafted feature extraction methods

since they are able to learn complex relationships in the data by progressively extracting higher-level features from the raw data. Figure 1.8 shows an example of the progresses made by CNNs in computer vision.

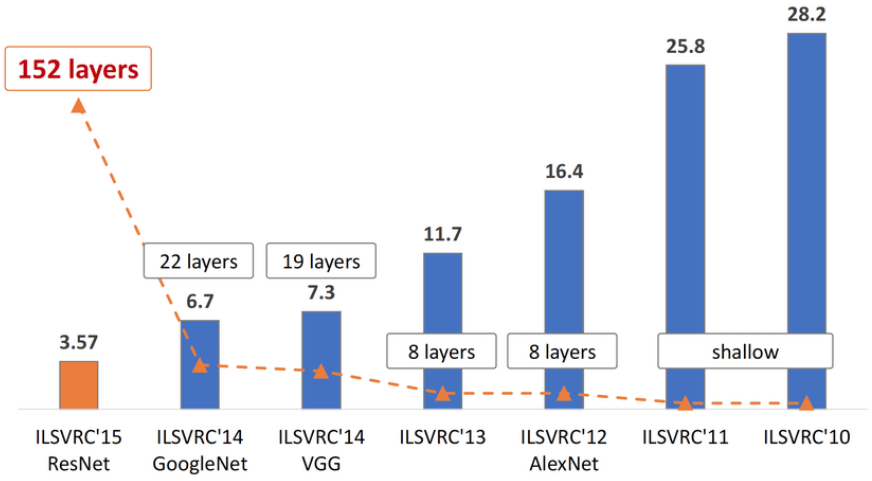


Figure 1.8: Evolution over the years of the top-5 error for the winning entries in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), from right to left[39][19].

Three main tasks can be identified in computer vision, all of which have benefited from the development of deep convolutional neural network architectures:

- **Image classification**, whose objective is to predict the class label for an image.

- **Object detection**, whose objective is to predict the position and the class label for all the objects that appear in an image, which may belong to different classes. The position of an object is represented by means of rectangular coordinates, called bounding box.
- **Semantic segmentation**, whose objective is to predict a class label for each pixel in an image. A possible extension is *instance segmentation*, which identifies the object instance for each pixel, distinguishing different objects belonging to the same class.

A summary is presented in Figure 1.9.

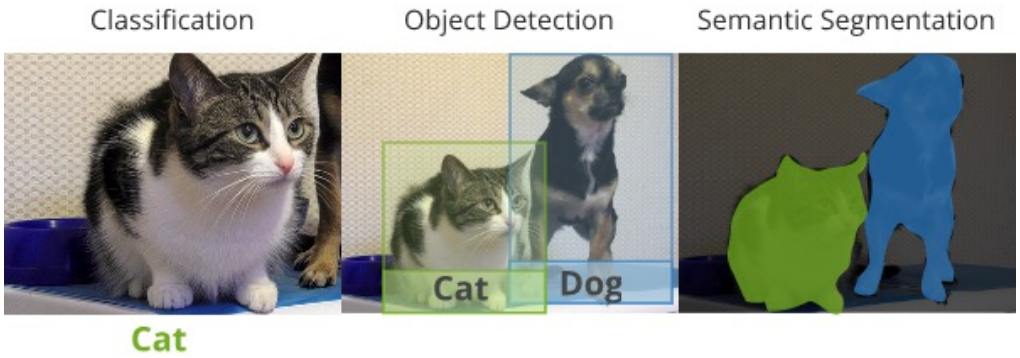


Figure 1.9: Comparison between classification, object detection and semantic segmentation

1.2 Object detection

Object detection methods are able to recognize the objects represented in an image and to highlight their position, returning the associated bounding boxes, rectangular boxes which contain an object. The bounding box format depends on the model and on the dataset. Typically, they are determined by the x and y coordinates of the upper-left corner or of the center and by the width and height of the bounding box.

Object detection is a fundamental task in modern computer vision, with applications such as autonomous driving, robot vision and human-computer interaction.

1.2.1 Existing methods

An object detection task can be seen as a combination of two main steps: finding image regions that may contain objects, and then independently classifying the

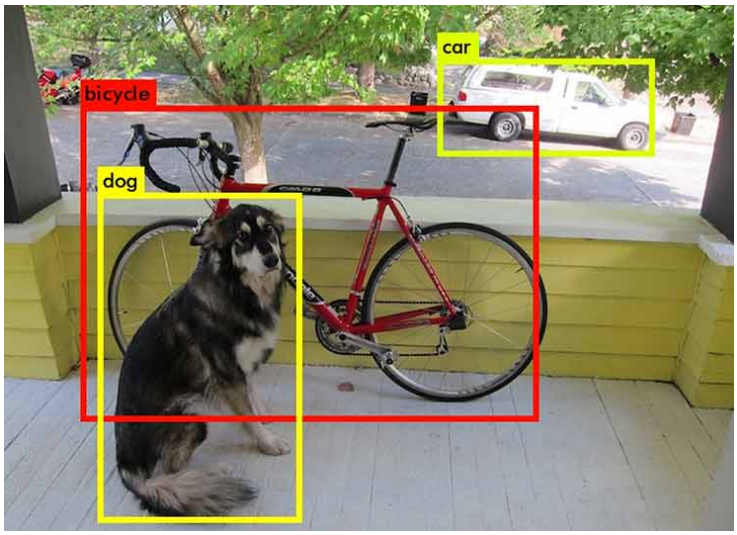


Figure 1.10: An example of object detection[35]

objects in those regions. Before deep learning, this was achieved using a sliding-window approach, where an image classifier was applied to different areas of the image and only the predictions with the highest probability were retained. Nowadays, two main families of object detectors can be identified.

Several network architectures have been using as backbones of object detectors, to extract features from input images to be fed to the high-level network components for the object detection task. Due to high performances, one of the most popular choices is exploiting a residual neural network (ResNet) architecture[16].

ResNets are neural networks characterized by the use of *shortcuts* or *skip connections*. Through these skip connections, the output of a network block is added to the same signal it received as input. This favors the transmission of the gradient from high level to low level layers during the backpropagation phase, avoiding *vanishing gradient*. The contribution of the information extracted by the first network layers during inference is favored as well. This has allowed the implementation of very deep networks, solving the problems encountered by previous architectures concerning training and accuracy. The structure of a residual block is reported in Figure 1.11.

The number of layers of a ResNet is usually specified as a suffix. Common values are 18, 34, 50 (ResNet-50), 152, up to over 1000 layers[17].

Two stage object detectors

As said, the most commonly used models for object detection are based on a two-stage approach and belong to the family of region-based convolutional neural networks (R-CNNs). In basic R-CNN[13], a manageable number of possible Regions

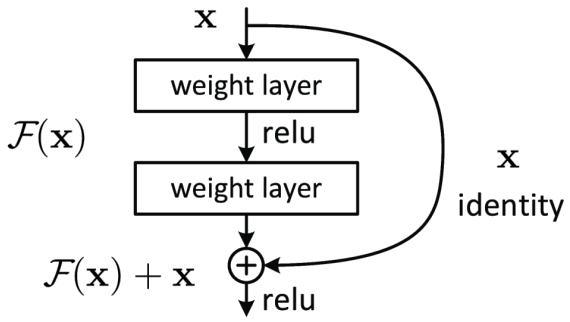


Figure 1.11: An example of building block for residual learning[16]

of Interest (RoIs) are extracted from an image, following the *Selective Search* algorithm.

Selective search aims at localizing objects in the input images, and is pitted against previously used *Exhaustive Search*, in which the sliding-window approach is followed to detect RoIs. In selective search, an input image is initially segmented in small regions, which are then recursively merged according to their similarity. The operation is repeated until bigger independent regions are generated. These regions are returned as candidate RoIs.

A CNN is then evaluated independently on each proposed RoI to extract features that are fed into a SVM to determine if an object is present in this bounding box (classification task) as well as its location to match that of the object (regression task). As SVM is a binary classification algorithm, as many SVMs as classes of interest are used for the classification task. A summary of the method is reported in Figure 1.12.

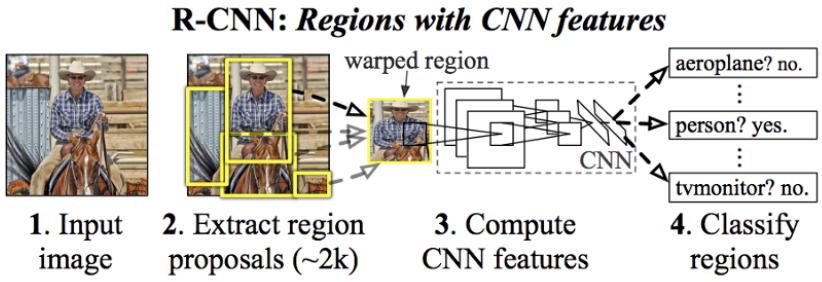


Figure 1.12: Description of standard R-CNN[13]

Fast R-CNN[12] and Faster R-CNN[36] enhanced the R-CNN architecture, improving its performances and reducing the inference time.

Fast R-CNN In Fast R-CNN[12], performances are improved through the introduction of a *RoI pooling* layer and of a single-stage training procedure with an optimized multi-task loss.

RoI pooling layer allowed to solve one of the main sources of slowness of the previous R-CNN architecture: as described, in R-CNN each proposed RoI is independently fed to a CNN for feature extraction.

With RoI pooling, the whole input image is forwarded only once to the feature extractor. Rectangular feature maps, corresponding to each proposed RoI, are then extracted from the image's features.

Following a single-stage training, instead, Fast R-CNN discards the previously used SVMs for classification. Two sibling output layers are used:

- the classification head returns, through softmax and for each RoI, a discrete probability distribution p over $K+1$ categories (the K classes of interest and an "extra" background class);
- the regression head returns, for each RoI and class k, a bounding box offset t^k, which specifies a scale invariant translation and height/width shift in logarithmic space.

The two heads are trained jointly, thanks to a purpose-built loss function that takes into consideration both classification and regression tasks. For each couple of classes u, v, with u ground truth class, this multi-task loss is defined as:

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda \, [u \geq 1] \, L_{\text{loc}}(t^u, v). \tag{1.1}$$

Here, $L_{\text{cls}}(p, u) = -\log p_u$ is the standard negative log-likelihood for the correct class u and is responsible for the training the classification head. $L_{\text{loc}}(t^u, v)$ is instead responsible for the regression head and is defined as a robust L_1 loss:

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

in which

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

$[u \geq 1]$ is an indicator function that evaluates to 1 when $u \geq 1$ and 0 otherwise, meaning that the regression head is not trained for RoIs belonging to the background class (labeled $u = 0$ by convention). Indeed, the concept of ground truth bounding boxes does not make sense for background regions.

Finally, the λ hyperparameter is used to assign a relative weight to the two tasks.

Faster R-CNN The main contribution of Faster R-CNN[36] over Fast R-CNN is the introduction of a Region Proposal Network (RPN), which integrates some convolutional layers of the image classifier into the region proposal phase. It is designed to replace selective search, an expensive and time-consuming algorithm, and helps to considerably reduce the required time. Faster R-CNN working scheme is reported in Figure 1.13.

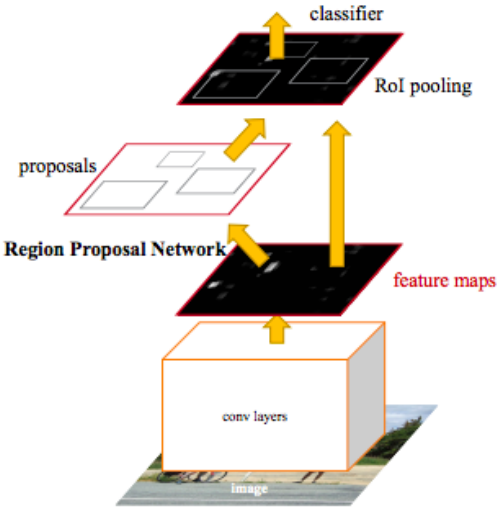


Figure 1.13: Description of Faster R-CNN[36]

The goal of RPN is to extract a set of RoIs from an input image, each with a probability that it contains indeed an object instead of a background region. Basically, this CNN shares a common set of convolutional layers with a Fast R-CNN object detector and highlights the image regions on which it should focus.

In order to generate RoI proposals, the RPN receives as input the features extracted by the last shared convolutional layer, and a sliding-window approach is followed to extract multiple proposals. Each RoI is mapped to a lower-dimensional feature map, which is fed to two sibling fully-connected layers: a binary classification layer which estimates the probability that the input RoI actually contains an object, and a box-regression layer to refine the bounding box proposals.

