UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
"TECNOLOGIE DELL'INFORMAZIONE"

XXXIII ciclo

Learning How To Perceive The Real World From Simulations

Coordinatore:
Chiar.mo Prof. Marco Locatelli

Tutore:
Chiar.mo Prof. Massimo Bertozzi

Dottorando: Luigi Musto

Anni accademici 2017/2018-2019/2020

Contents

Chapter 1

Introduction

1.1 Autonomous Driving

Autonomous driving has the potential to improve world transportation and reduce a lot the number of car crashes and human fatalities. Therefore, the scientific community is researching solutions to provide reliable self-driving cars. However, in order to outdo human driving performance, robots need a strong and robust perception system. In fact, in order to navigate in any environment, cars need the ability to detect objects and get a semantic understanding of their surroundings. These tasks require to process data from many sensors, which may include cameras and possibly LiDARs. Image processing, in particular, can be considered the most important source of semantic information, since it resembles how humans perceive the world.

1.2 Deep Learning for Computer Vision

Computer vision aims at giving machines a high-level understanding of the world through images. Many tasks, like depth estimation and obstacle detection, have always been solved through approaches based on geometric models and classical algorithms. Some other tasks, like classification and segmentation, had to rely on handcrafted features and databases that needed to be queried for recognition.

In the last decade, deep learning [1] has brought outstanding results in many tasks that seemed unfeasible, like the ones that require semantic understanding and human knowledge. Convolutional Neural Networks (CNNs), in particular, have been successful in computer vision since they mimic very well the human visual processing. Theoretically they can be trained for any task, depending on how their output layer is designed, but CNNs, like any model in machine learning, require data to be trained on. Deep CNNs have been shown to memorize very well the training data, and this is why they have such a success in computer vision tasks, especially if related to perception.

1.3 Supervised Learning

The most common setup for training neural networks is supervised learning. In this setting, the training data is comprised of a set of inputs (*e.g.* images) and their corresponding outputs (*e.g.* classes, bounding boxes, semantic maps, etc.). Given an input item, a neural network is optimized to predict an output corresponding to the desired one. This requires to formalize the network error through a loss function, which represents the distance between the predicted output and the correct one.

Still, any neural network has to be deployed in the real world, where the input data is slightly different from those seen during training. Therefore, it is important to carefully choose the training data. It should be helpful to obtain a general model, which has to perform well in any scenario. Modern deep models require large and heterogeneous datasets to reach their full potential, but gathering such data may be very expensive.

1.4 Datasets for autonomous driving

Driving datasets need to cover a lot of scenarios to provide effective training data. For many tasks, there have to be considered a lot of corner cases, a huge variation in content, weather and lighting conditions. Moreover, labels for computer vision tasks (*e.g.* semantic segmentation) can be very challenging to annotate. Annotation and

quality control may require 1.5 hours on average for a single image [2].

Even when the training dataset is huge and well annotated, it may still not be able to cover well all these cases. Moreover, each dataset is biased and unbalanced, which means that a neural network will overfit on some characteristics (*e.g.* camera parameters, sensor noise, resolution, etc.). In order to avoid the cost and human effort necessary to produce large general datasets that cover all cases needed, a lot of researchers are proposing methods that can simplify or even completely automate this process.

1.5 Simulators

Historically, many works have been conducted on synthetic data as proof-of-concept, when real data was not available. This limits the applicability of such works, but, nowadays, modern computer graphics keeps improving and becoming increasingly photorealistic. Therefore, researchers are trying to exploit simulated data to effectively train neural networks that will be deployed in the real world.

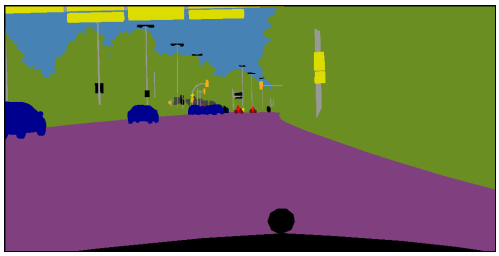
Simulators, in fact, provide an attractive solution to the labeling problem. Once the needed scenario has been setup, a simulator can be exploited to automatically generate the required ground-truth. Regardless of the target task, any kind of data from the rendering pipeline can be extracted from the engine and saved to create a training dataset. The only possible requirement is a user that interacts with the simulation (*e.g.* by driving through it while data is recorded).

1.6 Domain Shift

Despite modern computer graphics is capable of providing photorealistic renderings and simulations, they still lack many characteristics of real-world images. Lighting, colors and noise of the real world, in fact, are hard to imitate. Therefore, CNNs trained on synthetic images perform poorly when tested on real ones. This same effect happens when a CNN sees the same content in different conditions (*e.g.* a street in a sunny or a rainy day).

Figure 1.1: Screenshots taken from CARLA [3], an open-source simulator for autonomous driving.

(a) Real input image



(b) Ground truth

(c) Prediction

Figure 1.2: Example of domain shift. A semantic segmentation network trained on a synthetic dataset is unable to correctly process a real input image (a). This results in a poor prediction (c).

When circumstances or particular conditions change the distribution of content, we identify them as domains, *e.g.* sunny and rainy, summer and winter, or synthetic and real. The change between the distributions of two domains is called *domain shift*.

1.7 Unsupervised Domain adaptation

To cope with the issues that domain shift causes for neural networks, one has to solve a problem of *domain adaptation*. In our work, we consider the particular case of *unsupervised domain adaptation* (UDA), where no label is available for the domain used during testing. The recent works on UDA are detailed in Section 2.5. This work aims at developing perception methods that able to work in real environments without needing any labeled real image. We formally describe our problem in Section 3.1.

First we present a full system for UDA of Semantic Segmentation. Our system is a combination of pixel-level and feature-level alignment, which cooperate through bidirectional learning to improve each other. In particular we develop a Semantically Adaptive Image-to-image Translation method [4], which exploits the task network to adaptively align each class among the two input domains. This is used to translate synthetic images to the real domain and the result is then used to train the segmentation model (see Section 3.2). When training the segmentation model, we also perform feature-level alignment through adversarial learning and self-supervision (see Section 3.3). We extensively test our system on the common datasets and architectures to show how our results surpass the state-of-the-art of UDA for Semantic Segmentation in Section 4.3.

Then we extend our work to a more complex task, Panoptic Segmentation. We develop a method to separately adapt each subtask by training each branch in an adversarial fashion. In particular we align the features extracted by each branch of the task network through a different discriminator, making us able to also align the final panoptic output (see Sections 3.5 3.6 3.7). Finally, we experiment with this method on a synthetic-to-real setting of our choice and perform an ablation study to investigate the best solution for Panoptic Domain Adaptation in Section 4.4.

1.8 Contribution

This work improves the existing UDA approaches for Semantic Segmentation by strengthening the connection between pixel-level and feature-level alignment. To this end in [4] we introduce a novel Semantically Adaptive Image-to-image Translation method, which along with the semantic consistency induced by Symmetric Cross-Entropy (see Section 3.2.4) improves the power of translations. Such improvement is crucial to let the model improve when iteratively trained with Bidirectional Learning [76]. In Section 4.3 we highlight the importance of our contribution by comparing with the current state-of-the-art and performing an ablation study.

To the best of our knowledge there are still no works on UDA methods for the task of Panoptic Segmentation, which is getting a lot of interest since its first introduction in [42]. Therefore our contribution also includes the first attempt to adapt such a complex task across two domains. In particular we show how this can be achieved by carefully re-adapting existing methods for each panoptic subtask (*i.e.* semantic segmentation, object detection). In our experiments (see Section 4.4) we show how the source trained baseline can be improved by performing feature-level alignment at various levels.

Finally we suggest future directions for improving Panoptic Segmentation, like the adoption of a complete UDA pipeline (*e.g.* the one proposed for Semantic Segmentation) and the exploitation of its multiple tasks for a stronger Bidirectional Learning.

Chapter 2

Related Work

This chapter reviews the notions and methods that led to this work, starting from a brief overview of deep learning and including a review of the models currently employed to solve perception tasks, an explanation of Generative Adversarial Networks and the state-of-the-art of UDA.

2.1 Deep Neural Networks

Neural networks are function approximators. Given a function F that maps an input space X to an output space Y, the purpose of a neural network $\hat{F}$ is to approximate F. When fed with an input item $x \in X$, the neural network predicts an output $\hat{y} \in Y$. Training a neural network means optimizing its parameters to get an output $\hat{y} = \hat{F}(X)$ that is similar to the desired one $y = F(x)$. In fact, as introduced in Section 1.3, the most common approach is supervised learning, which requires a dataset of paired input-output items, *i.e.* pairs $(x,y)|x \in X, y \in Y, y = F(x)$.

In practice, a neural network is a sequence of simpler functions, called layers, which also map an input to an output. This functions are parameterized and the parameters, called weights w, are optimized during training. This optimization is usually based on gradient descent. The predicted output $\hat{y}$ is compared with the ground-truth y based on a loss function $\mathscr{L}(\hat{y},y)$, which is the objective to minimize. The

chain-rule is then used to compute the partial derivatives of the loss function *w.r.t.* each weight. This operation is called back-propagation, since each gradient is propagated from the output layer to the input ones. The objective is to find the global minimum of the loss function, but, actually, local minima are usually found. One of the main reasons behind the success of deep models *w.r.t.* shallow ones is that their local minima are more similar to the global one [5].

In the following we present some of the most used operations in deep neural networks and the advantages of using them.

2.1.1 Convolution

The discrete convolution is an operation that may have many parameters and, therefore, used in many ways. The parameters are the following:

- A filter, or kernel, with weights w that get *convolved* with the input to generate the output. A convolutional filter usually has 3 dimensions $C_{IN}\times H_K\times W_K$, where C_{IN} must be the same of the input feature map. A convolution with a 3-dimensional filter produces an output with 2 dimensions, which is why more filters are employed usually, stacking them in a 4-dimensional filter $C_{OUT}\times C_{IN}\times H_K\times W_K$, which produces an output with C_{OUT} channels.

- A bias b with the same length of the input channels C_{IN}

- A stride s, which controls the number of pixels for moving the kernel during the convolution, both in height and width. Stride is usually employed to downsample an input feature map.

- A padding p, which is useful to avoid losing border pixels from dimensions when performing convolution. When the filter is centered on the border of the input, in fact, the convolution would be invalid. A zero, replication, or reflection padding can be used to perform a valid convolution on each input pixel and its size depends on the kernel size.

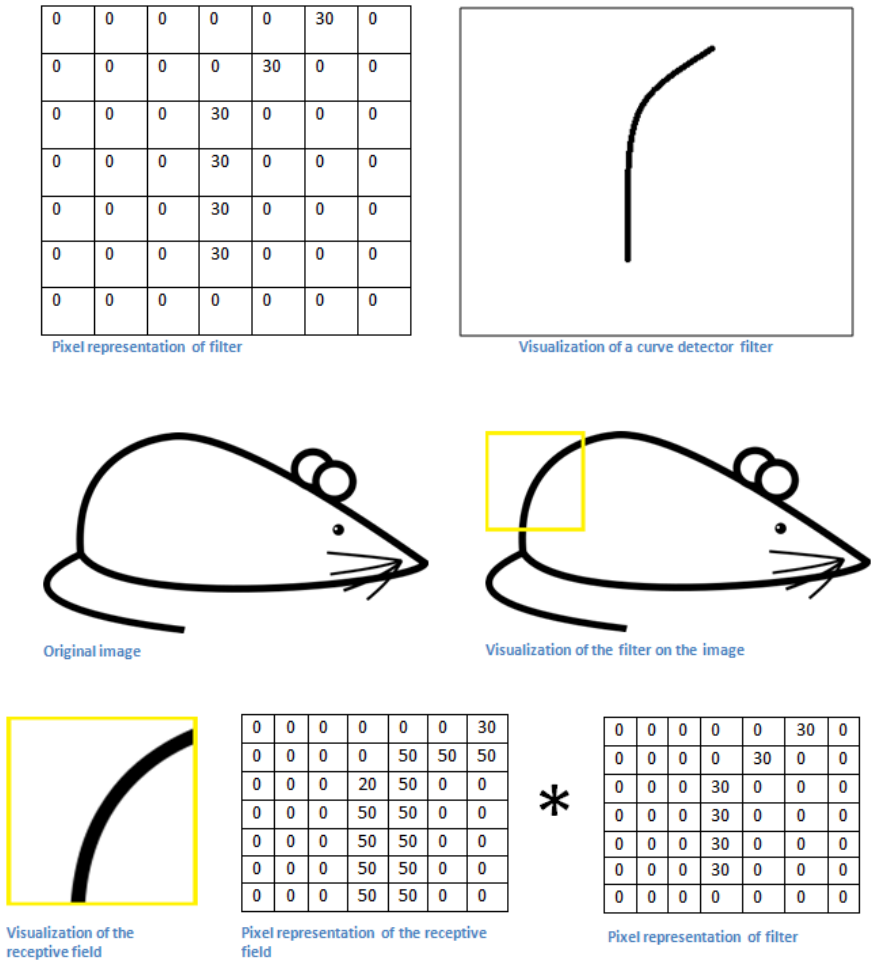
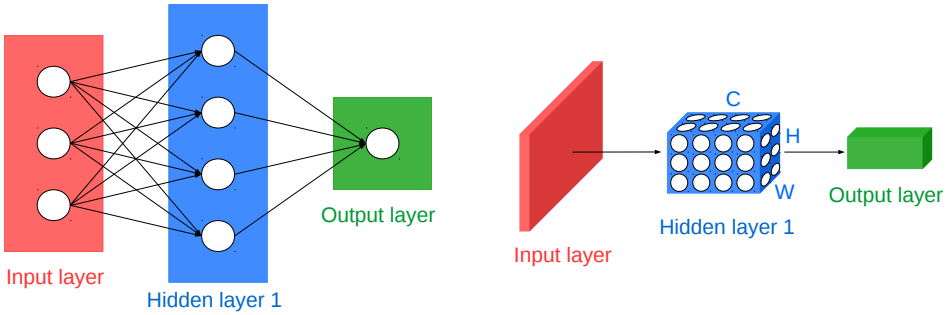


Figure 2.1: Example of how convolutional filters are used to search for recurrent patterns in images. Note how the filter pattern matches the one in the pixel representation of the receptive field.