The regions that are generated at each sliding-window location have different sizes and aspect rations and are called *anchor boxes*. This is an important concept in modern object detection, and is further detailed in Section 1.2.2.

RPN can then be trained with the goal of minimizing a multi-task loss function that follows the same reasoning of (1.1).

However, the RPN and the Fast R-CNN detector share a set of convolutional layers: independent training of the two components would modify the weights in different ways and results would be inconsistent. Three techniques are proposed

for jointly training networks with shared features:

- *Alternating training*: the RPN is initialized as a pre-trained CNN and fine-tuned end-to-end on the RoI proposal task, and the generated proposals are used to train a Fast R-CNN object detection network. In this phase, no convolutional layer is shared among the two networks. Fast R-CNN is then used to initialize RPN training, but fixing shared layers and fine-tuning only the ones unique to RPN, making the two components share convolutional layers. Finally, shared layers are kept fixed and unique layers of Fast R-CNN are fine-tuned. The process can be repeated for many iterations.

- *Approximate joint training*: RPN and Fast R-CNN components are merged into one network. However, this joint training leads to approximate results, as RoI proposals are treated like fixed, pre-computed proposals when training the Fast R-CNN. For this reason, during backpropagation the fact that the proposed bounding boxes' coordinates depend on the RPN is ignored.

- *Non-approximate joint training*: the above-mentioned problem could be resolved by computing gradients with respect to the box coordinates in the backpropagation step. This would require the development of a RoI pooling layer that is differentiable with respect to box coordinates. This is a nontrivial problem.

Because of its good performances and speed, Faster R-CNN is widely used for benchmarking and as base for several derived works.

One stage object detectors

Differently from two stage detectors, one stage object detectors skip the region proposal phase and focus on predicting object regions and classes together. This leads to faster predictions with respect to two stage models, making them suitable for real-time applications at the cost of lower prediction quality. The pioneer work for one stage object detection is OverFeat[42], published in 2014. Here, the last classification layers of a standard CNN are replaced by a regression network for each class, in order to predict the coordinates of the object bounding boxes.

YOLO The You Only Look Once (YOLO) model[35] uses pretrained CNN for classification and splits each image in cells. If the center of an object falls into a cell, that cell is responsible for detection and should predict the bounding box locations, a confidence score and a class probability for the detected objects. A specific loss function is used to efficiently learn to predict bounding boxes and object classes at the same time.

SSD Single Shot Detector (SSD)[27] uses a VGG-16 convolutional network as backbone and adds on top several convolutional layers of decreasing sizes, in order to build a *pyramid representation* of the images and efficiently detect object of different sizes, at each pyramid layer. Instead of splitting images in cells, anchor boxes are used for faster detection.

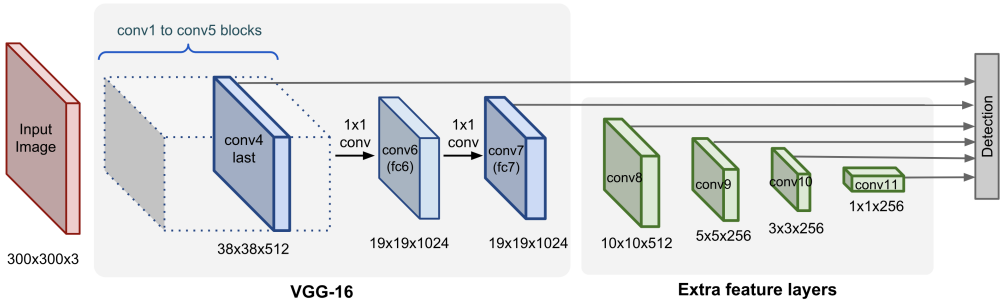


Figure 1.14: SSD's pyramid architecture[47]

After their proposal, YOLO and SSD have both been extensively studied and improved by the research community. The last versions of the two families are architecturally close and work in very similar ways.

RetinaNet The main improvements of RetinaNet[25] over previous one stage detection models are the use of *Feature Pyramid Network* (FPN) as part of the backbone and of *focal loss*, which helps to approach the results of two-stage detectors.

Feature Pyramid Network FPNs[24] adopt and improve the same approach as SSD's pyramid layers. They are built of a sequence of pyramid levels which correspond to network stages. Each stage is composed by multiple convolutional layers of the same size, divided by two at consecutive stages. In addition to standard feedforward, different stages are linked by a top-down pathway in the opposite

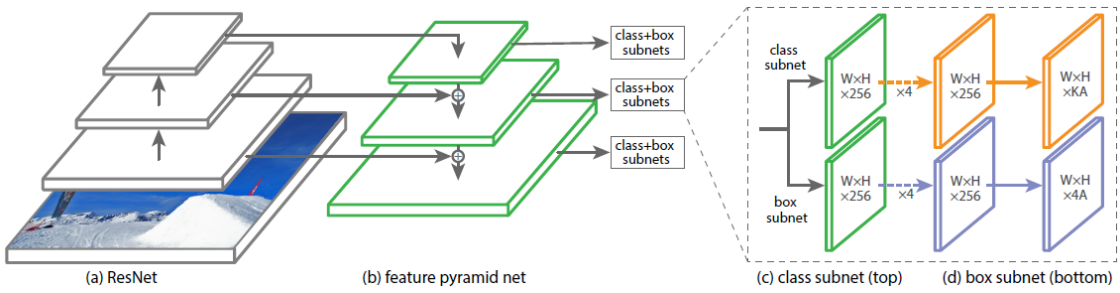


Figure 1.15: RetinaNet architecture[25]

direction, together with lateral connections. This allows the network to construct a rich and multiscale feature pyramid for each input image.

After their proposal in the context of single-stage detectors, FPNs have also been used as part of the backbone of two-stage detectors, aiming at improving their performances.

Focal loss One of the main obstacles for the training of other object detection models is the large imbalance between background and foreground examples, which drives the model to focus on irrelevant background regions. As explained in previous sections, each training of deep neural networks aims at minimizing or maximizing one or more loss functions that depend on the problem. In this case, a specific loss function can be used to encourage the network to focus on regions of the images that actually contain objects.

The *focal loss*, introduced by RetinaNet, is based on the *cross entropy* loss. For binary classification, the cross entropy loss is widely used and defined as:

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise} \end{cases} \tag{1.2}$$

where $y \in \{\pm 1\}$ is the ground truth class and $p \in [0,1]$ is the model's probability for class 1. In this context, $y = 1$ and $y = -1$ represent the foreground and background classes respectively. Defining p_t as:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases} \tag{1.3}$$

cross entropy can be written as $\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$.

A balanced version of the cross entropy loss is sometimes used to deal with class imbalance. A numeric factor $\alpha \in [0,1]$ can be introduced for class 1, with value $1 - \alpha$ for class -1. α_t is defined analogously to p_t. Therefore, the cross entropy loss, balanced by α_t, is written as:

$$\mathrm{CE}(p, y) = -\alpha_t \log(p_t). \tag{1.4}$$

With cross entropy, also examples that are correctly classified with high confidence (p close to 1 if $y = 1$, to 0 otherwise) produce non-negligible values for the loss.

In focal loss, an additional modulating factor $(1 - p_t)^\gamma$ is introduced, with tunable focusing parameter γ. Focal loss is therefore defined as:

$$\mathrm{FL}(p_t) = -(1 - p_t)^\gamma \log(p_t) \tag{1.5}$$

The contribution of easy predictions to the overall loss is more or less important depending on the value of γ. With a high γ, well detected objects provide a very little contribution to the total loss.

Considering the strong imbalance between the foreground ($y = 1$) and background ($y = -1$) classes, the focal loss balanced by the α_t parameter as previously defined is used in RetinaNet:

$$\mathrm{FL}(p_t) = -\alpha_t \, (1 - p_t)^{\gamma} \log(p_t). \tag{1.6}$$

1.2.2 Basic components

This section introduces some basic aspects of object detection models. They are necessary to understand the following sections and the performed work.

Object detection metrics

Algorithm evaluation is a fundamental part of machine learning projects and can be performed in different ways, depending on the task and objective. Object detection is a more complex task than traditional image classification: a non-fixed number of objects can be present in the same image and the detection model has to accurately predict their class, together with their position. For this reason, dedicated metrics must be introduced and defined.

Precision and recall First, the following concepts must be defined in the context of object detection:

- **True positive** (TP): a bounding box is correctly assigned to an object that actually appears in the image;
- **False positive** (FP): incorrect detection, an object is detected but is not really present in the bounding box or only partly;
- **False negative** (FN): an object present is the image is not detected by the object detection model.

The concept of true negative (TN) does not make sense in an object detection context, as the missing detection of a non-present object is not relevant. In fact, this would mean that the model has correctly identified the background as not belonging to any class.

Precision Precision measures how accurate the predictions of the model are, so it returns the percentage or fraction of correct predictions.

$$\text{precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{1.7}$$

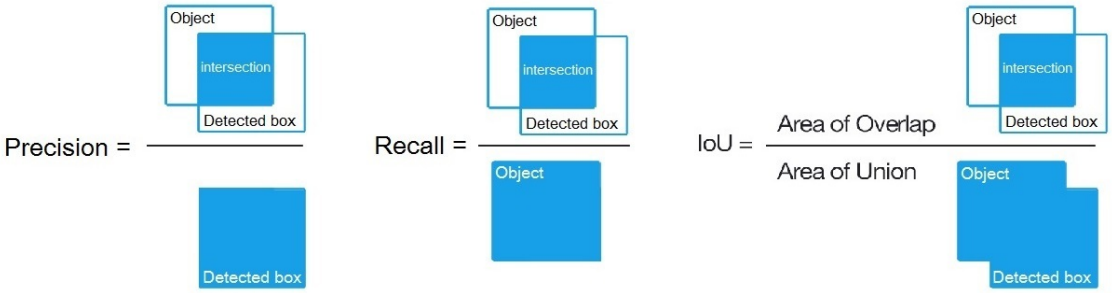


Figure 1.16: Graphical representations of precision, recall and IoU

Recall Recall measures the model detection ability, so it returns the percentage or fraction of correctly detected objects.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1.8}$$

Intersection over union For each predicted bounding box, the overlapping with the ground truth bounding boxes is computed, in order to classify a prediction as true or false positive. For a couple of bounding boxes, Intersection over Union (IoU) is computed as:

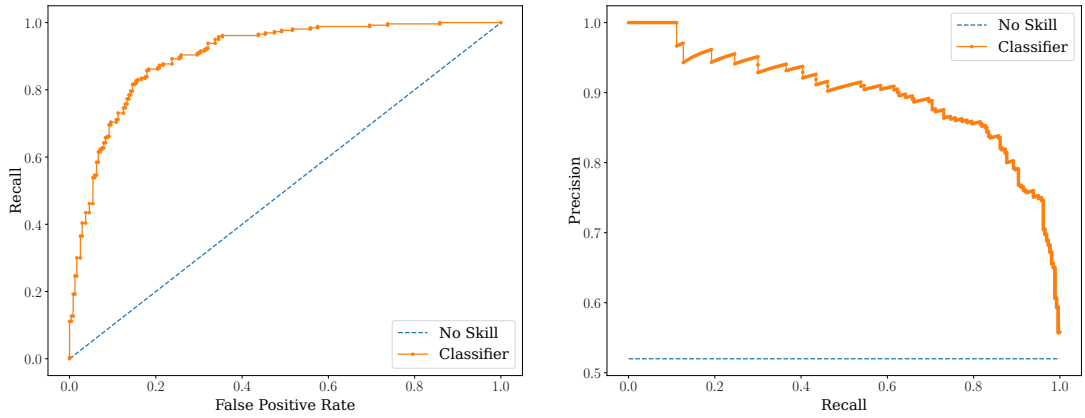
$$\text{IoU} = \frac{\text{area of overlap}}{\text{area of union}} \tag{1.9}$$

The resulting value is then compared to a threshold to classify a prediction as correct or not. Typical values for the threshold are 0.5, 0.75 and 0.95, and that defines how much we want a detection to be precise. Indeed, a prediction classified as true positive with a certain threshold may be considered as false positive if a higher value is used.

Mean Average Precision (mAP) Receiver Operating Characteristic (ROC) curve is a popular concept in machine learning. There, recall (the true positive rate) and false positive rate are plotted at different classification thresholds. The associated Area Under the Curve (AUC) metric considers the area under the ROC curve to assess the performances of a model[6]. It ranges from 0 to 1, with 0 meaning the model's predictions are all wrong and 1 they are all correct. AUC score is 0.5 for a model that always returns random predictions.