(a) Example of fully connected neural network.

(b) Example of convolutional neural network.

Figure 2.2: Comparison between the hidden layers used in fully connected and convolutional neural networks. Figure(a) shows an example of fully connected neural network, where neurons are arranged in 1 dimension inside an hidden layer and there is a global connectivity. Figure(b) shows an example of convolutional neural network, where neurons are arrange in 3 dimensions inside an hidden layer and the connectivity is local for dimensions H and W.

- A dilation d, which enlarges the kernel by filling the holes with zeros. In other words, the kernel gets a bigger size, but skips some of the input pixels. Dilation is often used to take context into account when processing images [6, 7, 8, 9, 10, 11].

Given an input of shape $C_{IN}\times H_{IN}\times W_{IN}$, the convolution is computed as following:

$$\text{out}_j = b_j + \sum_{k=0}^{C_{IN}-1} w_{j,k} \star \text{input}_k, \forall j \in [0, C_{OUT}-1] \tag{2.1}$$

where $\star$ indicates the 2D cross-correlation, while the output shape is

$$H_{OUT} = \left\lfloor \frac{H_{IN} + 2\cdot p_H - d_H \cdot (H_K - 1) - 1}{s_H} + 1 \right\rfloor \tag{2.2}$$

$$W_{OUT} = \left\lfloor \frac{W_{IN} + 2\cdot p_W - d_W \cdot (W_K - 1) - 1}{s_W} + 1 \right\rfloor \tag{2.3}$$

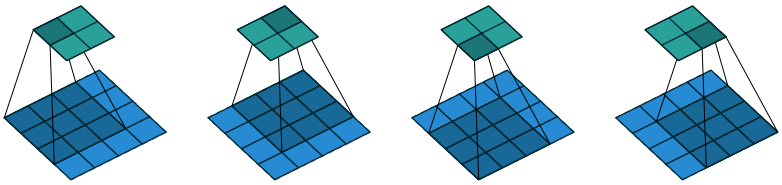


Figure 2.3: Example of convolution without padding, stride and dilation. The 3×3 kernel (in blue) slides on the input 4×4 matrix (in lighter blue), producing a 2×2 output (in green) [12].

where stride, padding and dilation have a pedix H or W depending on the case, since they can be different between height and width.

Convolutional Neural Networks (CNNs) employ convolutions in layers, and the elements of the kernel filters are the weights. As depicted in Figure 2.2, convolutional filters are arranged in 3 dimensions, differently from fully connected filters. Employing the convolution operation is particularly useful in computer vision, since learned kernels activate in correspondence of patterns that recur in images. When the searched pattern is found, the kernel activates, producing high-value outputs. A visual example of this process is presented in Figure 2.1. Stacking more and more layers helps searching for both low and hight level patterns at different scales. The final output is then used for prediction.

Convolutional layers may be followed by non-linear functions and arranged with different operations, depending on the architectural choices. The output layer also may differ based on the desired task.

2.1.2 Activation Functions

Convolutions alone do not guarantee the possibility to learn any function. The input-output relation produced by them is linear, while the desired output of our task may have a non-linear dependence from the input. Hence most architectures applies a non-linear activation function to each convolutional output. These functions make a neural network able to approximate any function, as proven in [13].

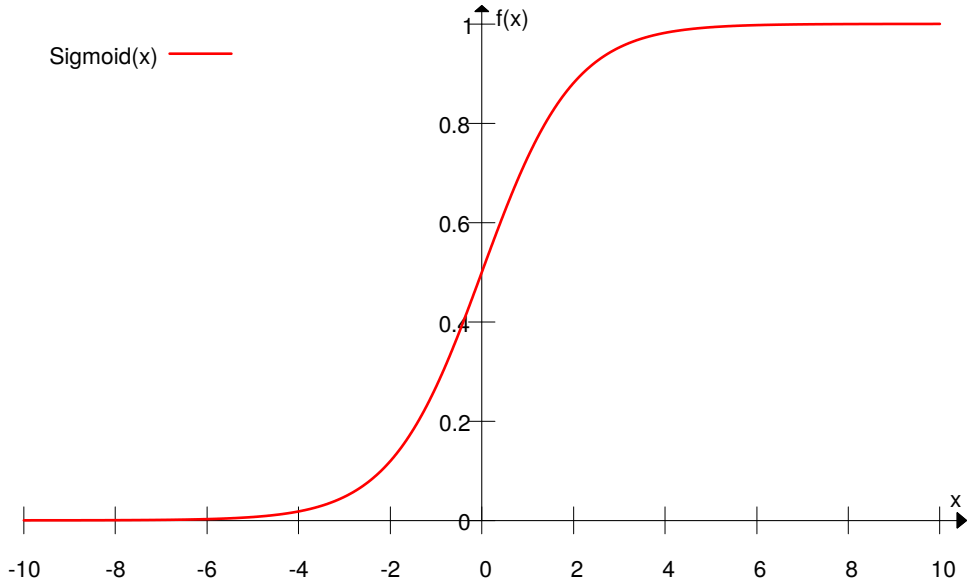


Figure 2.4: The Sigmoid function.

Sigmoid

Taking inspiration from biological neurons, the function adopted by early neural architectures has been the Sigmoid, shown in Figure 2.4 and represented by the following equation:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{2.4}$$

Still, it has the drawback of saturating its output in 0 or 1. These property is usually undesirable, since CNN learning is based on gradient descent, but the gradient of the output *w.r.t.* the input is zero in these regions, producing the *vanishing gradient problem*.

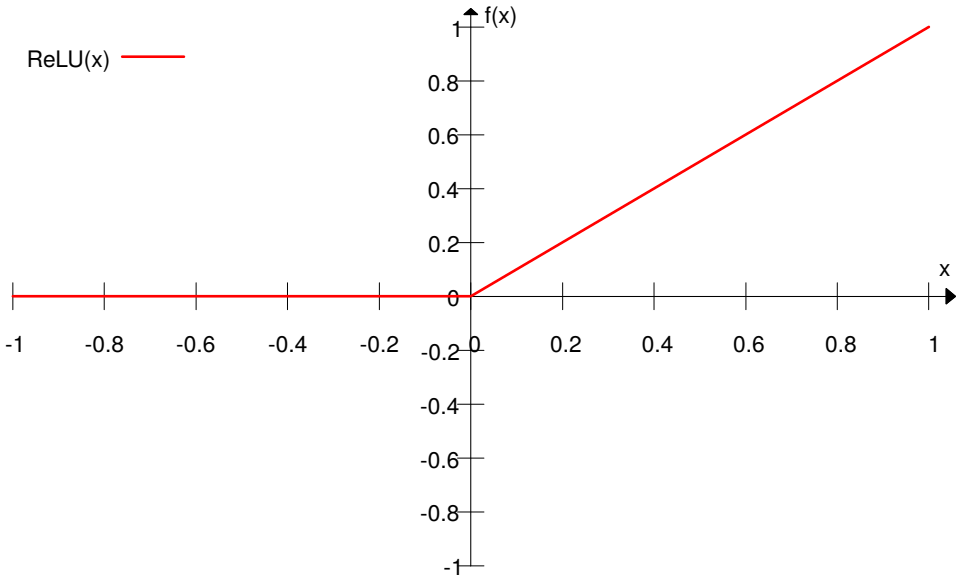


Figure 2.5: Rectified Linear Unit.

Rectified Linear Unit

The Rectified Linear Unit (ReLU) [14] is given by the following equation:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}. \tag{2.5}$$

As shown in Figure 2.5, ReLU allows the propagation of only positive signals, corresponding to the activation of neurons. Unlike the Sigmoid, the ReLU is not saturated, is easy to implement, has a very low computation cost and has brought great results on training convergence [15].

The main drawback of ReLU is that it can "die": when parameters are updated with an high magnitude gradient, they get modified in such a way that does not allow the neuron activation anymore; in other words, the neuron output remains zero for the rest of the training; the same happens to gradients, making them use for training impossible.

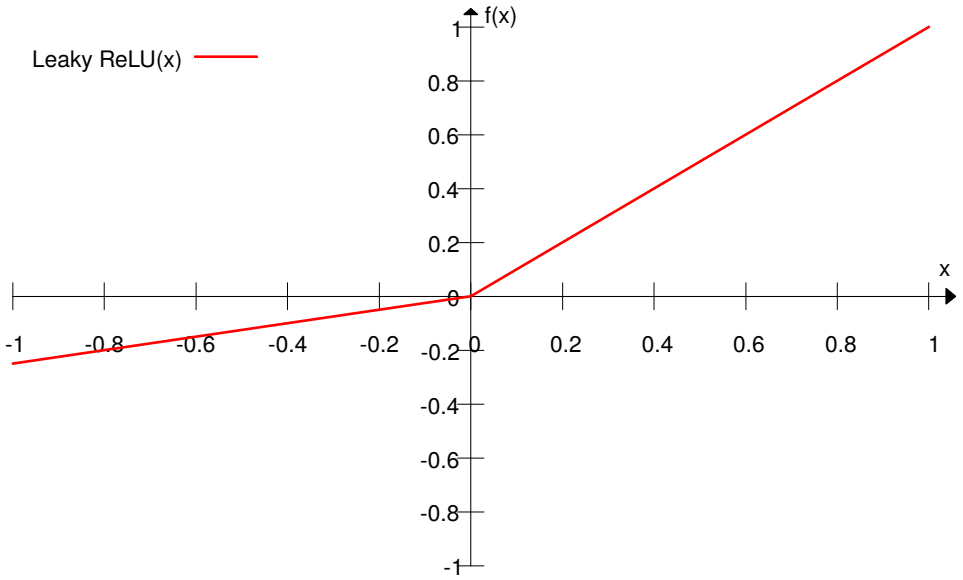


Figure 2.6: Leaky ReLU with $\alpha = 0.25$.

In order to solve this issue, a few variants of ReLU have been introduced.

Leaky ReLU and Parametric ReLU

Introduced in [16], Leaky ReLU is a variant of ReLU given by the following equation:

$$\text{LReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.6}$$

where $\alpha << 1$ is a factor that characterizes the slope of the negative part of the function, as show in Figure 2.6.

Parametric ReLU [17], instead, is an extension of Leaky ReLU where α is a parameter learned by the network itself.

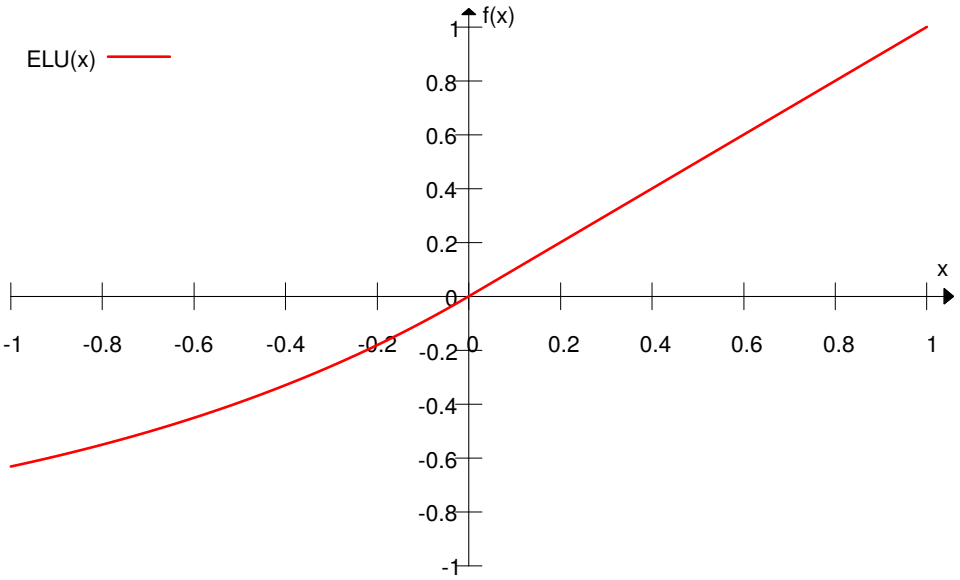


Figure 2.7: Exponential Linear Unit with $\alpha = 1$.