Similarly, the precision-recall curve shows the trade-off between precision and recall for different thresholds[34]. Average Precision (AP) is defined as the area under the precision-recall curve. A high area under the curve means both high recall and high precision. The definition of mean Average Precision (mAP) depends on the context. If AP is independently computed for each class or for different IoU

thresholds, then mAP is the average of AP. In other cases, mAP and AP can be used interchangeably.



(a) ROC curve

(b) Precision-recall curve

Figure 1.17: Comparison between ROC curve and precision-recall curve for a trained classifier

Datasets

Several datasets have been published in the last years, with the purpose of training and evaluating the performances of object detection algorithms for different use cases.

COCO Common Objects in Context (COCO) is a large-scale object detection and object segmentation dataset published by Microsoft, containing over 200000 images for 91 classes, annotated with bounding boxes and at pixel level[26]. It defines its own annotation format as JSON files.

PASCAL VOC PASCAL Visual Object Classes (VOC) is a classification and object detection dataset, developed for different machine learning competitions between 2005 and 2012[43]. Annotations are stored in XML format, with a file for each image in the dataset. Bounding boxes are determined by the x and y coordinates of the upper left and bottom right corners.

CityScapes CityScapes is an object segmentation dataset containing 5000 street scenes images from 50 different cities, with the aim of training models for autonomous driving[7]. When used for object detection, bounding boxes can be derived from the tightest rectangles containing the segmentation masks.

Several variations of the dataset have been proposed with artificially added adverse weather, such as fog[41] and rain[15], in order to evaluate the robustness of detection methods in various weather conditions.

KITTI The Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) dataset[11] contains 7481 images captured in a context similar to CityScapes. Bounding box annotations are stored in plain text, with one file per image.

Sim10k Sim10k is a synthetic dataset containing 10000 annotated images for car detection, rendered from the popular open-world video-game *Grand Theft Auto 5*, with the objective of training models in a rich virtual world for detection in the real[21].

Anchor boxes

In most cases, generating many possible bounding boxes is too cumbersome and leads to inaccurate predictions. If we already know which classes we want to detect, it may be more efficient to define a-priori a set of possible aspect ratios and zoom factors for the bounding boxes. For example, if we are interested in detecting cars, wide *anchor boxes* can be defined at different scales, to identify both near and far objects of car shape.

As mentioned in Section 1.2.1, anchor boxes were first introduced for the RPN of Faster R-CNN and were one of the main source of improvements in detection performances and speed[36]. The basic working schema is reported in Figure 1.19. Nowadays, anchor boxes are used in most architectures for object detection, such as YOLO, SSD and RetinaNet. From an anchor boxes point of view, the difference between one-step and two-step methods is that the former analyze each anchor box independently, then refine and return only the most promising ones, while the latter start from the anchor boxes to estimate the true bounding boxes of the objects in the images.

Non-maximum suppression

Object detection models usually produce more than one detection for each object, which is not desirable. *Non-maximum suppression* (NMS) is a method to select one of many overlapping bounding boxes. For a set of overlapping boxes, the one predicted with the highest confidence is selected and all remaining boxes whose IoU with the chosen bounding box is over a certain threshold are discarded. In this way, only the most probable detection is kept for each object. An example is reported in Figure 1.20.

(a) CityScapes[7]

(b) KITTI[11]

(c) Sim10k[21]

Figure 1.18: Examples of different street scenes datasets. It is evident that the appearance of images from different domains varies, even though they contain the same categories of objects. Indeed, images can be taken from different cameras and in different lightning or weather conditions. More extremely, *Sim10k* only contains computer generated images: the texture of the objects widely differ from the ones in the other datasets.

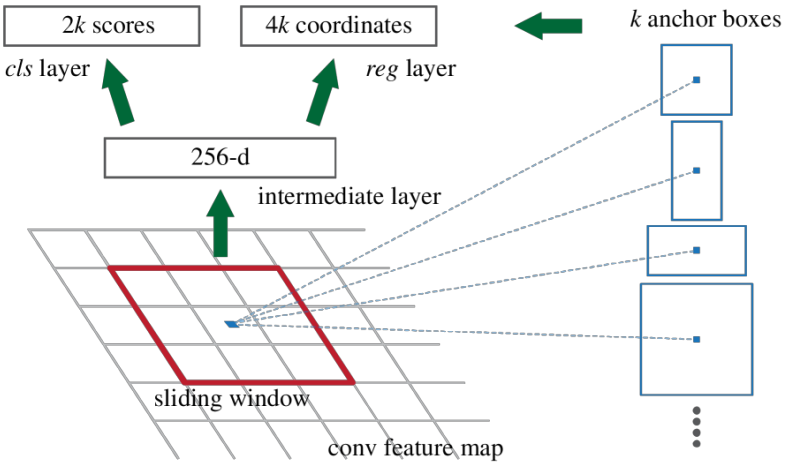


Figure 1.19: Anchor boxes, as used in Faster R-CNN's RPN[36]. At each position of the input image, k bounding boxes with different scales and aspect ratios are proposed and forwarded to the rest of the network for evaluation.

1.3 Domain adaptation

For most use cases, a huge number of annotated images are necessary to train robust and efficient deep learning models. In an industrial context, the process of collecting and annotating data suitable for each specific customer case is often too expensive and time-consuming. In contrast, synthetic data can be generated through various rendering software, even free and open-source, easily and in massive quantities, for simulating the environments in which the models will be applied. However, deep learning models are effective if the training and test data are drawn from the same data distribution, so a model trained on simulated data is likely to perform badly on real data. Domain adaptation techniques can be used to transfer the knowledge acquired on the **source domain** (e.g., synthetic data) to the **target domain** (e.g., real data) without training a new model on the latter. In a domain adaptation setting, the task to be performed on source and target domains and the classes to be predicted remain the same, while the domain differs.

Domain adaptation methods are effective for several computer vision tasks, therefore dealing with image classification, object detection and semantic segmentation. In addition to the knowledge transfer from synthetic to real images, among other cases they can be used to adapt a model to different cameras, different weather conditions and moments of the day (e.g. from sunny to rainy weather or from day to night) or from infrared to normal images.

Different types of domain adaptation can be identified, based on whether annotations are available or not for the target domain:

- **Supervised domain adaptation**: labeled data are available for both source

(a) Before non-maximum suppression

(b) After non-maximum suppression

Figure 1.20: Example of application of non-maximum suppression on a Faster R-CNN detector: several bounding boxes are proposed for each detected car, but only one per object is returned.

and target domains.

- **Semi-supervised domain adaptation**: labeled data are available for the source domain, but only some data from the target domain are actually labeled.
- **Unsupervised domain adaptation**: labeled data are available for the source domain, but no labeled data are available for the target domain. All data from the target domain is therefore unlabeled.

Moreover, domain adaptation can be classified as **homogeneous**, if the domains share the same feature space, or **heterogeneous**, if features differ.

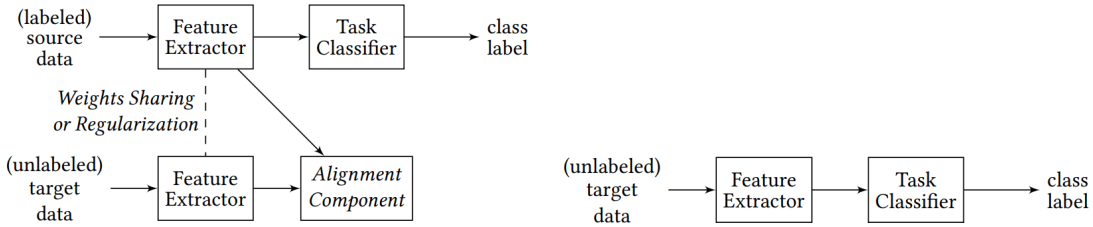
Domain adaptation is mostly studied in the unsupervised and homogeneous scenario, with one source and one target domain.

1.3.1 Domain adaptation methods

Different approaches to the problem have been proposed in recent years. The most frequent ones are *domain-invariant feature learning* and *domain mapping*.

Domain-invariant feature learning

Several methods try to align source and target domains by creating a domain-invariant feature representation at the level of the neural network for feature extraction. Intuitively, if a model performs well on domain-invariant features extracted from the source domain, it should generalize to the target domain since the two distributions are aligned. A schema is reported in Figure 1.21.



(a) Training (b) Testing

Figure 1.21: General schema for domain-invariant feature learning[48]

Methods following this approach differ in how they align the domains. An adversarial approach, as in generative adversarial networks (GANs), is often used through a domain classifier which outputs whether the features extracted by the CNN belong to source or target domain. At the same time, the feature extractor is trained with the aim of fooling the domain classifier, so that it is unable to correctly classify from which domain the feature representation originates. The two networks are trained simultaneously and in an adversarial fashion.

Domain mapping

An alternative to creating domain-invariant feature representations is mapping one domain to the other, making source images similar to the target or vice versa. For example, synthetic images from *Sim10k* can be mapped to the real world, transferring them to the *CityScapes* style. This can be obtained with conditional GANs for image-to-image translation that do not take into account the annotations of source images. Therefore, domain adaptation can be accomplished by training a GAN to map data from source to target and then training a machine learning model on the mapped labeled source images (that can now be considered as belonging to the target domain). At test time, the model is used to make predictions on unlabeled target data. A general schema is reported in figure 1.22. However, GAN

training is challenging and may incur in several problems such as mode collapse, when a generative network learns to only produce a limited amount of output images.

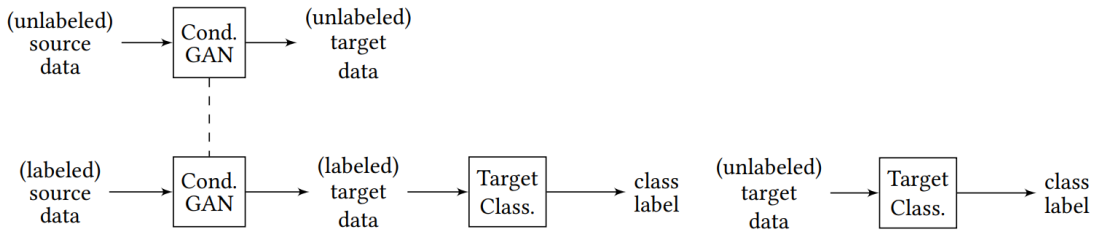


Figure 1.22: General schema for domain mapping[48]

Previous works

Several fundamental domain adaptation methods can be identified in the case of image classification. These methods have laid the foundation for many subsequent works.

Domain Adversarial Neural Network Domain Adversarial Neural Network (DANN)[10] is a classic method for unsupervised domain adaptation with the objective of learning domain invariant features. The network is composed of a standard feature extractor, a label predictor and a domain classifier: without this last component, it consists of a standard model for image classification.

The alignment between domains is achieved through the gradient reversal layer, introduced in this work: this layer leaves input unchanged during the forward step and reverses the gradient during backpropagation, multiplying it by a negative factor $-\lambda$. The network is trained adversarially, with the aim of minimizing the label predictor's loss. At the same time, it also seeks the backbone parameters that maximize the domain classifier loss and the domain classifier parameters that minimize its own loss.

The objective is to align the two data distributions, fooling the domain classifier: when alignment is accomplished, the label predictor is capable of correctly classify an image, whether it is taken from the source or the target domain, while the domain classifier can only return random guesses. A network representation is reported in Figure 1.23.

More formally and using the same notation as in Figure 1.23, the aim is to find the values of the parameters θ_f, θ_y and θ_d that minimize the following function:

$$E(\theta_f, \theta_y, \theta_d) = \sum_{\substack{i=1\ldots N \\ d_i=0}} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1\ldots N} L_d^i(\theta_f, \theta_d). \tag{1.10}$$

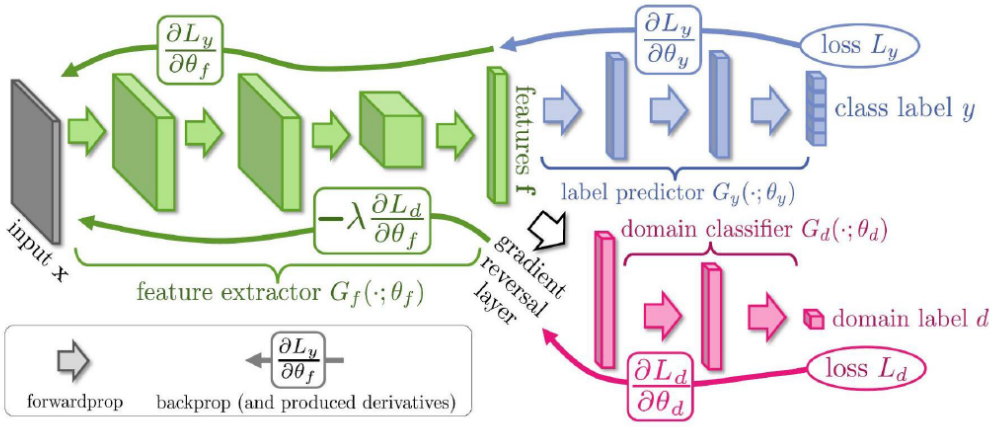


Figure 1.23: DANN architecture[10]

In (1.10), L_y and L_d are the objective functions for label and domain prediction respectively: L_y^i and L_d^i are computed for the i-th training image. By mapping the source and target domains to 0 and 1 respectively, the condition $d_i = 0$ in the first sum implies that the label classifier is trained only if the network receives annotated images from the source domain.

λ is therefore the only additional hyperparameter with respect to the training of a standard classification model and its value is to be fixed a priori. To increase the stability of the training and to foster good learning of the domain classifier at the beginning of the training procedure, λ is sometimes set dynamically. The following expression can be used to change its value from 0 (at the beginning of the procedure) to 1 (at the end) according to the training progress represented by p (in turn between 0 and 1):

$$\lambda_p = \frac{2}{1 + \exp\left(-\gamma \cdot p\right)} - 1. \tag{1.11}$$

This avoids the need to seek the optimal value for λ, but introduces the new hyperparameter γ (which represents the speed at which the value of λ_p converges to 1).

Adversarial Discriminative Domain Adaptation Adversarial Discriminative Domain Adaptation (ADDA)[46] is an adversarial method like DANN, but is composed of three consecutive phases:

1. A CNN is trained on labeled source examples in order to learn a feature representation for the source domain that can be successfully identified by a classifier. This source CNN is frozen.

2. Adversarial adaptation is then performed. The objective is to train a feature extractor for the target domain: features extracted from the target domain have to be aligned to the ones extracted by the feature extractor trained at step 1 on images belonging to the source domain. A domain discriminator that receives features extracted by the aligned feature extractor from an image belonging to the target domain should therefore state that these features are extracted from an image of the source domain instead.

   This is done through "flipped labels": the training procedure fools the domain discriminator into believing that the features extracted from the target domain belong to the source domain instead. As a consequence, the target feature extractor is lead to learn features for the target domain that the domain discriminator could misclassify (as belonging to the source domain).

3. For inference on images belonging to target domain, the aligned feature extractor (trained at step 2) is used to extract features that now belong to the shared feature space. These features are then classified by the classifier that was previously trained on the source domain (in step 1).

A method overview is reported in Figure 1.24.

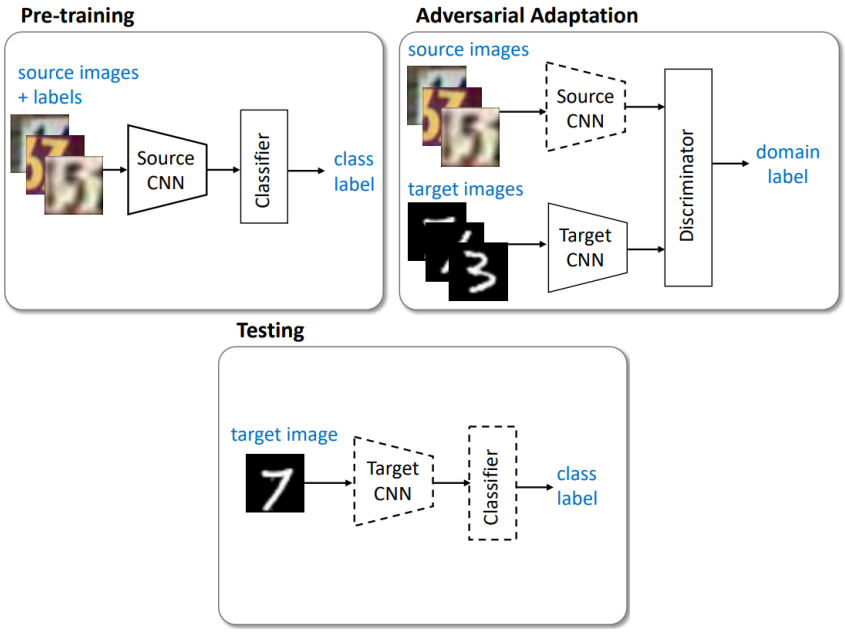


Figure 1.24: ADDA working schema[46]. The dashed lines indicate the parts of the networks that are previously learned and now frozen, while the network learns the remaining parameters.

Enhancing photorealism enhancement It is worth mentioning a domain mapping approach, recently proposed by Richter *et al.*[38], for generating photorealistic images from computer games. Synthetic images, taken for example from the video-game *Grand Theft Auto 5*, are processed in a way that they look realistic, following the camera and lightning conditions of real street scenes datasets such as *CityScapes* or *KITTI*. Resulting images are adapted to the data distribution of the dataset containing real-world images and can be used to train machine learning models for any computer vision task. The model can then be evaluated and applied on real data that the model has never seen. A few examples are reported in Figure 1.25.

Figure 1.25: Examples of application of photorealism enhancement of images from the video-game *GTA 5* (left), to mimic the style of the *CityScapes* dataset (right)[38]

In this work, photorealism enhancement is performed starting from a set of intermediate buffers, called *G-buffers*, that are produced by game engines during the rendering process. These buffers are extracted from the computer graphics pipeline and contain information about geometry, materials, and lightning in the scene. It is therefore required to have access to these low-level elements. After extraction, they are fed to an *image enhancement* CNN, with the goal of adapting image features.

The image enhancement network is trained jointly with two objectives:

- a Learned Perceptual Image Patch Similarity (LPIPS) loss, first introduced by Zhang *et al.*[50], is used to penalize large structural differences between the rendered and the enhanced images;

- a discriminator network is trained to discern between real and synthetic enhanced images and is used to evaluate the realism of adapted images.

In addition, a sampling strategy is followed to select synthetic and real image patches with similar content. The full training procedure allows reducing visual artifacts and providing temporally stable results.

1.3.2 Domain adaptation for object detection

Like classification models, object detectors suffer from domain shifts between different datasets, which can lead to drastic performance degradation. Annotating images for an object detection task is even more expensive and time-consuming than classification, since each image may contain more than one object that needs to be assigned a bounding box and object category. Due to the different nature of the problem, direct translation of domain adaptation methods for classification may not be effective, and ad-hoc methods have been proposed, generally based on existing object detectors.

Previous works

Domain Adaptive Faster R-CNN Domain Adaptive Faster R-CNN[3] is one of the first works to tackle the problem of cross-domain object detection. It uses a Faster R-CNN as base object detector and the network is trained with adversarial adaptation with gradient reversal layers at different levels. Indeed, in an object detection scenario the domain shift can occur at high level, such as image style and illumination, and at low level, such as object appearance and bounding box size. The former is referred to as *image level*, while the latter is defined as *instance level*. An additional loss function, referred as *consistency loss*, is used to enforce consistent predictions from the two domain classifiers.

At inference time, the aligned Faster R-CNN is normally used.

Strong-Weak Distribution Alignment Aligning image level features as in Domain Adaptive Faster R-CNN makes the strong assumption that image components, such as backgrounds and scene layouts, must be similar across domains in addition to object categories. In the case of large domain shifts, that may affect the model performances. Saito *et al.*[40] propose a weak global alignment (at image level) combined with a strong alignment of local features (such as texture or color of the objects in the two domains), which are most likely to be similar.

For weak alignment between domains, the domain classifier should focus on examples that are hard to classify, so near the boundaries between the two data distributions. This can be achieved using the focal loss to train the domain classifier, in order to tune the importance of source or target images whose domain is easy to predict.

Progressive Domain Adaptation Hsu *et al.*[18] introduce an intermediate domain between source and target, exploiting a GAN to map source images in the target domain. Adaptation is performed in two steps, from source to intermediate and then from intermediate to target, in order to introduce a bridge between source and target domains. That helps to reduce the domain shift and simplifies the adaptation process, such that an object detection model can be trained on the intermediate domain. In addition, a weight is assigned to generated data, giving more importance to images that are closer to the target distribution.

When adaptation between the source and the intermediate domains is completed, the aligned model can be used for inference on unlabeled images from the target domain.

Domain Adaptive RetinaNet Recently, Pasqualino *et al.*[31] proposed an adaptation approach for the RetinaNet model, but that can be applied to any architecture that includes a FPN backbone. Different domain discriminators with different architectures are applied at different levels of the ResNet backbone which is used to construct the feature pyramid, with gradient reversal layers. In this way, extracted features are aligned at different levels in the backbone and the representation derived by the FPN is coherent and meaningful.

Image statistics matching The method introduced by Abramov *et al.*[1] relies on image transformations at pixel level instead of adversarial techniques or complex architectures. The objective is to transform annotated source images in a way to make them similar to the unlabeled target domain, matching color statistics and histograms. During training, a target image is randomly chosen for each image belonging to the source domain and the mean and covariance of its color channels are aligned, as well as its color histogram that are matched accordingly. In this way, a standard detection model can be trained on modified source image with annotations and directly applied on the target domain.

Similar approaches are commonly used in networks for style transfer, but in these cases the correspondence between source and target images is performed at the level of extracted features and not at image level.

Chapter 2

Method

2.1 Introduction

This Master Thesis starts from the investigation of common approaches for domain adaptive object detection, in order to identify and possibly correct critical points and propose appropriate means of improvement.

A widespread belief is that extremely complex methods are more likely to achieve good performances than simple ones. Considering for example neural networks, the trend over the years has been to try making the networks deeper and deeper, which has produced extraordinary results but at the same time led to several problems in the training procedures. Recently, simple architectures like *MLP-Mixer*[45] and *Feed-Forward Stack*[29], composed of multi-layer perceptrons only, reached surprisingly good performances in image classification.

A similar question can be raised for the problem of domain adaptation in an object detection scenario: can simple methods perform comparably or even better than the complex ones that are mostly studied in the literature? Indeed, in an industrial context engineers do not often have the time to deeply understand elaborate network architectures, potentially not useful for the problem they are facing, and must consider the trade-off between complexity and expected efficacy. Simple methods that are capable of reaching good performances are greedily researched and can boost their productivity.

These "simple" methods are mostly explored in this work. However, traditional domain adaptation methods, that deeply affect the network architecture, play an important role in research, and present unresolved issues that may undermine their use in actual engineering applications. Possible ways to address some detected issues are explored and tested, in parallel to the aforementioned simple approaches to domain adaptive object detection.

For the sake of completeness, the studied methods are implemented and tested

on two popular object detection models presented in Section 1.2, when they are compatible with both architectures: Faster R-CNN and RetinaNet.

2.2 Data augmentation for domain randomization

Data augmentation is a widely used technique to ease the training of deep learning models and improve their generalization capability. In a typical application, different transformations are randomly applied on training images, in order to vary their appearance and virtually introduce new data on which a model can learn.

Data augmentations can modify the orientation of the images or operate at pixel level, transforming their color and appearance. Those operations can be geometric, if they geometrically distort the image without modifying its content, or color-based, if they distort color channels without modifying the position of the pixels. Common transformations include, but are not limited to, horizontal and vertical flip, rotation, translation and color shift. For an object detection task, bounding boxes have to be updated coherently with the applied geometric data augmentation.