Exponential Linear Unit

The Exponential Linear Unit [18] is given by the following equation:

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.7}$$

ELU is an activation function that, as show in Figure 2.7, besides keeping the same properties of ReLU, has mean activations that are near 0, which may guarantee more stability in gradients. Moreover, negative inputs are saturated in $-\alpha$, providing more noise robustness.

2.1.3 Normalization layers

In this Section we present the normalization problem for Deep Neural Networks and how it can be solved using different normalization layers. In Figure 2.8 a comparison

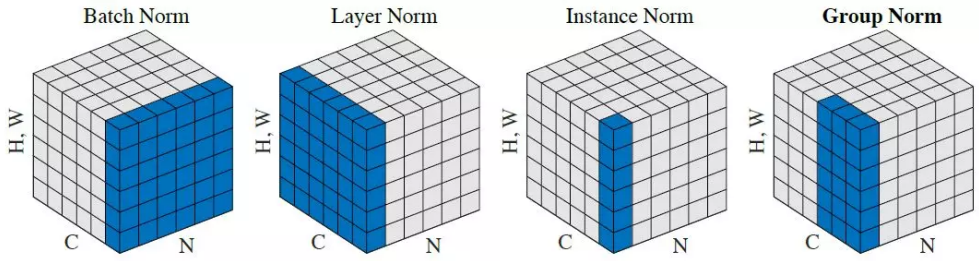


Figure 2.8: A comparison of the different Normalization Layers from [19].

of the different layers is shown. In Section 2.4.3 we also provide some insights on how they are used for image-to-image translation and, in particular, in our work.

Covariate shift

Preprocessing techniques for Deep Learning include the normalization of input data, since this ensures that the input layers always see a similar distribution. Still, because of the depth of modern architectures, this operation is not sufficient, since the intermediate operations may cause a divergence of the intermediate feature maps.

Covariate shift refers to the change in the input distribution to a learning system. In the case of deep networks, the input to each layer is affected by parameters in all the input layers. So even small changes to the network get amplified down the network. This leads to change in the input distribution to layers of the deep network and is known as *internal covariate shift*.

It is well established that networks converge faster if the inputs have been whitened (*i.e.* they have zero mean and unit variance) and are uncorrelated, but internal covariate shift leads to just the opposite. In order to solve this problem, Normalization layers are often applied after activation functions.

Batch Normalization

Given an input x of size $B \times C \times H \times W$, Batch Normalization [20] (BN) produces an output y as:

$$\begin{aligned} \mu_c &= \frac{1}{BHW} \sum_{b=1}^{B} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{b,c,h,w} \\ \sigma_c^2 &= \frac{1}{BHW} \sum_{b=1}^{B} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{b,c,h,w} - \mu_c)^2 \\ y_{c,h,w} &= \frac{x_{c,h,w} - \mu_c}{\sqrt{\sigma_c^2 + \varepsilon}} * \gamma_c + \beta_c \end{aligned} \tag{2.8}$$

where ε is a parameter that ensures numerical stability (*e.g.* $\varepsilon = 10^{-5}$). The mean μ_c and variance σ_c are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size C. These two parameters also provide the possibility to learn an identity transformation if it is the optimal one, obtaining $\gamma_c = \sqrt{\sigma_c^2 + \varepsilon}$ and $\beta_c = \mu_c$.

We know that normalizing the inputs to a network helps it learn. But a network is just a series of layers, where the output of one layer becomes the input to the next. That means we can think of any layer in a neural network as the first layer of a smaller subsequent network.

Thought as a series of neural networks feeding into each other, we normalize the output of one layer before applying the activation function, and then feed it into the following layer, or sub-network.

Since mean and variance are computed through the whole batch of input items, these will be more accurate if the batch size is large enough. Note that with $B = 1$, we will have $\sigma_c^2 = 0$, which does not allow BN to work.

Using this layer usually allows using an higher learning rate without training divergence and can lead to a better global precision.

Instance Normalization

Given an input x of size $B \times C \times H \times W$, Instance Normalization [21] (IN) produces an output y as:

$$\begin{aligned} \mu_{b,c} &= \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{b,c,h,w} \\ \sigma_{b,c}^2 &= \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{b,c,h,w} - \mu_b, c)^2 \\ y_{b,c,h,w} &= \frac{x_{b,c,h,w} - \mu_{b,c}}{\sqrt{\sigma_{b,c}^2 + \varepsilon}} \end{aligned} \tag{2.9}$$

The main difference with BN is that mean and variance are computed separately for each element of the mini-batch and for each feature map. This also means that each output channel of each element of the mini-batch is normalized separately. This leads to the possibility of using batches of size 1. Unlike BN, IN is applied at test time as well.

Layer Normalization

Given an input x of size $B \times C \times H \times W$, Layer Normalization [22] (LN) produces an output y as:

$$\begin{aligned} \mu_b &= \frac{1}{CHW} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{b,c,h,w} \\ \sigma_b^2 &= \frac{1}{CHW} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{b,c,h,w} - \mu_b, c)^2 \\ y_{b,h,w} &= \frac{x_{b,h,w} - \mu_b}{\sqrt{\sigma_b^2 + \varepsilon}} \end{aligned} \tag{2.10}$$

LN normalizes input across the features. This means that each element of the mini-batch is normalized separately using statistics computed on all the channels. LN can be considered a permutation of BN: while BN normalizes separately each $c \in C$ with statistics computed across B, LN normalizes separately each $b \in B$ with

statistics computed across C.

Group Normalization

Given an input x of size $B \times C \times H \times W$, C is divided into G groups C_i and the output activations y_{c_i} are computed as Group Normalization [19] (GN):

$$\begin{aligned} \mu_{b,c_i} &= \frac{1}{\frac{C}{G}HW} \sum_{c \in C_i} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{b,c_i,h,w} \\ \sigma^2_{b,c_i} &= \frac{1}{\frac{C}{G}HW} \sum_{c \in C_i} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{b,c,h,w} - \mu_b, c)^2 \\ y_{b,c,h,w} &= \frac{x_{b,c,h,w} - \mu_b}{\sqrt{\sigma^2_b + \varepsilon}} \end{aligned} \tag{2.11}$$

GN is particularly useful for alleviating the training problems with small batch sizes, since it acts as BN where instead of the groups of channels form a minibatch. When we put all the channels into a single group, GN becomes LN. When we put each channel into different groups, it becomes IN.

2.2 Perception tasks

Deep learning has been applied in many perception tasks. The following provides details about the ones that are most common and related to this work.

2.2.1 Image classification

One of the most basic tasks for perception is image classification: given an image, the objective is to find the class of the main subject (*e.g.* dog, cat or plane) among a fixed set of classes. There are many huge datasets available for this task because it is relatively easy to annotate images for it. The most important is probably ImageNet [23], which, for several years, served as a benchmark for CNN architectures [24].

In fact, the first work that successfully employed a CNN to boost performance in the ImageNet classification challenge was AlexNet [15] in 2012. In subsequent

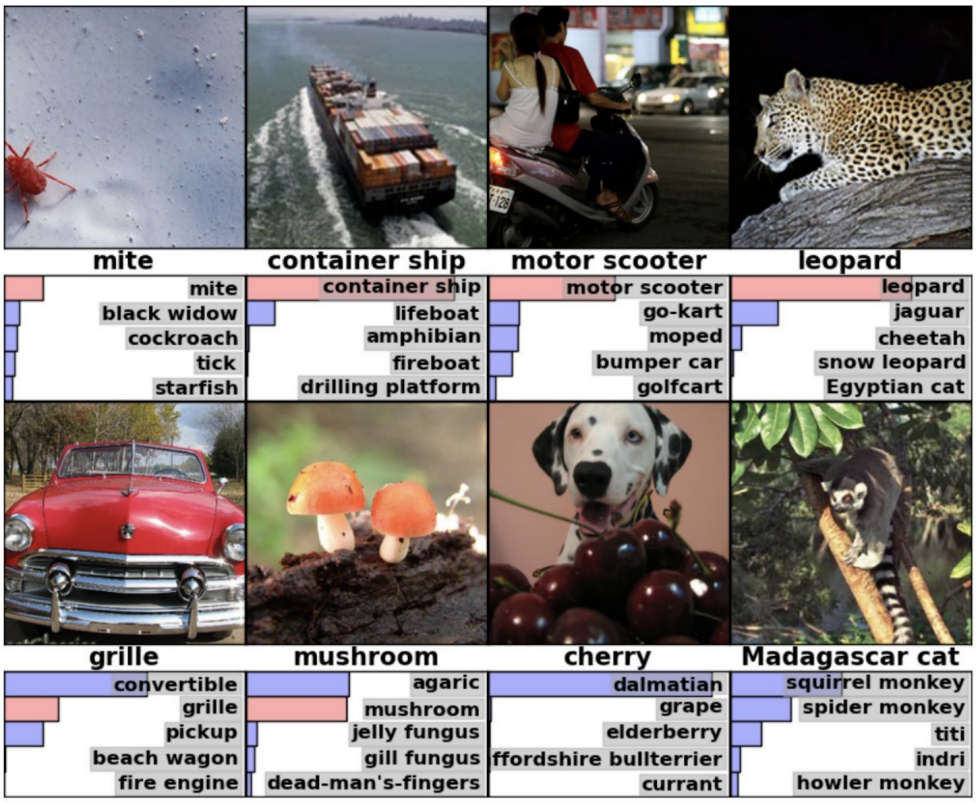


Figure 2.9: Example of an image classification output.

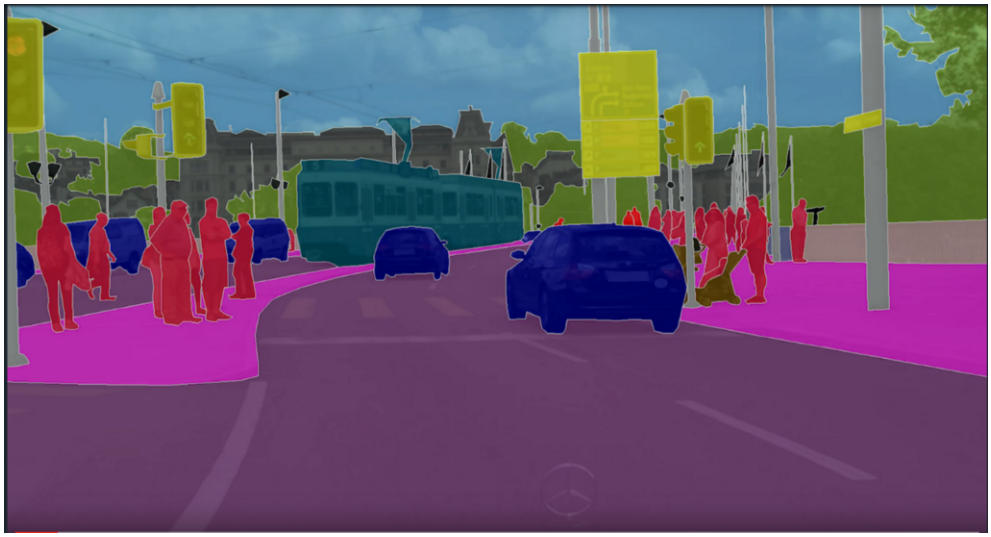


Figure 2.10: Example of semantic segmentation of an image. Each class is assigned a different color for visualization.

years, deeper and more complex architectures were designed to improve the accuracy of classification, like VGGNet [25], GoogLeNet [26] and ResNet [27]. Since their specialization as classifiers is mostly provided by the final layer, these models have also been readapted for other tasks. Some examples of this will be provided in the next sections.