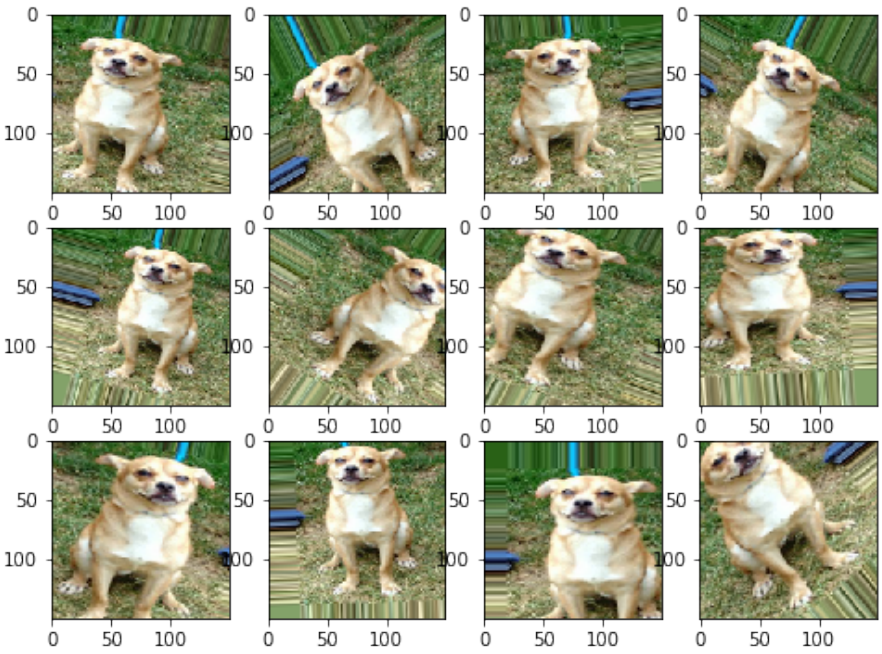


Figure 2.1: A few examples of data augmentation

Therefore, data augmentation techniques can extensively alter the image appearance. If a model is trained with high-impact data augmentation on labeled images from a domain, it can potentially achieve good results on a different domain it has

never seen. That falls in the case of **domain randomization**: training images are drawn from a data distribution which is way larger than the original one, so the model can learn to generalize to different domains. In some ways, the network is "overwhelmed" with different versions of the same images, in order to recognize their contents in different conditions. For domain adaptation, the resulting data distribution can possibly include the target domain or intermediates domains that are closer to the target domain than the source domain. Intuitively, performances on non-augmented images drawn from the source domain decrease as the data distribution gets larger, while the generalization capability to remote target domains increases.

Most works that investigate domain randomization focus on *sim2real* knowledge transfer and address the problem generating a large variety of synthetic training images. The famous one by Tobin *et al.*[44] defines different randomization parameters that control, inter alia, position, number and type of the objects present in an image, as well as lighting and camera conditions. An interval of values is defined for each parameter and, for generating an image, each parameter is uniformly sampled within its range and used to control the generator. A detection model is then trained on the generated synthetic images and should be able to generalize to real images containing the same kind of objects. Examples are reported in Figure 2.2.

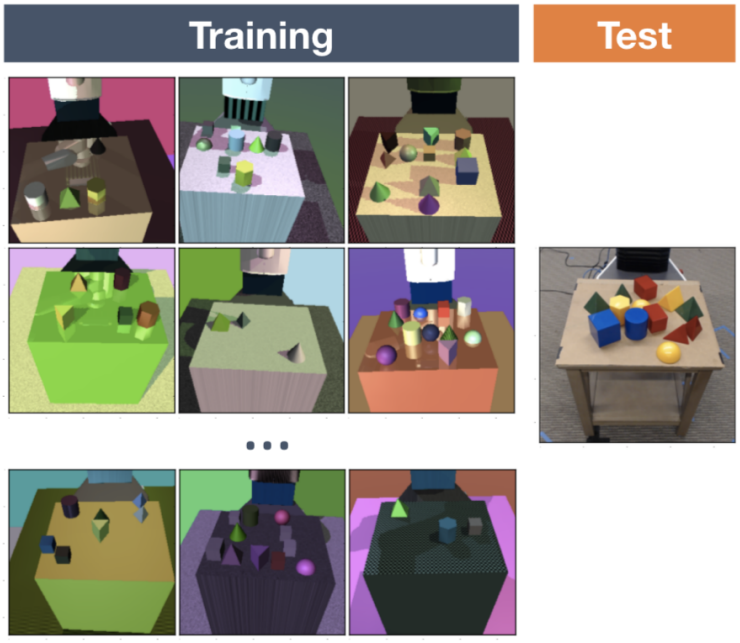


Figure 2.2: Randomization in generating training images, in the framework proposed by Tobin *et al.*[44]

In the present work, instead, randomization is introduced through the combination of popular and commonly used operations for augmentation of image data, both geometric and color-based:

- *Horizontal flip*: geometric transformation that horizontally mirrors the image.
- *Rotation*: geometric transformation that rotates image and bounding boxes of a degree value, sampled from a provided interval.
- *Translation*: geometric transformation that horizontally and vertically translates image and bounding boxes of a given number of pixels, sampled from a provided interval.
- *Shearing*: geometric transformation that deforms the image in a parallelogram shape according to horizontal or vertical axis. Horizontal and vertical shearing degrees are sampled from a provided interval.
- *Color jitter*: color-level transformation that changes brightness, contrast, saturation and hue of the image. Jitter factors for each of the properties are sampled from provided intervals.
- *Solarization*: color-level transformation that, for an image, inverts the color of all pixels whose value is above a threshold. The threshold is sampled from a provided interval.
- *Equalization*: color-level transformation that equalizes the color histogram of the image.

The operations can be easily implemented and used in most popular deep learning frameworks.

Many other operations are often used for data augmentation, but are not included in the performed experiments due to time constraints and since the possible combinations of the aforementioned operations are already numerous. An example is *cutout*, which randomly masks regions of the training images in order to improve networks' robustness. Other operations, such as *vertical flip*, do not make any sense in this context. Indeed, vertical flip would completely change the appearance of objects and background, leading to scenes that can never (or extremely rarely) appear in the real or simulated world.

More advanced data augmentation techniques, first introduced by Zoph *et al.*[51], are specifically designed for the object detection task: they take into account the bounding boxes and only apply data augmentation operations to the objects they contain or to the background. The original work aims at finding the best combination of data augmentation operators, given the task and the training data. This

(a) Original *Sim10k* image

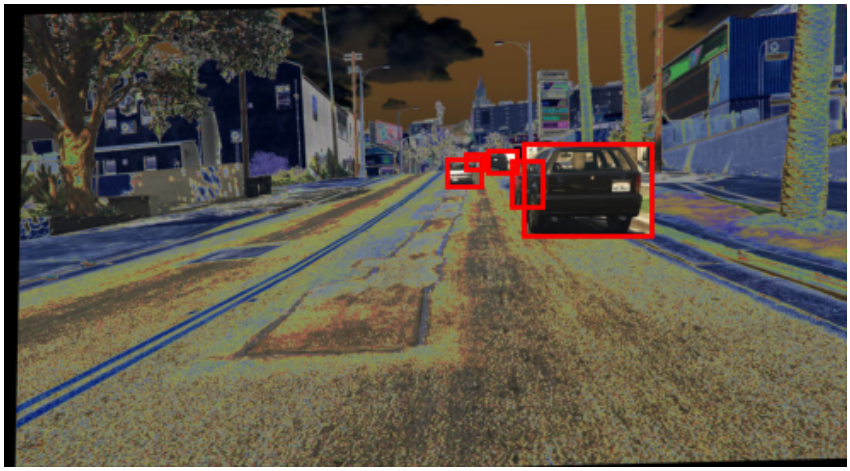
(b) *Sim10k* image with random augmentations

Figure 2.3: Comparison between original and transformed images

combination is sought with reinforcement learning and should lead to the best performances when training a model for image classification or object detection. A few examples of the proposed transformations are reported in Figure 2.4.

In a data randomization scenario, the data augmentation operators presented in the aforementioned work can be exploited to increase the variability of training images and widen the data distribution associated to the source domain. However, these peculiar transformations have to be used carefully: geometric operations can lead to unintended results, if applied to bounding box content or to background only. For example, if the bounding box is small and its content is translated horizontally, the object may disappear if the number of translated pixels is too high (see, for example, Figure 2.5b). Likewise, geometric transformations on background

(c) *Sim10k* image with random augmentations



(d) *Sim10k* image with random augmentations

Figure 2.3: Comparison between original and transformed images (cont.)

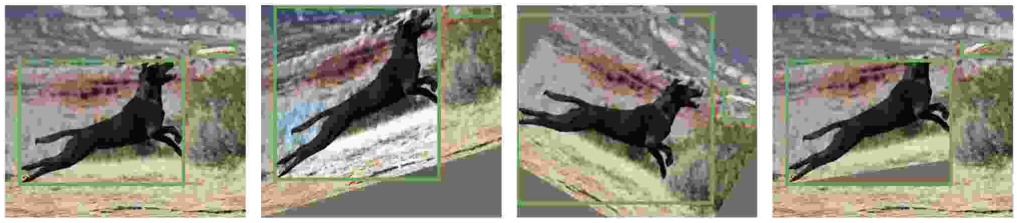


Figure 2.4: Examples of usage of the data augmentation operators introduced by Zoph *et al.*[51]

(the whole image except the content of bounding boxes) do not make sense, as they inevitably duplicate objects without associated bounding box. This bad behavior

is clear in Figure 2.5d. In both cases, this would produce erroneous training images and lead to incorrect training procedures.

A few images produced by these wrong transformations are reported in Figure 2.5.

(a) Original *Sim10k* image

(b) Horizontal translation of objects inside bounding boxes

Figure 2.5: Examples of unsuccessful geometric transformations on bounding boxes or image background only

2.3 Adapting a two stage object detector

Domain Adaptive Faster R-CNN[3], presented in Section 1.3.2, is without any doubt the most popular method designed to attain successful object detection in different domains. Since its introduction, the paper has been cited hundreds of times and

(c) Rotation of objects inside bounding boxes

(d) Rotation and translation of image background, keeping the objects in the original bounding boxes fixed

Figure 2.5: Examples of unsuccessful geometric transformations on bounding boxes or image background only (cont.)

several methods based on it have been proposed. It is therefore a good adaptation method to be used as baseline and for comparison with other architectures and methods.

The training process considers annotated images from the source domain and unlabeled images from the target domain at the same time. Adaptation is performed at two different levels using the gradient reversal layers presented in Section 1.3.1:

- Image level: the features extracted by the network backbone are forwarded to a patch-based domain classifier. Such patch classifiers, introduce by Isola

et al.[20], divide each feature map of an image into $N \times N$ patches and classify them independently. For a domain classifier, each patch is classified as extracted from source or target domain. In this way, the importance of the global differences inside images is reduced and the classifier is trained more precisely. Adaptation at this level aims at reducing the global differences such as image style and illumination.

- Instance level: the features extracted by the RPN are passed to a domain discriminator before being fed to the label classifier and bounding box regressor for the final prediction. That helps to reduce differences related to the objects that appear in the images, such as object size, texture and viewpoint.

In addition, a consistency regularization is added to encourage similar predictions from the two domain classifiers. A method overview is reported in Figure 2.6.

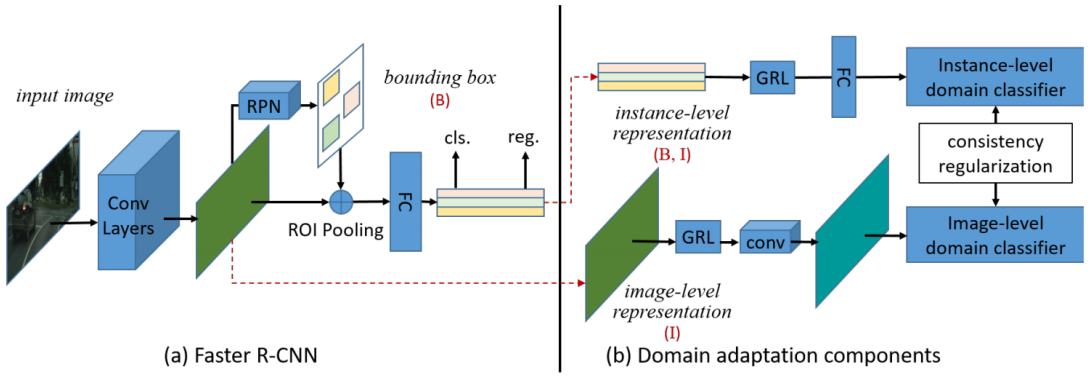


Figure 2.6: Overview of the Domain Adaptive Faster R-CNN model[3]

A training batch is composed of both source and target images and the training procedure is organized as follows:

- When the network receives, as input, labeled images from the source domain, they are used to train both the base Faster R-CNN and the domain adaptation components.

- When the network receives, as input, unlabeled images from the target domain, they are only used to train the domain adaptation components. Indeed, unlabeled images are not useful for the base detector as object bounding boxes and class information are not available.

For inference, additional domain adaptation components are not considered and the standard Faster R-CNN with learned weights is used.

2.4 Adapting the FPN

Feature Pyramid Network (FPN), first introduced by Lin *et al.*[24], is a cheap but effective method to increase the performances of a detection model. Nowadays, it is often included in the backbone network of object detectors such as Faster R-CNN and is one of the fundamental bricks of the RetinaNet model. One of the most popular computer vision frameworks for research, *torchvision* by *PyTorch*[33], only provides pretrained backbones with FPN for Faster R-CNN models. Therefore, using FPN for object detection problems is straightforward and highly performing with respect to standard backbones without FPNs, that are rarely or never used in real applications.

However, the FPN architecture introduces some problems for adversarial feature alignment on the output of the backbone (such as the *image* level in the Domain Adaptive Faster R-CNN model): backbone layers must be aligned in a way that the extracted feature pyramid representation is logical and coherent. Consider, for example, a case in which domain alignment is performed only on the last layer of the backbone: at the end of the training procedure, the features extracted from the last layer would be aligned between the domains, while the features extracted from the previous layers may not be so. As FPNs construct rich feature representations by linking the features extracted from different backbone layers, that would lead to a meaningless feature pyramid.

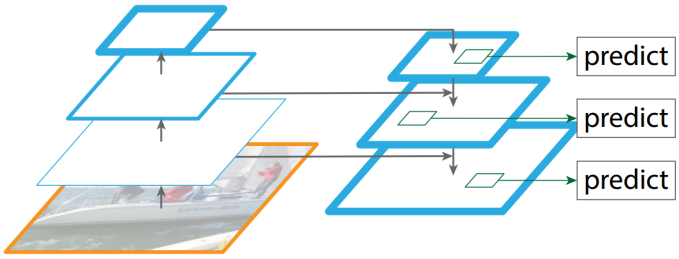


Figure 2.7: FPN architecture[3]

2.4.1 Implementation on RetinaNet

Domain Adaptive RetinaNet, introduced in Section 1.3.2, takes into account adversarial domain adaptation for a one-stage object detector with FPN backbone. More precisely, in the RetinaNet model the FPN uses the features extracted from the last three layers of a ResNet to construct a rich and multiscale feature pyramid of three levels from an input image at single resolution. This representation is then forwarded to two separate fully convolutional subnetworks, in order to predict the probability of presence of an object in each anchor box and the bounding box locations.

As FPNs return feature maps of different sizes, considering all of them at the same time for adaptation is not straightforward and may lead to non-optimal performances. Pasqualino *et al.*[31] perform adaptation through three different domain classifiers with different architectures. They receive as input the features extracted from the three ResNet layers that are used to build the FPN representation.

For adversarial alignment, the features are forwarded through gradient reversal layers as in DANN (described in Section 1.3.1): the overall network is trained normally but the gradient resulting from the domain classifiers is multiplied by a negative factor $-\lambda$, as in (1.10), during the backpropagation phase, in order to push the ResNet to extract domain invariant features that can be used to accurately detect objects in target images. This allows the model to preserve a meaningful feature pyramid while aligning the two data distributions.

Labeled source images are used to train the whole network, while unlabeled target images only train the three domain classifiers. At test phase, the domain classifiers are plugged off and the adapted RetinaNet is normally used.

2.4.2 Implementation on Faster R-CNN

The addition of a FPN component to the backbone of a Faster R-CNN model considerably improves detection performances. For example, the authors of the paper that introduced the FPN method reported 3.7 AP points improvement on the COCO dataset[24]. Thanks to the FPN, RPN receives a rich feature representation at different resolutions and is able to better identify candidate RoIs of higher quality. As it can achieve better results than the standard architecture without FPN at a negligible price, Faster R-CNN with FPN backbone is commonly used.

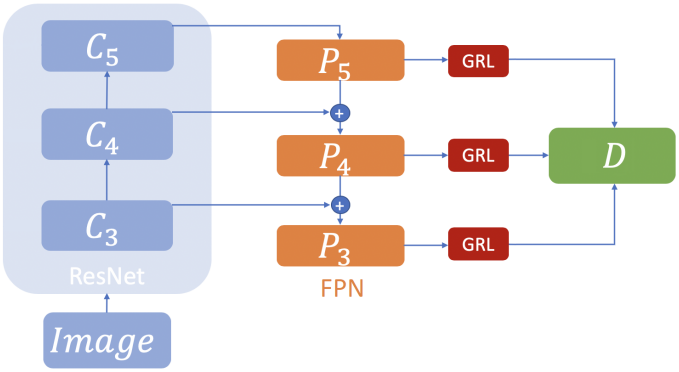
If the base detector includes a FPN component, adaptation at image level as in the Domain Adaptive Faster R-CNN method has to be reconsidered. Two possible solutions can be identified in order to align the feature representations between different domains:

- Adaptation can be directly performed on features extracted by the FPN. Different levels of the feature pyramid can be forwarded, one at a time, to a domain classifier through gradient reversal layer (Figure 2.8a). The resulting domain losses can be summed or averaged across all levels. This way, the objective is to push the ResNet backbone to extract feature representations that can be used to build a feature pyramid which is independent of the domain of each input image.

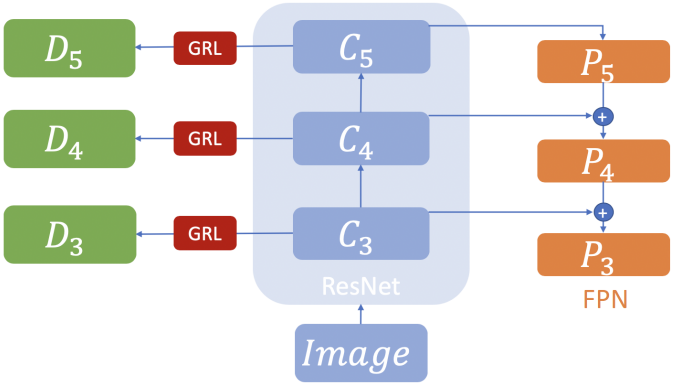
- Similarly to what is done by Pasqualino *et al.*[31], it is otherwise possible to directly align the ResNet features that serve as input for the FPN: if they are not dependent from the domain, neither the resulting pyramid representation will be. The last three ResNet layers are therefore used to build the feature

pyramid and are forwarded to three different domain classifiers with different architectures, through gradient reversal layers for adversarial alignment (Figure 2.8b).

Such approaches have not yet been followed in the academic literature for two stage object detectors. In both cases, object detection network and domain adaptation components are trained jointly.



(a) Adversarial alignment on pyramidal feature representations, built by the FPN from ResNet layers



(b) Adversarial alignment on features extracted by ResNet layers[31]

Figure 2.8: Approaches to adversarial domain adaptation at image level for an architecture with FPN

The presence or absence of the FPN in the backbone has no impact on adaptation at instance level, as RPN returns RoIs in the same format. If different domain classifiers are used, the consistency regularization loss is independently computed for each ResNet layer between the domain prediction and the prediction at instance level, unique at each iteration. Otherwise, training proceeds as previously described.

2.5 Image statistics matching

As discussed in Section 2.2, data augmentation techniques that significantly modify the appearance of images taken from a specific domain extend their data distribution, possibly covering several domains. A model can be trained on transformed images from the source domain without any architectural modification, possibly performing well on images drawn from a target data distribution.

Likewise, domain mapping methods transform images from a source domain in a way that a model can confuse them with images from the target domain: they focus on accurate modifications of images rather than upgrading base architectures. That is typically attained through GANs for style transfer, but training of adversarial networks is generally complicated and time-consuming. Instead, the method introduced by Abramov *et al.*[1] tries to "keep it simple", by performing easy but effective operations at image level with the objective of matching image statistics between source and target domains.

More precisely, for each labeled image from the source domain, an unlabeled image belonging to the target domain is randomly selected. Two transformations are applied one after the other:

1. Feature Distribution Matching: the first operation transforms the source image in a way that it obtains the same color mean and covariance of the target image, while keeping the image content. It works pixel-wise through Principal Component Analysis (PCA) whitening with Singular Value Decomposition, in order to efficiently uncorrelate the distribution of image pixels.

2. Histogram Matching: the second transformation is a common one in image processing. It manipulates the pixels of the source image in order to align the distributions of its color histograms to the ones of the target image.

A sample application of the method is reported in Figure 2.9.

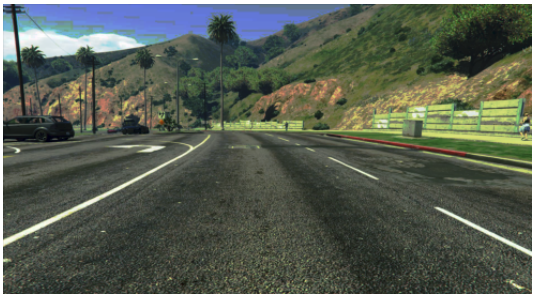
A detection model is therefore trained on labeled mapped images that share statistics with target images and can thus be considered as belonging to the target domain. The model can then be directly used for inference on the target domain, without any additional operation.

Additionally, transformations can be applied on a patch basis, in order to locally match images from source and target domains. Aforementioned transformations can be performed on smaller regions of the images, in correspondence between source and target domain. This way, matching focuses on local features and is less dependent on the global appearance of selected images from the two domains. An example of application of patch-based Feature Distribution Matching and Histogram Matching is reported in Figure 2.10.

(a) Source image

(b) Target image

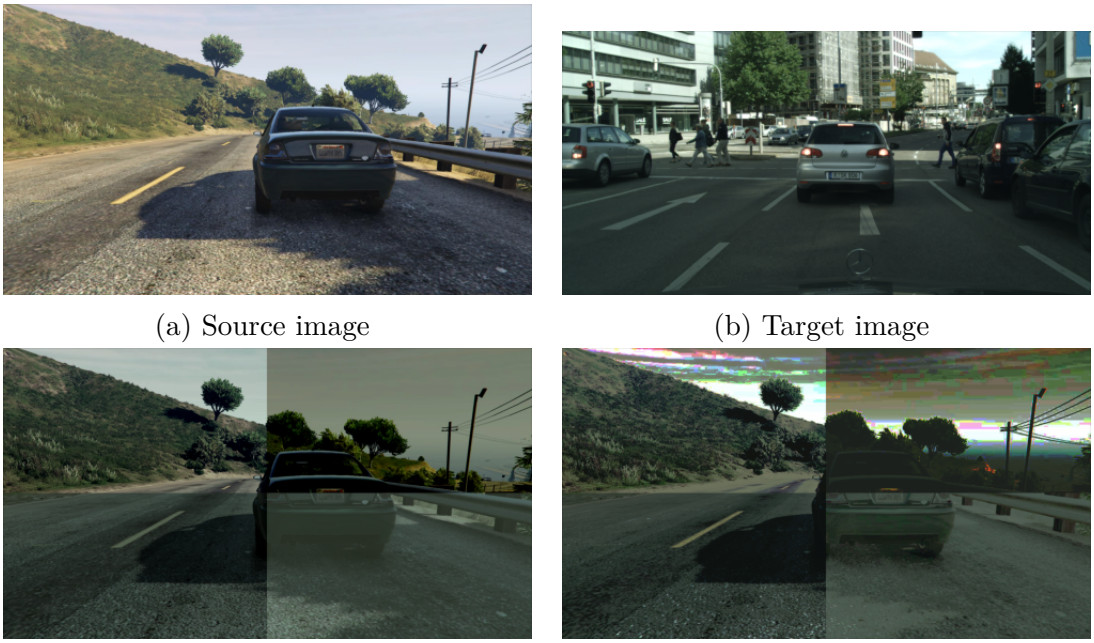
(c) Feature Distribution Matching



(d) Histogram Matching

(e) Combination of Feature Distribution Matching and Histogram Matching

Figure 2.9: Results obtained through Feature Distribution Matching and Histogram Matching on a sample image. *Sim10k* and *CityScapes* are considered as source and target domains respectively.



(a) Source image

(b) Target image

(c) Feature Distribution Matching

(d) Histogram Matching

Figure 2.10: Results obtained through 2×2 patch-based Feature Distribution Matching and Histogram Matching on a sample image. *Sim10k* and *CityScapes* are considered as source and target domains respectively.

Chapter 3

Experimental results

3.1 Experimental settings

This chapter reports, compares and discusses the results obtained with the methods presented in the previous chapter. For coherence with previous works, *Sim10k* and *CityScapes* are respectively considered as source domain and target domains. Experiments deal therefore with *sim2real* domain adaptation, training a model to recognize cars in a simulated world for detection in the real. Results are presented in terms of mAP computed at IoU threshold of 0.5, separately for the experiments performed with RetinaNet and Faster R-CNN base detectors. Impact of data augmentation operations, in addition to domain adaptation methods, is also studied. All experiments have been conducted with the PyTorch deep learning framework and run on an NVIDIA GeForce RTX 2070 Graphics Processing Unit (GPU) with 8 GB of memory.