2.2.2 Semantic segmentation

The objective of semantic segmentation (or pixel-level segmentation) is to find the class of each pixel. This is particularly interesting in street scenes, where many classes are present at the same time. However, finely annotating images for semantic segmentation can be very expensive and, likewise, obtaining high resolution results requires big models and lots of resources. The output semantic map is useful for a generic understanding of the environment that surrounds the car (*e.g. for road, lanes and curbs*), but it lacks the ability to distinguish among objects. In fact, there is no

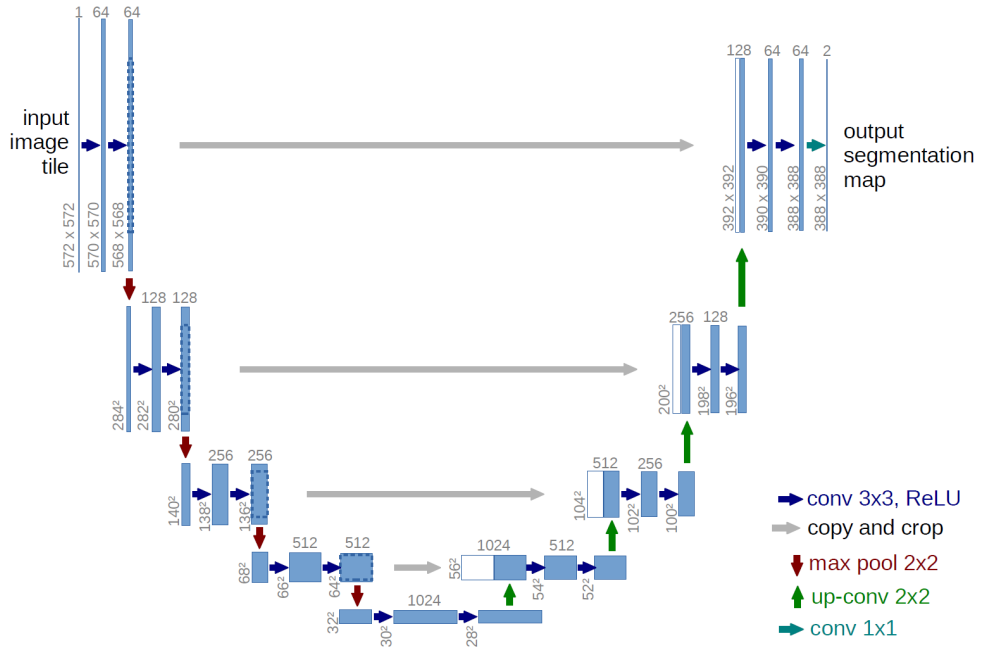


Figure 2.11: Architecture of U-Net [28].

intra-class boundary in a semantic map, which means that, from a cluster of pixels labeled as "car", it is hard to detect if there is only one car, or more.

Semantic segmentation models are usually built on top of classification architectures. The model chosen as backbone acts as feature extractor, or encoder. The encoded features are then decoded and upsampled to obtain a feature map with same spatial size as the input image. This is the main idea behind Fully Convolutional Networks [29], which combine the features extracted by different layers of the encoder with the ones of the corresponding scale in the decoder. Similarly, U-Net [28] (see Figure 2.11) decodes by combining upsampling and convolutions, where it also reuses features from the encoder through skip-connections.

The main challenge of semantic segmentation is how to exploit context to refine a prediction. Solutions like U-Net solve it by performing a high spatial compression

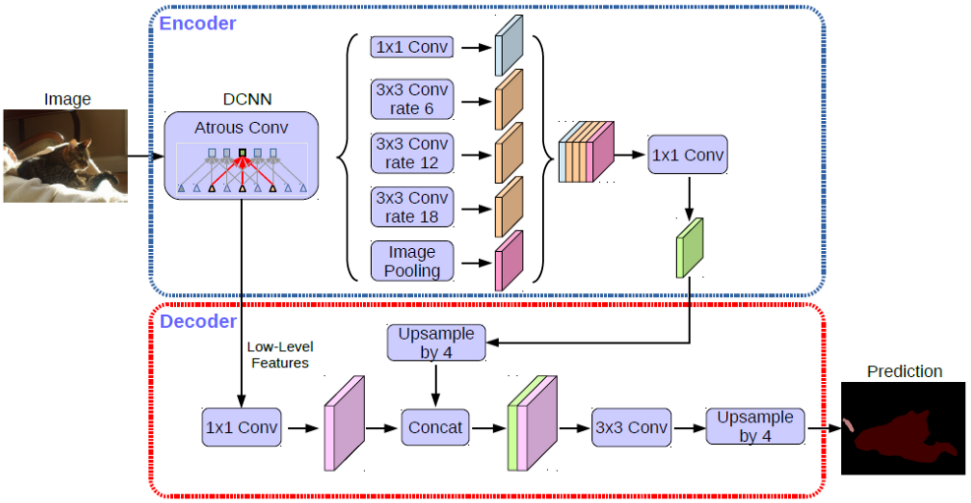


Figure 2.12: Architecture of DeepLabV3+ [11].

of the feature maps, which allows the deeper filters to have a big receptive field (*i.e.* the patch they convolve with corresponds to a large patch in the input image). Dilated convolutions, instead, expand the convolutional filter to cover a broader area of the input feature map. They have been highly exploited in Dilated Residual Networks [6, 7] and DeepLab [8, 9, 10, 11] (see Figure 2.12) to aggregate multi-scale contextual information and establish the current state-of-the-art of semantic segmentation.

2.2.3 Object detection

Object detection or localization asks to find a bounding box for each object in an image. The task may be restricted to a single class (*e.g.* a pedestrian detector), but modern benchmarks treat many classes, so usually it is also necessary to assign a class to each box. Clearly object detection is used only for those classes that have instances or can be counted, sometimes referred to as *things*, *e.g.* cars, bikes and pedestrian. This leaves out all the classes for which it would make no sense drawing a bounding box, like road, sidewalk and sky, sometimes referred to as *stuff*.

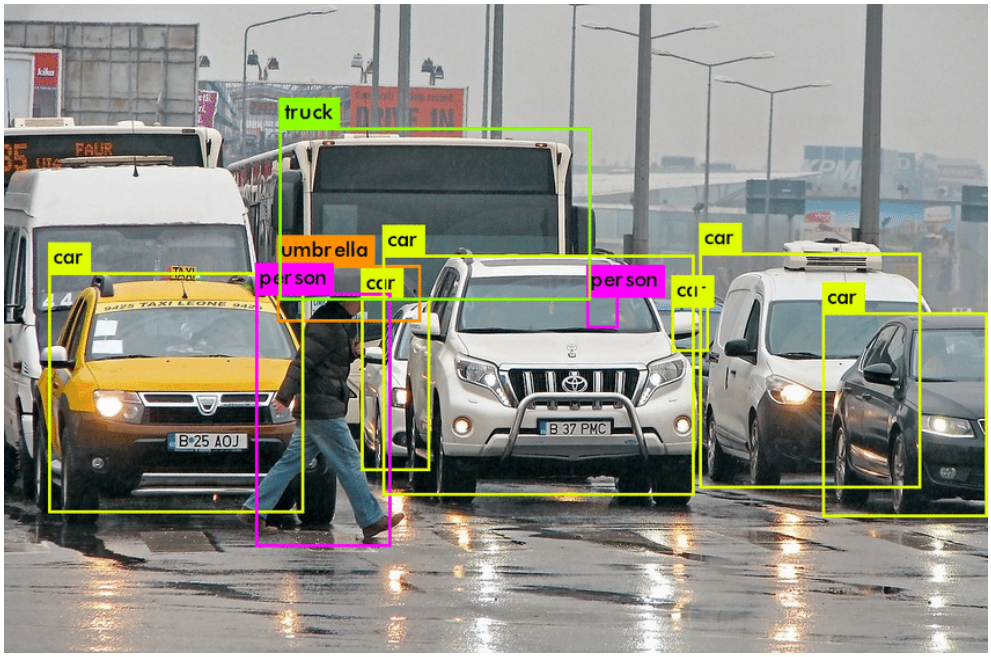


Figure 2.13: Example of object detection in an image. Each class is assigned a different color and label for visualization.

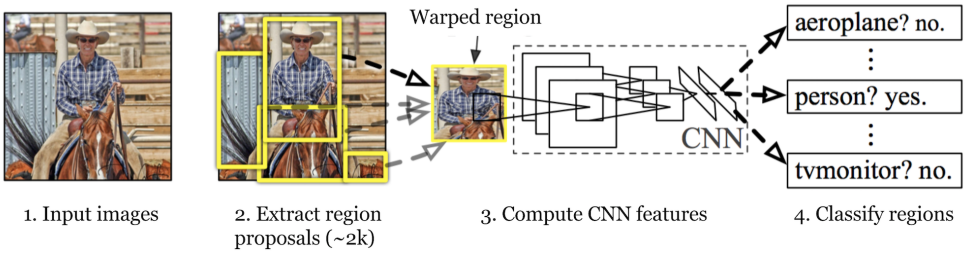


Figure 2.14: Basic scheme of a two-stage object detector [30].

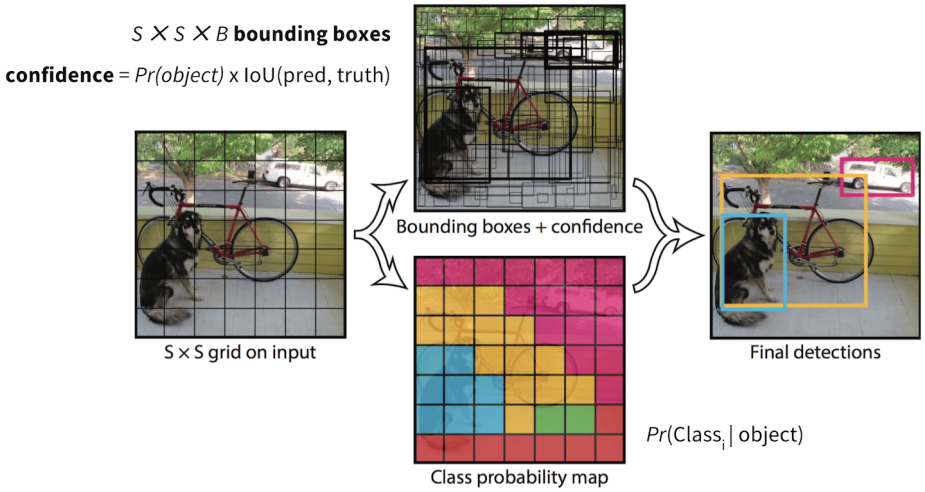


Figure 2.15: Scheme of a one-stage object detector [31].

Object detection models may be based on two possible approaches. The first one, slower but more accurate, is that of region-based object detectors. These models, mostly represented by the R-CNN family [30, 32, 33], operate in two stages: first, the model proposes a set of regions of interest; then, the region candidates are processed by a classifier. Figure 2.14 provides a scheme of a two-stage detector. The second approach, faster but possibly less accurate, skips the region proposal stage and runs detection directly over a dense sampling of possible locations. One-stage object detectors are an attempt to solve the task in real-time, leaving out possible performance drops. The most famous ones include the YOLO family [31, 34, 35] (see Figure 2.15), SSD [36] and RetinaNet [37].

2.2.4 Instance segmentation

Instance segmentation asks for an output that is more precise than object detection. It treats the same classes, but each object should be separately segmented by finding its mask. Usually this is considered an extension of detection, since it is much easier to

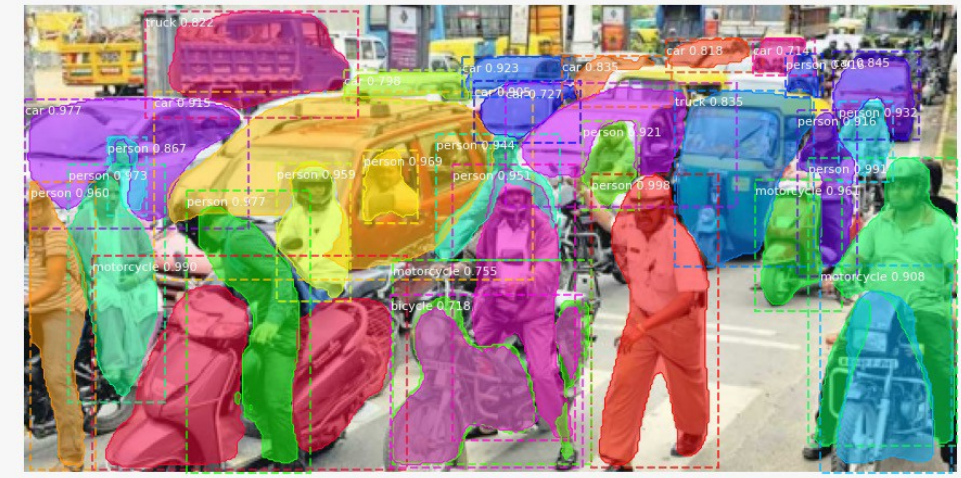


Figure 2.16: Example of instance segmentation of an image. Each class is assigned a different color for visualization.

first detect all objects, and then segment a mask inside each bounding box.

In fact, one of the first works on CNN-based instance segmentation is Mask R-CNN [38]: it extends Faster R-CNN [33], one of the most employed two-stage object detectors, by adding a segmentation head to it. Faster R-CNN uses the features inside the proposed regions to predict the box class and correct coordinates. As depicted in Figure 2.17, these same features are used in Mask R-CNN to segment exactly the object inside the bounding box.