In the following sections, the models indicated as "oracle" are trained and tested directly on the target domain (*CityScapes* in the *sim2real* context) using the base detectors and without applying any data augmentation technique, except for the simple horizontal flip. This represents an upper bound for domain adaptation performance. Such a model cannot be trained in real applications, as, in the common case of unsupervised domain adaptation, annotations are not available for the target domain.

3.1.1 Data processing

As said, *Sim10k* and *CityScapes* are considered as source and target domain, for unsupervised domain adaptation between synthetic and real image data. As mentioned in Section 1.2.2, the two dataset use different formats for storing bounding box annotations. The Python library *Datumaru*[30] is therefore used to convert them into a common format.

As an official Python interface is provided to easily load the annotations and compute the detection mAP, the Microsoft COCO format has been chosen. Only annotations for the class "car" are kept, as this is the only common category in the two datasets containing street scenes.

Finally, the open source tool *cocosplit*[23] is used to partition the *Sim10k* dataset into training, validation and test sets with a proportion of 7:2:1, as standardized set splits are not officially available. The dataset is quite large (10000 annotated images) and it can be assumed that image distributions are similar between the three splits. On the other hand, splits are made public for *CityScapes*, but annotations are not available for the test images. The performances of the tested domain adaptation methods are therefore evaluated on the provided validation set, as it is never used during training.

When fed to the object detection models for training or evaluation, input images are resized so that the shortest side is equal to 600 pixels.

3.1.2 Data augmentation

Impact of data augmentation is extensively tested, for both domain randomization by itself or in combination with other methods for domain adaptation. The following parameters are used for each previously introduced data augmentation technique:

- random rotation (`Rot.`) between -10 and 10 degrees;
- random translation (`Tr.`) between -100 and 100 pixels, independently chosen for the two axis;
- random shearing (`Sh.`) between -20 and 20 degrees (not always used);
- brightness, contrast, saturation and hue randomly jittered (`C.J.`) between 0.5 and 1.5, independently chosen for each property;
- for solarization (`Sol.`), the threshold over which pixels are inverted is randomly sampled between 64 and 192 (with RGB pixel values going from 0 to 255 for each color channel).

Additional performed data augmentation techniques are horizontal flip (`HF`, a basic transformation which is always used) and equalization (`Eq.`). Both operations are parameter-free and are applied with probability 0.5 each.

Notations in parentheses are referred to when reporting the results in the next sections.

The Python library *imgaug*[22] is exploited for data augmentation.

3.1.3 RetinaNet

The open-source RetinaNet implementation included in the PyTorch's library for computer vision (*torchvision*)[37] has been used as starting point for the experiments with the base object detection model, which included domain randomization and image statistics matching. For Domain Adaptive RetinaNet, the implementation provided by the same authors of the original paper[31] using the *detectron2* framework[49][32] is followed and adapted.

All used RetinaNet networks include a ResNet-50 backbone and are pre-trained on the COCO dataset. They are then finetuned on the data from one or both domains using the Adam optimizer for 15 epochs, with initial learning rate 1×10^{-4}, reduced to 1×10^{-5} after 10 epochs. A training batch is composed of four annotated images from the source domain. When the target domain is needed for adversarial adaptation, two images are added to each training batch without considering image annotations, for a final batch size of 6.

Additional configurations for RetinaNet include:

- anchor boxes of size 32, 64, 128, 256 and 512 are generated with aspect ratios 0.5, 1 and 2, for a total of 15 anchor boxes for each location of the feature map returned by the ResNet backbone;
- best 1000 bounding box proposals are kept before NMS, and only if detection confidence is greater than 0.05;
- best 300 bounding boxes are returned after NMS, performed with IoU threshold 0.5.

The focal loss hyperparameters, as in (1.6), are set to $\alpha = 0.25$ and $\gamma = 2$.

When adversarial adaptation is applied on the last three layers of the ResNet-50 backbone (layers 3, 4 and 5), the λ coefficient introduced in (1.10) evolves from 0 to 1, and is limited to 0.5 for layers 3 and 4, and to 0.1 for layer 5. The dynamic update is performed independently for each adversarial coefficient following the policy illustrated in (1.11). In this last equation, γ is set to 10.

3.1.4 Faster R-CNN

As for RetinaNet, *torchvision*'s implementation of Faster R-CNN model is used as basic detector[9]. For Domain Adaptive Faster R-CNN, an implementation by the authors of the paper[3] is considered[4], but is reimplemented in a more recent version of PyTorch and *torchvision*. The original implementation is only used for testing the method without FPN.

As for RetinaNet, all Faster R-CNN models include a ResNet-50 backbone and are pre-trained on COCO. Networks are then finetuned on *Sim10k* (annotated)

and *CityScapes* (without annotations) for 15 epochs using the SGD optimizer with momentum of 0.9 and weight decay of 5×10^{-4}. Initial learning rate is set at 1×10^{-3} and is reduced to 1×10^{-4} after 10 epochs. In order to improve stability of two-stage object detectors, the learning rate is initialized using a warm-up policy. It is linearly increased from 0 to 1×10^{-3} during the first 1000 training iterations.

Batch size of 4 is used when the model is trained only on annotated images from the source domain. When images from the target domain are used for adversarial domain adaptation the size of a training batch is 2: it is composed of one annotated image from the source domain and one unlabeled image from the target domain.

Additional configurations for Faster R-CNN include:

- anchor boxes of size 32, 64, 128, 256 and 512 are generated with aspect ratios 0.5, 1 and 2, for a total of 15 anchor boxes for each location of the feature map returned by the ResNet backbone;
- during training, the RPN extracts 1000 bounding box proposals after NMS, performed with IoU threshold 0.7;
- during inference, 100 bounding boxes are returned after NMS, performed with IoU threshold 0.5, and only if their detection confidence is higher than 0.05.

When adversarial domain adaptation is performed, the adversarial coefficient λ is set to 0.1.

3.2 Domain randomization

Table 3.1 and Table 3.2 contain the results obtained by RetinaNet and Faster R-CNN (with FPN backbone) respectively, trained on *Sim10k* images using several data augmentation techniques and evaluated on *CityScapes* test images. Images belonging to the target domain as considered as they are for evaluation, without applying any transformation. For each combination of data augmentation operations, the tables contain the detection mAP on *CityScapes*.

The resulting mAP values confirm the hypothesis of Section 2.2: the random component introduced by the fundamental data augmentation operations helps to reduce the mismatch between source and target domains. Presumably, the source data distribution is expanded such that the intersection between the source and target distributions becomes larger too.

Comparing the two tables, it can be noticed that the Faster R-CNN base detector with FPN, trained with horizontal flip only, performs considerably better than RetinaNet with the same transformation (difference of 15.9 mAP) and without having seen any image belonging to the target domain, thanks to the use of the

Table 3.1: mAP with different data augmentation techniques for a RetinaNet model, trained on *Sim10k* and evaluated on *CityScapes*. Table columns indicate the transformations that are employed in each experiment. Only horizontal flip is used for the "oracle".

HF	Rot.	Tr.	Sh.	C.J.	Sol.	Eq.	mAP@0.5
✓							30.4
✓	✓	✓					37.6
✓	✓	✓	✓				39.2
✓	✓	✓		✓			49.4
✓	✓	✓		✓	✓	✓	**52.7**
✓	✓	✓	✓	✓	✓	✓	49.8
Oracle							68.7

Table 3.2: mAP with different data augmentation techniques for a Faster R-CNN model with FPN, trained on *Sim10k* and evaluated on *CityScapes*. Table columns indicate the transformations that are employed in each experiment. Only horizontal flip is used for the "oracle".

HF	Rot.	Tr.	C.J.	Sol.	Eq.	mAP@0.5
✓						46.3
✓	✓	✓				48.8
✓	✓	✓	✓			**57.9**
✓	✓	✓	✓	✓	✓	**57.9**
Oracle						76.3

features extracted by the FPN in the IoU proposal phase. The feature extractor helps in extracting relevant features from the different backbone levels, which are used for the RoI proposal. This helps to identify low-level image features that are common to different domains, such as tires or windows for cars, and capture them in a meaningful way.

The addition of simple geometric transformations (rotation, translation and shearing) helps both networks to better learn important features and has a large impact on RetinaNet in particular (+7.2 mAP, +2.5 for Faster R-CNN). The usage of shearing further improves the detection performances of the RetinaNet model (+1.6 mAP).

The important contribution of randomization introduced by the data augmentation techniques starts to become evident if color operators are used: the large

variation in brightness, contrast, saturation and hue of the images makes the detection results rise on *CityScapes* (+11.8 for RetinaNet, +9.1 for Faster R-CNN). At this point, the additional solarization, equalization or shearing improve the final performances slightly or not at all.

3.3 Domain adaptation methods

Table 3.3 and Table 3.4 present the results obtained by RetinaNet and Faster R-CNN networks, with the domain-invariant feature learning and domain mapping approaches described in the previous chapter. Models are always trained on *Sim10k* and evaluated on *CityScapes*. Contribution of several data augmentation techniques in addition to the methods themselves is also analyzed, and related performance variations are reported. In Table 3.4, the `FPN` column states if a FPN is used or not as part of the backbone.

For Domain adaptive Faster R-CNN, `RN` means that adversarial alignment is performed at the level of the last three ResNet layers before being used to build the feature pyramids, as done in Domain Adaptive RetinaNet[31] and described in Section 2.4.2. Instead, the entry with the simple ✓ contains the result obtained with adversarial adaptation on the network's FPN.

For the RetinaNet detector, tested adaptation methods (image statistics matching and alignment on FPN) performed well, with an increase over the baseline of 17.8 and 22.4 mAP respectively. The use of data augmentation does not have a considerable impact for the Domain Adaptive RetinaNet, while color transformations remarkably increase the detection performances when matching image statistics between source and target domains.

Regarding Faster R-CNN, adaptation at image and instance levels yields improvements with respect to the basic Faster R-CNN architecture without FPN, consistently with previous works[1][3][40].

On the other hand, adding a FPN to the network backbone (a scenario that was not considered in the previously mentioned works) significantly increases the performances of the base detector (around +12.6 mAP) and mAP results are way better than the ones obtained through image and instance level adaptation without FPN, even on a domain it has never seen for the already mentioned reasons. Adapting the FPN by gradient reversal layer gives no contribution to the knowledge transfer and even degrades the mAP values when all levels of the feature pyramid are considered at the same time for adversarial domain alignment. If different discriminators are used to align different ResNet layers, the information acquired by the FPN is preserved (+0.5 mAP). Since the Faster R-CNN network without FPN can be considered as outdated and adversarial adaptation on the features extracted

Table 3.3: mAP with different domain adaptation method for a RetinaNet model, trained on *Sim10k* and evaluated on *CityScapes*. Table columns indicate the transformations that are employed in each experiment. Only horizontal flip is used for the "oracle".

Method	HF	Rot.	Tr.	C.J.	Sol.	Eq.	mAP@0.5
Baseline	✓						30.4
	✓	✓	✓	✓	✓	✓	52.7
Image statistics matching	✓						48.2
	✓	✓	✓				52.0
	✓	✓	✓	✓			**62.7**
DA RetinaNet	✓						52.8
	✓	✓	✓				54.2
	✓	✓	✓	✓			54.8
Oracle	✓						68.7

Table 3.4: mAP with different domain adaptation method for a Faster R-CNN model, trained on *Sim10k* and evaluated on *CityScapes*. FPN column states if FPN is used or not (`RN` means different domain discriminators are applied to the three ResNet layers used to build the FPN). The other columns indicate the transformations that are employed in each experiment. Only horizontal flip is used for the "oracle".

Method	FPN	HF	Rot.	Tr.	C.J.	Sol.	Eq.	mAP@0.5
Baseline		✓						33.7[1]
	✓	✓						46.3
	✓	✓	✓	✓	✓	✓	✓	57.9
Image statistics matching	✓	✓						55.0
	✓	✓	✓	✓				62.0
	✓	✓	✓	✓	✓			**62.5**
DA Faster R-CNN		✓						36.3
	✓	✓						45.0
	RN	✓						46.8
Oracle	✓	✓						76.3

Table 3.5: mAP with different combinations of data augmentation techniques, independently applied to whole images (`I`), bounding boxes (`O`) or backgrounds (`B`) only, for a RetinaNet model trained on *Sim10k* and evaluated on *CityScapes*. Table columns indicate the transformations that are employed in each experiment. `y` means operation is only applied on the vertical axis. Only horizontal flip is used for the "oracle".