In Mask R-CNN, the classification score is used as mask quality score (*i.e.* confidence for the predicted mask). However, it is possible that due to certain factors, (*e.g.* background clutter or occlusions) the classification score is high, but the mask quality is low. Mask Scoring R-CNN [39] uses a network that learns the quality of mask. The mask score is reevaluated by multiplying the predicted mask quality and classification score. One of the main drawbacks of models based on R-CNN is their computational complexity: being based on a two-stage detector, they usually struggle to achieve real-time performance.

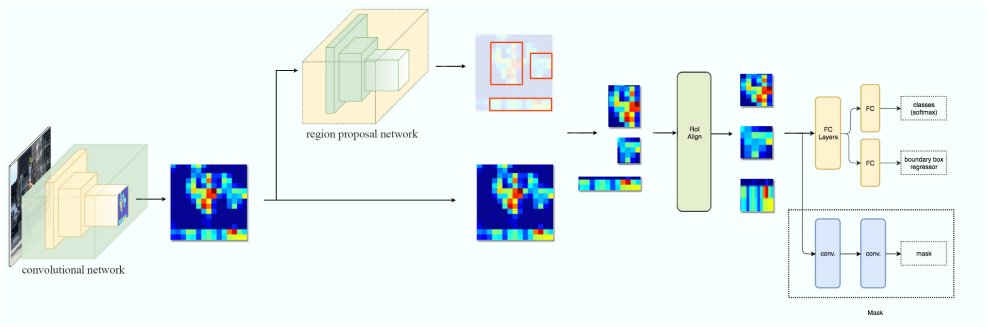


Figure 2.17: Scheme of Mask R-CNN [38].

Yolact [40, 41] tries to solve this by redesigning the model and splitting instance segmentation into two subtasks: generation of prototype masks and per-instance mask coefficients predictions. The results are then linearly combined to produce the per-instance masks and it is able to run in real-time.

2.2.5 Panoptic segmentation

Semantic segmentation is useful to understand the surrounding environment, but instance segmentation is also important to precisely distinguish each object. Since they are both needed to thoroughly perceive the environment, recently they have been merged into a single task, panoptic segmentation [42].

Classes are split into *things* and *stuff* classes. Things are countable objects and, for each one, it is necessary to identify its mask (*i.e.* the pixels it fills), its class, and an identifier, used to distinguish it from the other instances of its class. Stuff, instead, are classes described by amorphous regions of the same texture or material, which get treated as in semantic segmentation. Panoptic segmentation requires to correctly classify each pixel in the image. For pixels belonging to things, it is also necessary to correctly distinguish each instance.

One of the ground-breaking works of panoptic segmentation is Panoptic FPN [43]. As depicted by Figure 2.19, it extends Mask R-CNN by adding a global segmentation branch. This branch is fed with the global features extracted from the backbone,

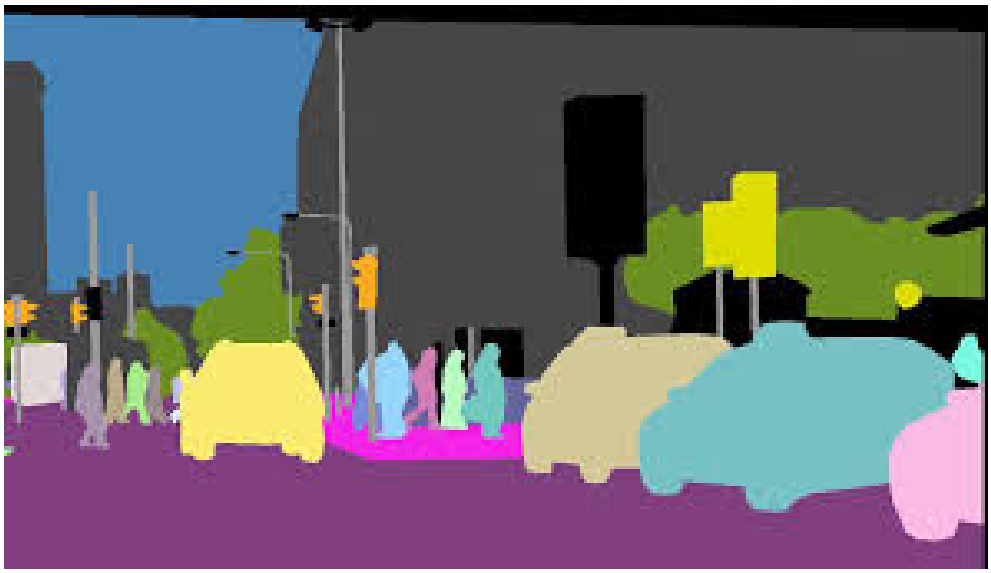


Figure 2.18: Example of panoptic segmentation of an image. Each class and instance is assigned a different color for visualization.

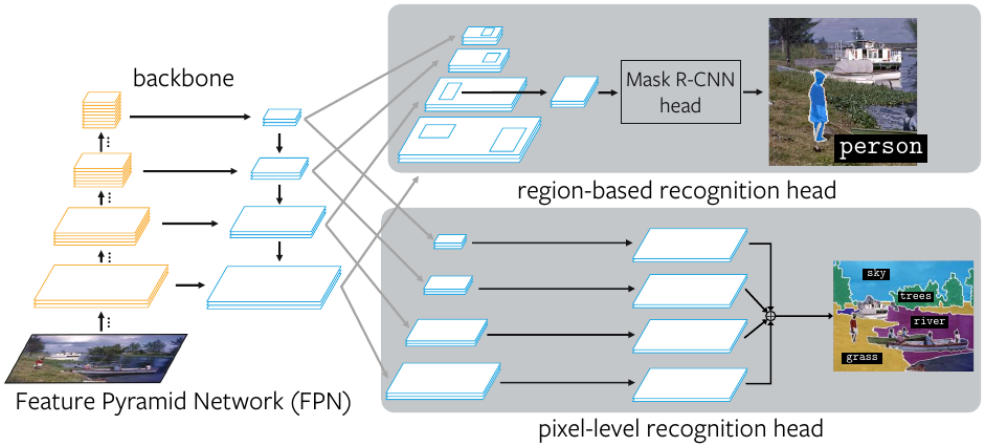


Figure 2.19: Scheme of Panoptic FPN [43].

which are the same that, in parallel, are used to detect instances. This is possible thanks to the Feature Pyramid Network [44] module, which has been shown to encode very well features from multiple scales, enabling many applications that require context understanding, like segmentation ones. Similarly, Seamless Scene Segmentation [45] extends Mask R-CNN with a light-weight DeepLab-like module used to obtain the semantic segmentation of the stuff classes.

Panoptic-DeepLab, instead, extends DeepLab with an instance segmentation branch that is different from Mask R-CNN. As can be seen in Figure 2.20, this produces two outputs: a center-of-mass prediction, for each instance, and instance center regression, where the model learns to regress each instance pixel to its center of mass.

2.3 Generative Adversarial Networks

A possible high-level interpretation of Deep Neural Networks is that they extract information from data (*e.g.* features from images) and encode them in a latent space. This information is then used to perform inferences depending on the task at hand. An interesting question is whether it is possible to invert this process and use a neural network to generate data from a random code, obtaining a Generative Model. There are many variants of Generative Models, depending on whether the probability density function that describes the data is explicitly or implicitly defined.

The most successful approach is an implicit class of generative models: Generative Adversarial Networks (GANs) [46]. The idea behind them is that of a minimax game where two neural networks compete against each other. The first one is a generator G and is in charge of imitating the real data distribution (*e.g.* by generating images). The second one is a discriminator D and has to distinguish between real data and generated one (*i.e.* produced by the generator). By learning how to fool the discriminator, the output produced by the generator becomes more and more realistic. As depicted in Figure 2.21, GANs are trained by alternatively updating the generator and the discriminator. First, the generator is fed with a random input noise z Z, which usually is sampled from a normal distribution. This is used to create a fake data sample $G(z)$, which the discriminator tries to classify as real or fake. The discriminator

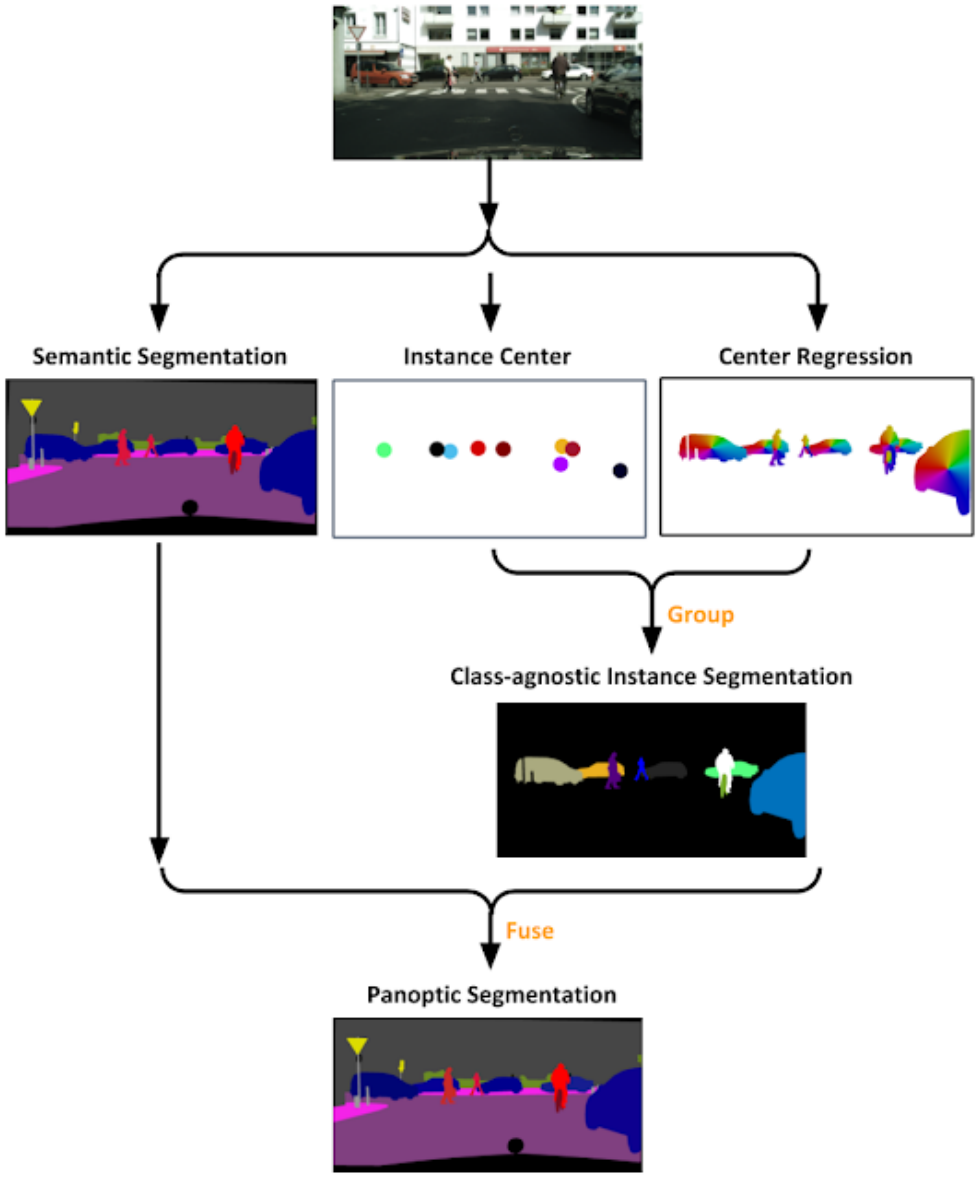


Figure 2.20: Scheme of Panoptic-DeepLab [43].

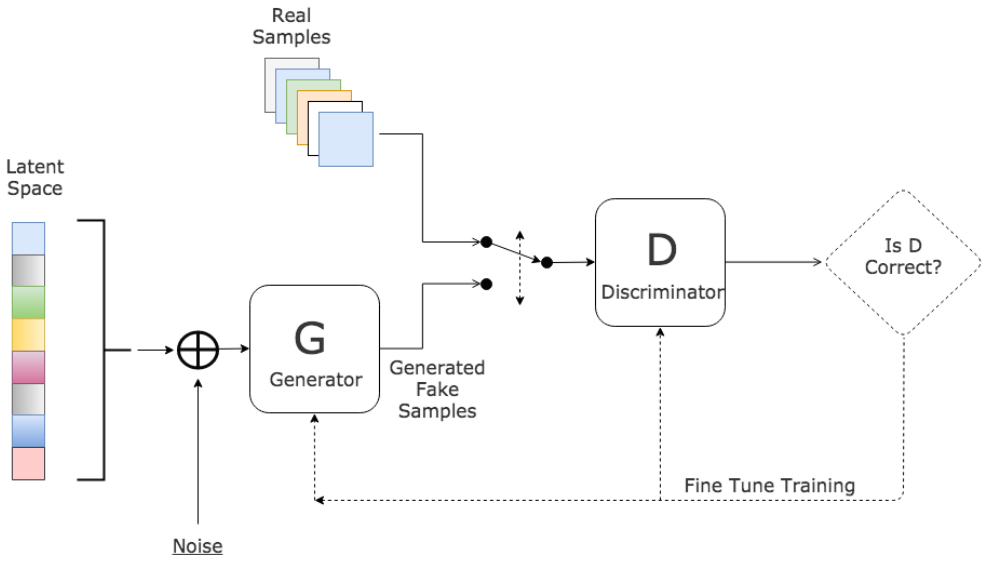


Figure 2.21: Training scheme for GANs [46].

also tries to classify a real data sample x X, which makes it able to learn the real data distribution. The objective function of the GAN framework is then formulated as:

$$\min_G \max_D \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))] \tag{2.12}$$

This formulation leads to the minimization of the Jensen-Shannon divergence but has been modified in many ways. The original paper itself notes that, in practice, Equation 2.12 may not provide sufficient gradient for G to learn well, especially in the early training stages. Because of this, instead of minimizing $\log(1 - D(G(z)))$, the generator is trained to maximize $\log(D(G(z)))$.

Wasserstein GANs (WGANs) [47] employ the earth mover distance between the real and generated distributions:

$$\max_D \mathbb{E}_x[D(x)] - \mathbb{E}_z[D(G(z)))] \tag{2.13}$$

$$\min_G \mathbb{E}_z[D(G(z))] \tag{2.14}$$

Here D is no longer a discriminator, but is called *critic*, since its output is no longer a probability, but may assume any value. In practice it tries to maximize the distance between its output on real samples and its output on fake ones. This makes the training less vulnerable to getting stuck and avoids problems with vanishing gradients.

Least Squares GANs (LSGANs) [48], instead, adopt the least squares loss function:

$$\min_D \frac{1}{2}\mathbb{E}_x[D(x)^2] + \frac{1}{2}\mathbb{E}_z[(1 - D(G(z)))^2] \tag{2.15}$$

$$\min_G \frac{1}{2}\mathbb{E}_z[(D(G(z)))^2] \tag{2.16}$$

This leads to the minimization of the Pearson χ^2 divergence. LSGANs are able to provide higher quality images than regular GANs and are more stable during the training process.

Conditional GANs (CGANs) [49] introduced the possibility to condition the generated output. In particular, the input noise is concatenated with a one-hot vector that encodes the desired output class and allows the generator to create a corresponding image. Similarly, the one-hot vector is concatenated with the generated image when presented to the discriminator, in order to balance the adversarial training. Figure 2.22 also depicts how Conditional GANs are trained.

Deep Convolutional GAN (DCGAN) [50] is the first architecture that has been able to generate realistic images thanks to a fully convolutional approach. First, it projects and reshapes the latent code to obtain a 3D tensor. Then it employs fractionally strided convolutions (sometimes also called deconvolutions) to upsample this feature map and decode it into an image.

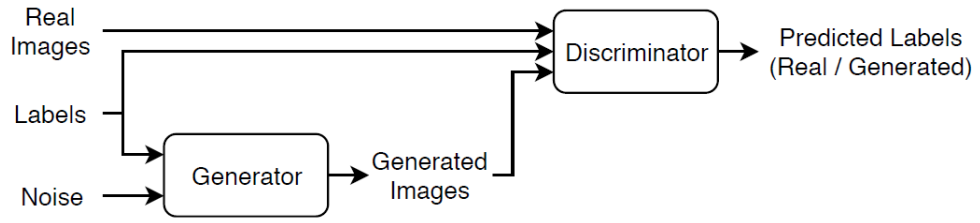


Figure 2.22: Scheme of a Conditional GAN [49].

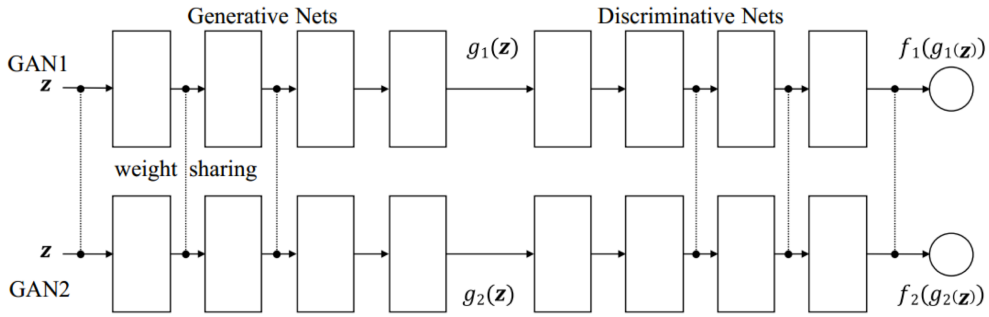


Figure 2.23: Scheme of two Coupled GANs [51].

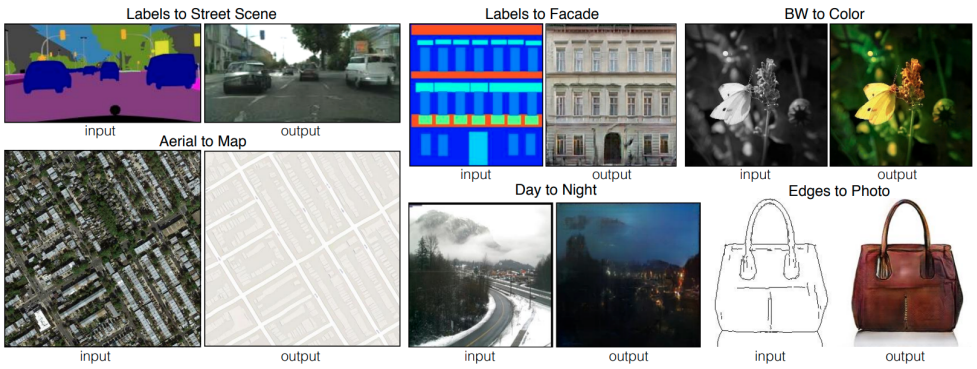


Figure 2.24: Example applications of image-to-image translation in Pix2Pix [52].

Coupled GANs (CoGANs) [51], instead, use a method to learn joint distributions from samples of the marginals. Two generators are coupled and trained to trick two discriminators, also coupled. As depicted in Figure 2.23, the coupling is achieved through a partial weight sharing and allows to get the same high level representation for the two distributions or, in other words, for two domains. The main advantage of CoGANs is the possibility to generate data for K domains by requiring less than K GANs, thanks to the shared parameters.

2.4 Image-to-image Translation

Besides creating images from scratch, sometimes may come in handy to generate them based on another input image. The works in this area consider many cases for this, *e.g.* label-to-photo, aerial photo-to-map or day-to-night.

2.4.1 Paired methods

One of the most successful methods for translation has been presented in Pix2Pix [52], which employs Conditional GANs [49] to condition the generated outputs based on the input label or photo. Pix2PixHD [53] extends this work for semantic image synthesis by designing an architecture and a loss function that makes it able to generate

high-resolution photorealistic images. Moreover, SPADE [54] further improves the quality of semantic image synthesis by designing a new spatially-adaptive normalization layer. This allows for a more precise semantic manipulation when synthesizing the output image and achieves one of the best results so far in this area.

2.4.2 Unpaired methods

Still, all these methods require a dataset composed of paired images: semantic image synthesis, for example, requires an image and its semantic map to learn the mapping between the two domains. In some cases this is unfeasible, in particular when none of the two domains is artificial, textite.g. horses-to-zebras, snow-to-summer or photo-to-painting. This kind of problem requires unpaired (*i.e.* unsupervised) solutions, where the generative model has to learn the joint distribution from the marginals. This family of translation problems is sometimes blended with *style transfer*, which is used to transfer the style of one domain, called target, into another one, called source. In this case, *style* refers to the bias that is particular of one domain (*i.e.* its distribution) and is common to all the data that identifies it. Therefore, instead of creating a new image from random noise, style transfer aims at changing the distribution of the input image to make it look like an image in the target domain.

One of the main challenges of image-to-image translation is how to change the style of the input image without modifying its content. A trivial solution would be to supervise the translation, *i.e.* to provide paired images with the same content but different styles. Still, this is unfeasible in many of the cases of interest or would require too much human effort. Therefore the focus of research in this area has been towards unpaired (*i.e.* unsupervised) solutions.

The most employed solution to content preservation has been provided by CycleGAN [55], which imposes cycle consistency in image translation. As depicted by Figure 2.25, the model contains two mapping functions, $G : X \rightarrow Y$ and $F : Y \rightarrow X$, that translate images from the domain X to the domain Y and vice versa. The mapping functions also have associated discriminators D_Y and D_X. Cycle consistency is

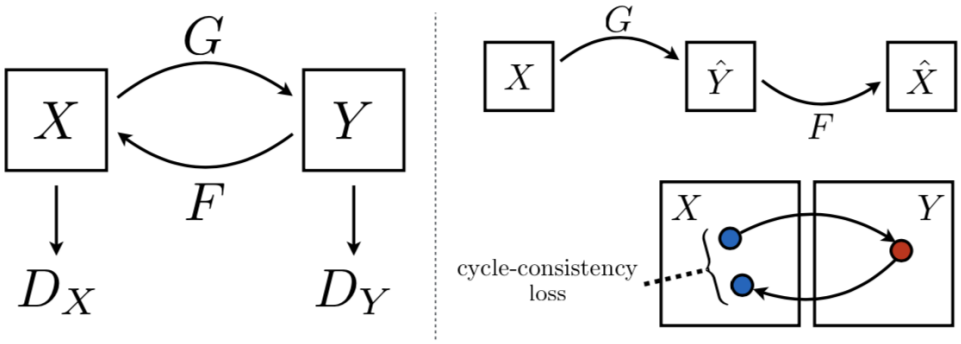


Figure 2.25: Scheme of CycleGAN [55].

achieved by minimizing the following loss function:

$$\mathscr{L}_{cyc}(G,F) = \mathbb{E}_{x\sim X}[||F(G(x)) - x||] + \mathbb{E}_{y\sim Y}[||G(F(y)) - y||] \tag{2.17}$$

This requires that each mapping can be inverted and successfully allows the disentanglement of content and style. Thanks to adversarial training (*i.e.* the discriminator of each domain tries to classify the given image as real or fake) this approach is completely unsupervised and no paired couple of images is required.

Similarly, UNIT [56] tackles image translation with cycle consistency, but also adds the hypothesis that the two domains share a common latent space which can be exploited in neural style transfer to preserve content. Therefore each mapping function is split in encoder and decoder, where the encoder extracts the latent code from an image, while the decoder translates it into an image of the target domain. The shared latent space assumption, depicted in Figure 2.26, is inspired by CoGAN [51] and enforced by sharing some weights across the domain mappers (*i.e.* the encoders for the two domains share the weights of the last layers, and the decoders share the weights of the first ones).

The framework of UNIT has been extended in MUNIT [57] for multimodal translation. In particular, MUNIT formalizes the distinction between style and content. As illustrated in Figure 2.27, the shared latent space is now formalized as content, while

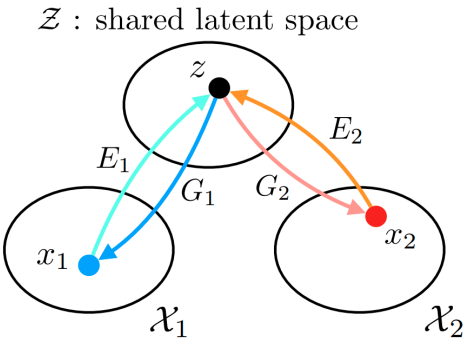


Figure 2.26: Shared latent space assumption in UNIT [56]. The encoders E_1 and E_2 are coupled with the generators G_1 and G_2 to translate images from the domain X_1 to the domain X_2 and vice versa. Therefore, the items x_1 and x_2 from the two domains can be mapped to the same latent space Z.

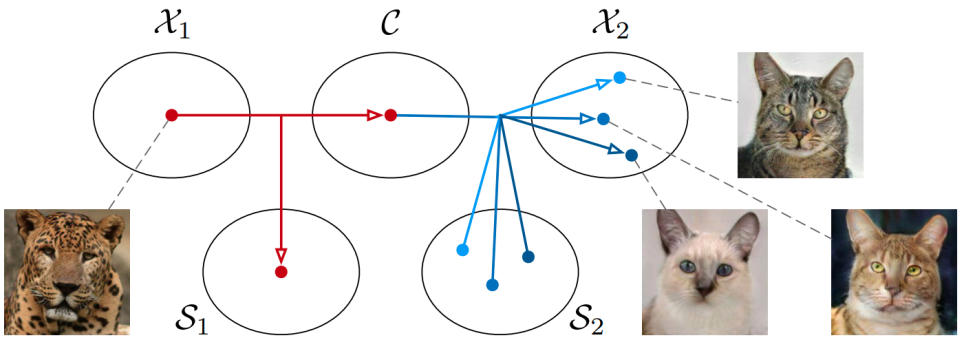


Figure 2.27: Illustration of the method used for translation in MUNIT [57]. The source domain X_1 is separated into style S_1 and content C. Then, the style S_2 is sampled and used to generate an image of the target domain X_2 from the content C. Multiple modes of S_2 can be sampled to obtain different images. The same flow can be replicated when translating X_2 into X_1.

style is a code to be sampled and used to generate images with multiple modes in the target domain.