HF	Rot.	Tr.	Sh.	C.J.	Sol.	Eq.	mAP@0.5
I							30.4
I	I	I + O	I			I + O	21.0
I			O	B		O	29.1
I	I	Oy				I	**34.1**
Oracle							68.7

by the backbone with FPN is not efficient, further improvements through data augmentation operators are not sought for Domain Adaptive Faster R-CNN.

Finally, image statistics matching proves to be effective for domain adaptation on Faster R-CNN with FPN as well (+8.7 mAP), even though it is a very simple method. Statistics matching is even more efficient if additional geometric transformations are used.

3.4 Additional experiments

3.4.1 Bounding box data augmentation

A few extra experiments have been performed with data augmentation techniques on bounding boxes, mentioned in Section 2.2. In this case, the method is only tested on RetinaNet. All the transformations described above, applied to the whole images, to bounding boxes or to backgrounds only, add a considerable number of additional combinations. It is not obvious a priori which ones can have a more important contribution than the others, with the goal of effectively reduce the distance between source and target domains. The tested combinations are therefore selected almost randomly, mostly considering the operations that contributed to achieving the best results in the original work by Zoph *et al.*[51], but obtained results (reported in Table 3.5) are not satisfactory. Actually, they often degrade the performances of the basic detector.

Ostensibly, the achievements of different combinations highly depend on the dataset, and so they should be investigated thoroughly. For this reason, these transformations have not been explored in detail.

Table 3.6: mAP with different domain adaptation method for RetinaNet and Faster R-CNN models, trained on *Sim10k* and evaluated on *CityScapes* with patch-based image statistics matching.

Model	Method	mAP@0.5
RetinaNet	Baseline	30.4
	1×1 matching	**48.2**
	2×2 matching	45.0
	Oracle	68.7
Faster R-CNN	Baseline	46.3
	1×1 matching	55.0
	2×2 matching	**55.5**
	Oracle	76.3

3.4.2 Patch-based domain mapping

A similar reasoning as bounding box augmentations is followed for experiments on patch-based image statistics matching. Matching on 2×2 grids is only tested, on RetinaNet and Faster R-CNN with FPN, and results are reported in Table 3.6. Again, results are neither completely satisfactory nor explanatory. The method would therefore require further study.

3.5 Final considerations and possible research directions

Reported results show that implemented approaches are indeed effective for domain adaptive object detection, from a computer-generated source domain to a real-world target domain. Extremely simple modifications of the appearance of training images have high impact on detection performances: the best results, on both RetinaNet and Faster R-CNN detectors, are obtained through domain mapping based on matching the image statistics of source and target domains. This method works without any architectural modifications and does not require any annotation for the target domain.

Prior to this, performed experiments clearly show that domain randomization alone, introduced through basic data augmentation operators, can return satisfactory results and considerably reduce the shift between source and target domain. The method is extremely simple to set up and experiment with and, if combined

with fast one stage object detectors such as RetinaNet, can help to achieve good performances in a short time. It is therefore suitable for real applications and for use in production environments. In addition, the experiments also showed that the data augmentation component combines well with existing domain adaptation methods. Their matching significantly improved the results obtained with a single domain adaptation method, either based on domain mapping or domain-invariant feature learning. Furthermore, domain randomization approaches can be easily extended to all computer vision tasks, and are not limited to object detection.

On the other hand, widely-studied methods for domain adaptation, mostly based on adversarial feature learning and requiring deep modifications to the network architectures, have reached similar or lower performances than these simple methods, despite requiring much more time to investigate and implement. Furthermore, often it is not possible to transfer these methods from one base detector to another. They are surely interesting from a research point of view, but their actual viability to real use cases seems uncertain, especially by engineers that may have little or no experience in the domain adaptation setting.

Studied domain adaptation approaches present interesting open questions that can be further studied. Among them, how to align the features extracted by a backbone with FPN is probably the prominent one. Although adversarial adaptation of the ResNet layers that are used to build the feature pyramid, as proposed by Pasqualino *et al.*[31], is particularly effective for the RetinaNet model, it does not generalize well to two stage object detectors as Faster R-CNN, in which the FPN is used to extract interesting RoIs. Due to the higher detection performances with respect to one stage architectures, Faster R-CNN and the like are mostly used for applications in which high accuracy is preferable over high speed.

In a recent work, Chen *et al.* have tackled this issue proposing what they call *Scale-Aware* Domain Adaptive Faster R-CNN[5]. They extend their previous Domain Adaptive Faster R-CNN (widely discussed in the preceding sections), applying adversarial adaptation at image and instance level independently for each pyramid level. It might be interesting to study this method more specifically.

Additionally, image transformations that take into account the bounding boxes could be explored more broadly for domain randomization. Taking inspiration by the work by Zoph *et al.*[51], it may be interesting to study a way to find an optimal data augmentation pipeline starting from the available data for source and target domains.

However, when performing domain adaptation, one must be aware of the nature of the two domains. They could impose limitations on the performances of even the best of the methods.

Consider the discussed *Sim10k* and *CityScapes* datasets. Neglecting for a moment their different nature (synthetic and real datasets), the street scenes contained

in the two datasets are largely different: *CityScapes* contains images captured in the center of several German cities, while the video-game *Grand Theft Auto V* is set in a fictional region of Southern California. Scenarios are clearly diverse. This means that, for example, surrounding buildings and more popular car models are different in the two datasets. In addition, *CityScapes* may contain more scenes with high numbers of pedestrians, while *Sim10k* contains landscapes (such as desert areas) which are unlikely to be found in Europe. Discussed domain adaptation methods cannot deal with such structural difference between domains.

Chapter 4

Conclusions

In a real production environment, being able to successfully transfer the knowledge learned by a deep learning model from one domain to another can introduce considerable improvements in many projects, new and already in place, and increase general productivity. This would allow the same data to be used in multiple projects that deal with the same task. More extremely, data collection and annotation may not be necessary anymore, once a generator of synthetic data is available and domain adaptation from simulated to real data is deployed effectively. Many methods address the domain adaptation problem in the different computer vision domains: image classification, object detection and semantic segmentation.

Focusing on object detection, most studied methods in the literature involve architecture-specific modifications. However, all this complexity seems excessive when very simple methods can obtain similar or even better detection performances. Overall, domain mapping based on image statistics can yield results that approach models with full knowledge of the target domain, and perform even better when supported by data augmentation techniques. The domain randomization introduced through classic image transformations also helps to reduce the mismatch between the two data distributions and may be sufficient for several use cases. However, domain adaptation is an open problem, for which research is always active. New promising methods and innovative approaches can be published from one day to another.

Bibliography

[1] Alexey Abramov, Christopher Bayer, and Claudio Heller. *Keep it Simple: Image Statistics Matching for Domain Adaptation*. 2020. arXiv: 2005.12551 [cs.CV].

[2] Massih-Reza Amini. *Apprentissage machine: de la théorie à la pratique*. Editions Eyrolles, 2015.

[3] Yuhua Chen et al. *Domain Adaptive Faster R-CNN for Object Detection in the Wild*. 2018. arXiv: 1803.03243 [cs.CV].

[4] Yuhua Chen et al. *GitHub - krumo/Domain-Adaptive-Faster-RCNN-PyTorch: Domain Adaptive Faster R-CNN in PyTorch*. 2019. URL: https://github.com/krumo/Domain-Adaptive-Faster-RCNN-PyTorch.

[5] Yuhua Chen et al. «Scale-aware domain adaptive faster r-cnn». In: *International Journal of Computer Vision* 129.7 (2021), pp. 2223–2243.

[6] *Classification: ROC Curve and AUC | Machine Learning Crash Course*. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[7] Marius Cordts et al. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. 2016. arXiv: 1604.01685 [cs.CV].

[8] Navneet Dalal and Bill Triggs. «Histograms of oriented gradients for human detection». In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee. 2005, pp. 886–893.

[9] *Faster R-CNN - torchvision.models — Torchvision 0.11.0 documentation*. URL: https://pytorch.org/vision/stable/models.html#id57.

[10] Yaroslav Ganin and Victor Lempitsky. *Unsupervised Domain Adaptation by Backpropagation*. 2015. arXiv: 1409.7495 [stat.ML].

[11] Andreas Geiger et al. «Vision meets Robotics: The KITTI Dataset». In: *International Journal of Robotics Research (IJRR)* (2013).

[12] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[13] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

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

[15] Shirsendu Sukanta Halder, Jean-François Lalonde, and Raoul de Charette. *Physics-Based Rendering for Improving Robustness to Rain*. 2019. arXiv: 1908.10335 [cs.CV].

[16] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[17] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

[18] Han-Kai Hsu et al. *Progressive Domain Adaptation for Object Detection*. 2019. arXiv: 1910.11319 [cs.CV].

[19] Hanqing Hu, Jin Lyu, and Xiaolin Yin. «Research and Prospect of Image Recognition Based on Convolutional Neural Network». In: *Journal of Physics: Conference Series* 1574 (June 2020), p. 012161. DOI: 10.1088/1742-6596/1574/1/012161.

[20] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].

[21] Matthew Johnson-Roberson et al. *Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?* 2017. arXiv: 1610.01983 [cs.CV].

[22] Alexander B. Jung et al. *imgaug*. 2020. URL: https://github.com/aleju/imgaug.

[23] Artur Karaźniewicz. *GitHub - akarazniewicz/cocosplit: Simple tool to split COCO annotations into train/test datasets.* URL: https://github.com/akarazniewicz/cocosplit.

[24] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[25] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[26] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[27] Wei Liu et al. «SSD: Single Shot MultiBox Detector». In: *Lecture Notes in Computer Science* (2016), pp. 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[28] David G Lowe. «Object recognition from local scale-invariant features». In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[29] Luke Melas-Kyriazi. *Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet*. 2021. arXiv: 2105.02723 [cs.CV].

[30] OpenVINO™. *Dataset Management Framework (Datumaro)*. 2020. URL: https://github.com/openvinotoolkit/datumaro.

[31] Giovanni Pasqualino et al. *An Unsupervised Domain Adaptation Scheme for Single-Stage Artwork Recognition in Cultural Sites*. 2020. arXiv: 2008.01882 [cs.CV].

[32] Giovanni Pasqualino et al. *GitHub - fpv-iplab/DA-RetinaNet: Detectron2 implementation of DA-RetinaNet*. 2020. URL: https://github.com/fpv-iplab/DA-RetinaNet.

[33] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[34] *Precision-Recall — scikit-learn 0.24.1 documentation*. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

[35] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[36] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[37] *RetinaNet - torchvision.models — Torchvision 0.11.0 documentation*. URL: https://pytorch.org/vision/stable/models.html#id58.

[38] Stephan R. Richter, Hassan Abu AlHaija, and Vladlen Koltun. *Enhancing Photorealism Enhancement*. 2021. DOI: 10.48550/ARXIV.2105.04619. URL: https://arxiv.org/abs/2105.04619.

[39] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[40] Kuniaki Saito et al. *Strong-Weak Distribution Alignment for Adaptive Object Detection*. 2019. arXiv: 1812.04798 [cs.CV].

[41] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. «Semantic Foggy Scene Understanding with Synthetic Data». In: *International Journal of Computer Vision* 126.9 (Mar. 2018), pp. 973–992. ISSN: 1573-1405. DOI: 10.1007/s11263-018-1072-8. URL: http://dx.doi.org/10.1007/s11263-018-1072-8.

[42] Pierre Sermanet et al. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. 2014. arXiv: 1312.6229 [cs.CV].

[43] *The PASCAL Visual Object Classes Homepage*. URL: http://host.robots.ox.ac.uk/pascal/VOC/.

[44] Josh Tobin et al. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*. 2017. arXiv: 1703.06907 [cs.RO].

[45] Ilya Tolstikhin et al. *MLP-Mixer: An all-MLP Architecture for Vision*. 2021. arXiv: 2105.01601 [cs.CV].

[46] Eric Tzeng et al. *Adversarial Discriminative Domain Adaptation*. 2017. arXiv: 1702.05464 [cs.CV].

[47] Lilian Weng. *Object Detection Part 4: Fast Detection Models*. URL: https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html.

[48] Garrett Wilson and Diane J. Cook. *A Survey of Unsupervised Deep Domain Adaptation*. 2020. arXiv: 1812.02849 [cs.LG].

[49] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[50] Richard Zhang et al. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. DOI: 10.48550/ARXIV.1801.03924. URL: https://arxiv.org/abs/1801.03924.

[51] Barret Zoph et al. *Learning Data Augmentation Strategies for Object Detection*. 2019. arXiv: 1906.11172 [cs.CV].