2.4.3 Normalization layers

The key insight for image-to-image translation is in the ability to disentangle style and content. In fact, in order to move from one domain to the other, one has to be able to change the style while preserving the image content.

It has been noted [58] that the most effective way to swap styles is by using normalization layers. Batch Normalization [20] has been used in [59], but [60] found that replacing it with Instance Normalization (IN) [21] leads to significant improvements. IN works in the feature space in the same way Contrast Normalization works in the pixel space, which makes it much more effective. A more general approach has been introduced by Adaptive IN (AdaIN), which computes the affine transformation from a style input. UNIT [56] uses IN to swap the source and target style. Instead of performing a global translation with IN, we exploit the task network to translate each region of the image according to its semantic meaning. To this end, in Section 3.2.5 we choose to denormalize the generator activations with the SPADE layer [54], giving a result that naturally cooperates with the learning of the semantic segmentation task.

2.5 Unsupervised Domain Adaptation

As introduced in Section 1.7, this work aims at training deep CNNs on synthetic images. These networks, however, have to process real images, but there is a domain shift caused by the differences between real and simulated worlds. This can be framed as a problem of Unsupervised Domain Adaptation (UDA), where the main idea is to align the distribution of the synthetic and real domain at either feature level, pixel level, or both.

2.5.1 Image classification

The first task where UDA has been employed is image classification [61, 62, 63, 64, 65, 66, 67]. The first approaches are based on the minimization of the Maximum Mean Discrepancy (MMD) [68, 63]. MMD is a measure of distance between the mean embeddings of features. In other words, these systems try to align the two domains by minimizing the distance between the features extracted in them:

$$MMD(X_1, X_2) = ||\mathbb{E}_{x_1 \sim X_1}[\phi(x_1)] - \mathbb{E}_{x_2 \sim X_2}[\phi(x_2)]|| \tag{2.18}$$

where X_1 and X_2 are the two domains and ϕ is the embedding function that maps an input item x to the feature space.

Deep CORAL [69], instead, uses the correlation distance between the extracted features. In particular, given two batches of n_{X_1} and n_{X_2} data from the two domains, Deep CORAL first computes the covariance matrices as

$$C_{X_1} = \frac{1}{n_{X_1} - 1}(\phi(X_1)^T \phi(X_1) - \frac{1}{n_{X_1}}(\mathbf{1}^T \phi(X_1))^T (\mathbf{1}^T \phi(X_1)) \tag{2.19}$$

$$C_{X_2} = \frac{1}{n_{X_2} - 1}(\phi(X_2)^T \phi(X_2) - \frac{1}{n_{X_2}}(\mathbf{1}^T \phi(X_2))^T (\mathbf{1}^T \phi(X_2)) \tag{2.20}$$

where $\mathbf{1}$ is a column vector with all elements equal to 1. Then, the CORAL loss is defined as the norm of the difference between the covariance matrices:

$$\mathscr{L}_{CORAL} = \frac{1}{4d^2}||C_{X_1} - C_{X_2}||_F^2 \tag{2.21}$$

where d is the dimension of the activation layer mapped by ϕ and $||\dot{|}||_F^2$ is the squared matrix Frobenius norm.

Rather than introducing constraints on feature distributions, works such as [61, 64] have shown that it is best to let an adversary network force them. Thanks to the introduction of the Gradient Reversal Layer (GRL), these works have added a small domain classifier to the end of the classification model. As depticted by 2.28, GRL is a simple inversion gate that negates the flowing gradients. This has the effect of up-

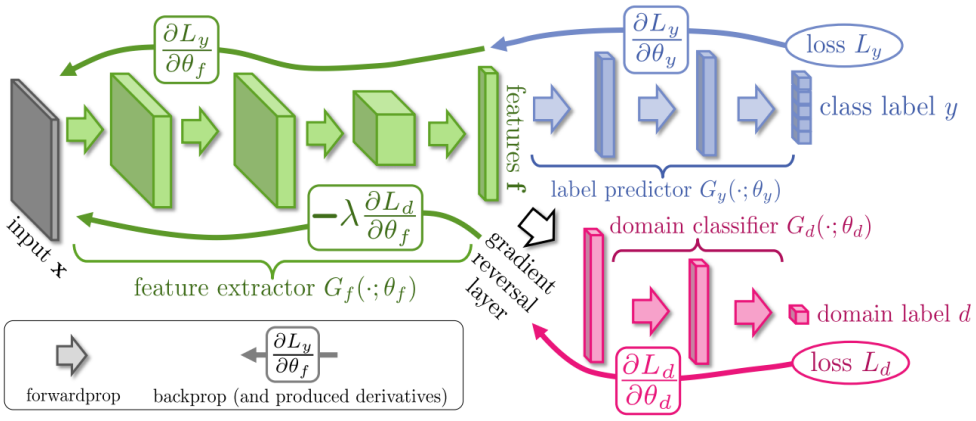


Figure 2.28: Architecture proposed by [61]. The Gradient Reversal Layer inverts the gradient flowing during backpropagation, allowing the domain classifier to update its weights in order to fool the classification model.

dating the base model and the domain classifier in opposite directions, making them play an adversary game similar to GANs [46] (see Section 2.3), but with simultaneous updates and identical loss formulation (except for the negative factor).

ADDA [66] is one of the first approaches that employ adversarial training for domain adaptation. The task network (*i.e.* the classification network in this case) is trained to extract features that trick a discriminator when asked which domain they belong to. In particular, the same setting of Equation 2.12 is used where instead of real or fake images the discriminator is asked to distinguish among features from the two domains:

$$\min_D -\mathbb{E}_{x_1\sim X_1}[\log D(\phi(x_1))] - \mathbb{E}_{x_2\sim X_2}[\log(1-D(\phi(x_2)))] \tag{2.22}$$

$$\min_G -\mathbb{E}_{x_2\sim X_2}[\log D(\phi(x_2))] \tag{2.23}$$

As illustrated by Figure 2.29, in this case the task network acts as generator and the discriminator takes the role of domain classifier.

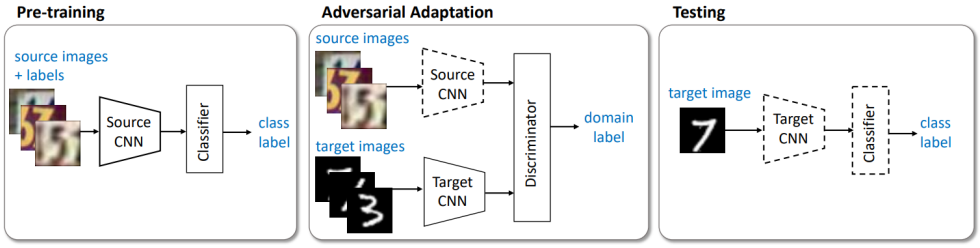


Figure 2.29: Illustration of the different phases of ADDA [66]. The task network (*i.e.* the classification CNN) is different for the source and target domain. In order to perform adaptation, the target CNN is trained in an adversarial fashion to trick the discriminator when extracting features from the target images.

2.5.2 Semantic segmentation

Semantic segmentation is a much more complex task than classification and, therefore, its adaptation requires more advanced methods.

One of the first works in this area has been proposed in [70], where dilated FCN [6] has been adopted as baseline. Here the output feature map acts as input of a domain classifier, which is update alternatively in an adversarial fashion. Curriculum learning was used in Curriculum Domain Adaptation (CDA) [71], where the authors proposed first to learn the global distribution of the image and the local distribution of superpixels, and then train the segmentation network according to these properties. Global feature alignment was also adopted in [72], where different discriminators are employed for features at different levels.

CLAN [74] improves feature level alignment by reweighting the adversarial loss with a discrepancy map based on categories, while other works introduced methods for class-wise adversarial learning [73] and pseudo-labels [73, 75, 76]. Pseudo-labels, in particular, are a powerful method employed for self-supervised or weakly-supervised learning. Given a pretrained network, its predictions are used as pseudo-labels, meaning that strong and confident predictions can be adopted as ground-truth according to some thresholded heuristic. The threshold may be either manually chosen (*e.g.* usually 0.9) or computed based on the global distribution of confidence on

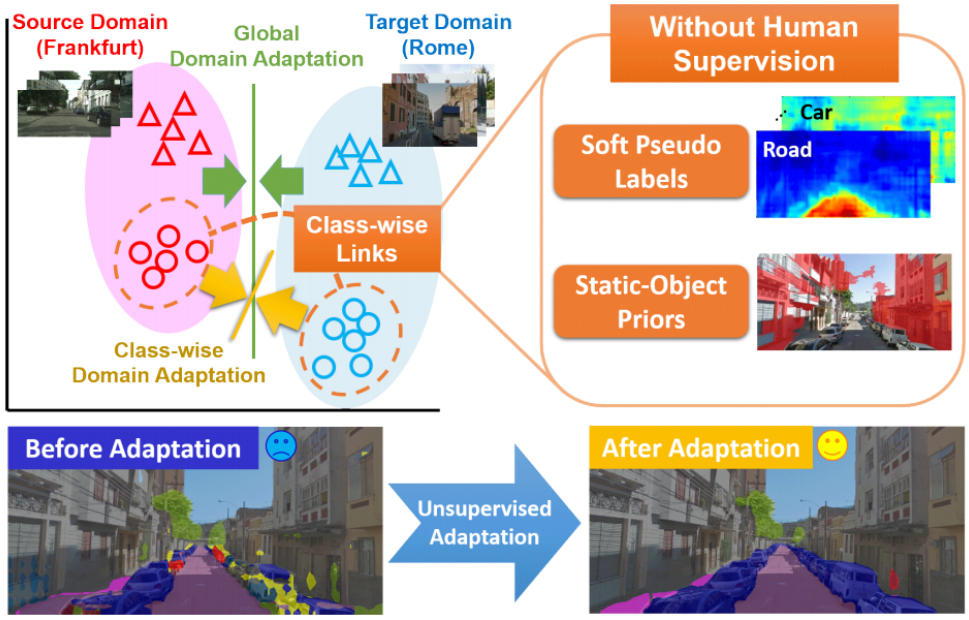


Figure 2.30: Illustration of the work presented in [73]. Pseudo-labels and class-wise adversarial learning are combined to adapt segmentation networks through different cities.

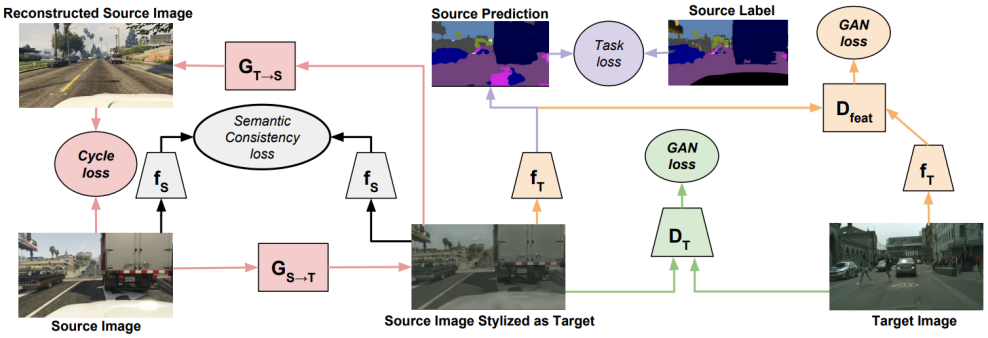


Figure 2.31: Illustration of the Cycle-consistent Adversarial Domain Adaptation in [78]. It uses separate task networks f_S and f_T for the source and target domain. The CycleGAN [55] source-to-target generator is used to stylize the source images as the target domain. The translated images are then used as supervised source for the task network. The target unlabeled images, instead, are used for feature alignment.

each class. A common choice is to adopt a separate threshold for each class and compute it based on some statistical measure of the output predictions for that class. In this way, the network may be retrained on unlabeled data to strengthen its confidence on classes that may already be correctly classified and expand it to all those new features that caused confusion and noisy outputs.

Feature level alignment, however, is not enough to adapt to different domains, which is why the most recent approaches [77, 78, 79, 80, 76] introduced also pixel-level alignment. Pixel-level alignment is based on a simple idea: aligning the input of the two domains to ease the alignment of the extracted features. In other words, the images of the source domain are modified in order to make them look more similar to the ones in the target domain. Such modifications are purely based on what neural networks extract from those images, which means that sometimes they might be tricky for a human observer. Domain adaptation methods that employ pixel-level alignment often make use of image-to-image translation (see Section 2.4) methods and modify the training objective to enforce semantic consistency between the target and the translated source domain.

Since feature-level and pixel-level alignment are both very important for domain

adaptation, these methods are the most complete and very similar to the ones developed in this work. CyCADA [78] (Figure 2.31) trains the segmentation network on images translated with CycleGAN [55] modified by semantic consistency loss. DCAN [79] adopts a custom image-to-image translation network and performs feature alignment both in the translation and in the segmentation step. CrDoCo [80] uses a cross-domain consistency loss to improve the translation. Similarly BDL [76] links translation and segmentation with a perceptual loss, where the training is iterated to gradually improve both tasks.

2.5.3 Object detection

Object detection presents less problems when facing domain shift, specially if the detector has two-stages and is trained on heterogeneous datasets. Still, as for semantic segmentation, its performance degradates when a network trained in a synthetic domain migrates to the real world.

In [81] the authors propose a method to adapt Faster R-CNN [33] across two domains. Domain adaptation components are two domain classifiers attached to Faster R-CNN with a GRL [61] (see Section 2.5). The first component classifies the image level representation of the detector, while the second classifies the instance level representation. The two predictions are then compared with a consistency regularization loss function.

In [82] domain adaptation is achieved thanks to self-training in the target domain. The detector trained on the source domain is used a pseudo-labeler for creating the target ground-truth. Detections are evaluated through the confidence of supporting regions (*i.e.* overlapping proposals). Supporting regions are all those RoIs that are usually discarded through non-maximum suppression but may indicate the presence of many activations for the detected object. Here the authors, for each output detection r^*, compute the Intersection over Union (IoU) with all RoIs. If the IoU is higher than some threshold δ, then the RoI r_i is a supporting region. Then they compute the

Supporting Region-based Reliability Score (SRRS) of each detection r^* as:

$$\text{SRRS}(r^*) = \frac{1}{N_S} \sum_{i=1}^{N_S} \text{IoU}(r_i, r^*) \cdot P(c^*|r_i) \tag{2.24}$$

where N_S is the number of supporting regions found for the detection r^*, while $P(c^*|r_i)$ is the score of RoI r_i for the detected class c^*. All those detections with $\text{SRRS}(r^*) \geq \varepsilon$ for some threshold ε are considered as pseudo-labels.

Progressive Domain Adaptation [83], instead, proposes to iteratively construct intermediate domains to perform the adaptation process. These intermediate domains are source domain translations that reduce the gap with the target domain. Intermediate data is also treated unequally based on its proximity to target.

Chapter 3

Method

3.1 Problem setting

Suppose we have a deep neural network $M : X \rightarrow Y$, where X is the input space and Y is the output space. We will refer to it as *task network*. Given an input item $x \in X$, M outputs a prediction $\hat{y} \in Y$. The task network weights are optimized in order to get a prediction similar to the ground truth $y \in Y$. In supervised learning the ground truth is available, which allows the formulation of loss functions based on the distance between y and $\hat{y}$.

Task networks, however, adapt to the domain they are trained on (*i.e.* the bias given by the input data). We will refer to this as source domain S. Suppose we train the task network in a domain where labels are easily available (*e.g.* through a simulator) and we obtain M_S. In many cases, however, we may want to deploy a task network in another domain, where labels are unknown or hard to obtain. We will refer to this as target domain T. Ideally, the best solution would be to train F directly on T, obtaining F_T, but this is not possible when Y_T is not available. The naive solution would be to substitute M_T with M_S, but the domain gap between S and T causes a huge drop in performance.

This problem, known as *domain shift*, is solved in this work through methods of Unsupervised Domain Adaptation (UDA). In other words, we show how to obtain a

working version of M_T without any label Y_T in the target domain, only by using X_S, Y_S and X_T. First, we show how to align X_S and X_T through image-to-image translation, *i.e.* pixel-level alignment. Then we present various unsupervised methods to align Y_S with Y_T, *i.e. feature-level alignment*. Different tasks are studied in this work: despite they share all the same ideas, we present a different feature aligment method for each task, since they require different implementations.

3.2 Image-to-image translation

The prominent difference between source and target domain is given by style. We refer to style as the dataset bias introduced as changes in color, light and textures. These visual changes reflect a shift in the input distribution of the task network, which is the main cause of the poor performance of the task network when deployed in the target domain.

Therefore, the first aid to domain alignment should be at pixel level. Training a CNN with images that have a similar distribution is very important to obtain similar features in the deep layers. To this end, we perform image-to-image translation on the source dataset to make it look more similar to the target one.

The training objective for the image translation model is comprised of several loss functions computed as the sum of two components, one per domain. The final objective is given by:

$$\mathscr{L} = \mathscr{L}^S + \mathscr{L}^T \tag{3.1}$$

$$\mathscr{L}^S = \lambda_{recon}\mathscr{L}^S_{recon} + \lambda_{GAN}\mathscr{L}^S_{GAN} + \lambda_{CC_l}\mathscr{L}^S_{CC_l} + \lambda_{CC_H}\mathscr{L}^S_{CC_H} + \lambda_{SCE}\mathscr{L}^S_{SCE} \tag{3.2}$$

$$\mathscr{L}^T = \lambda_{recon}\mathscr{L}^T_{recon} + \lambda_{GAN}\mathscr{L}^T_{GAN} + \lambda_{CC_l}\mathscr{L}^T_{CC_l} + \lambda_{CC_H}\mathscr{L}^T_{CC_H} + \lambda_{SCE}\mathscr{L}^T_{SCE} \tag{3.3}$$

3.2.1 Image reconstruction

Translating an image first requires the possibility to extract a latent code from it. Thus, we setup an encoder for each domain, E_S and E_T , coupled with a corresponding gen-

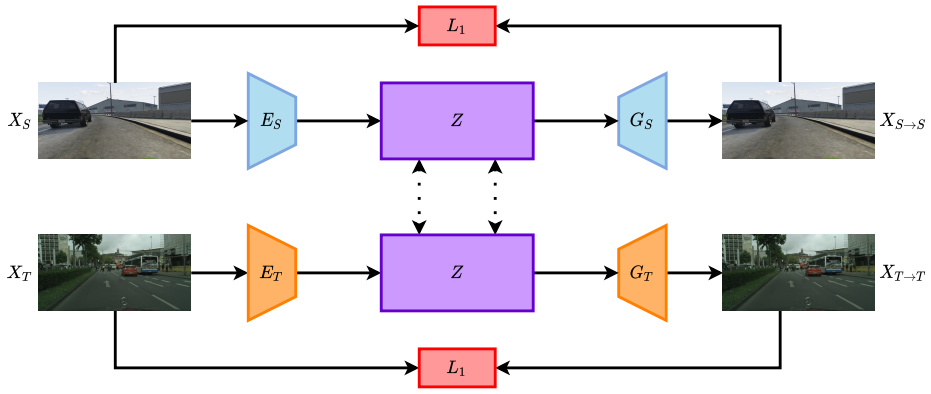


Figure 3.1: Pipeline for image reconstruction. The input images are encoded in a latent space, which is shared across the two domains. The latent codes are then used to reconstruct the input images.

erator for each domain, G_S and G_T , to form two Autoencoders. The source encoder E_S takes as input an image x_S from its domain and extracts a latent code $z \sim Z$, which is a compressed description of the image content. Here, the input style is removed through Instance Normalization layers [21] placed after each convolutional layer of the encoder. The corresponding generator G_S takes the latent code as input and uses it to generate a new image $x_{S\to S} = G_S(E_S(x_S))$, which is the reconstruction of the source image x_S. The same works in the target domain. See Figure 3.1 for an illustration of this process. The reconstruction is achieved through an L_1 loss which compares the reconstructed image with the original one:

$$\mathscr{L}^S_{recon} = \mathbb{E}_{x_S\sim X_S}\left[||x_{S\to S} - x_S||\right] \qquad \mathscr{L}^T_{recon} = \mathbb{E}_{x_T\sim X_T}[||x_{T\to T} - x_T||] \tag{3.4}$$

3.2.2 Image translation

As depicted by Figure 3.2, by combining E_S with G_T and vice versa, we get the actual translation models $F_{S\to T}$ and $F_{T\to S}$. These are trained along with two corresponding discriminators D_T and D_S, which compete against the translation models in an adver-

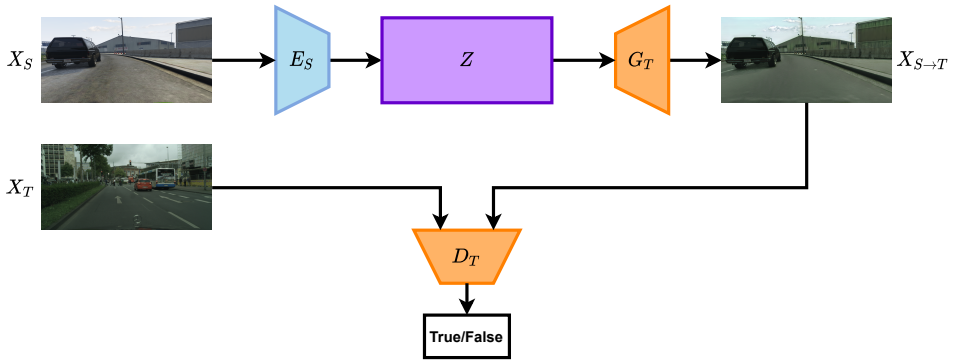


Figure 3.2: Pipeline for source-to-target translation. The input images are encoded in a latent space, which is shared across the two domains. The latent codes are then used to generate an image that looks like the target domain.

sarial fashion. We adopt the LSGAN [48] formulation (see Section 2.3) and (2.15) in this case becomes

$$\begin{aligned} \mathscr{L}_{GAN}^{S} &= \frac{1}{2}\mathbb{E}_{x_S\sim X_S}\left[(D_S(x_S))^2\right] + \frac{1}{2}\mathbb{E}_{x_T\sim X_T}\left[(D_S(x_{T\to S})-1)^2\right] \\ \mathscr{L}_{GAN}^{T} &= \frac{1}{2}\mathbb{E}_{x_T\sim X_T}\left[(D_T(x_T))^2\right] + \frac{1}{2}\mathbb{E}_{x_S\sim X_S}\left[(D_T(x_{S\to T})-1)^2\right] \end{aligned} \tag{3.5}$$

This is the loss that enables the actual change in the image distribution, but has no constraint regarding content preservation. In fact, in order to trick the discriminator, the generator might change too many details in the input image, which is why we also introduce cycle consistency.

3.2.3 Cycle consistency

Cycle consistency [55] imposes that the image translation may always be reverted (see Section 2.4). This allows the translation model to preserve content when translating, since it is necessary to revert the translation and go back to the original image. As depicted by Figure 3.3, by combining $F_{S\to T}$ with $F_{T\to S}$ and vice versa we apply

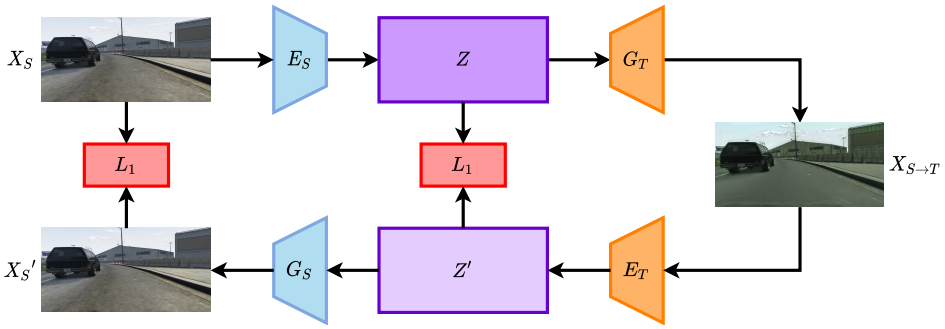


Figure 3.3: Pipeline for cycle consistency for the source-to-target translation. The translated images are translated back to the source domain. Both the reconstructed image and latent code of the target-to-source translation are compared with the ones in the source-to-target one.

the cycle consistency loss to images and latent spaces:

$$\begin{aligned} \mathscr{L}^S_{CC_I} &= \mathbb{E}_{x_S\sim X_S}[||x_{S\rightleftarrows T} - x_S||] & \mathscr{L}^T_{CC_I} &= \mathbb{E}_{x_T\sim X_T}[||x_{T\rightleftarrows S} - x_T||] \\ \mathscr{L}^S_{CC_H} &= \mathbb{E}_{z_S\sim Z}[||z_{S\to T} - z_S||] & \mathscr{L}^T_{CC_H} &= \mathbb{E}_{z_T\sim Z}[||z_{T\to S} - z_T||] \end{aligned} \tag{3.6}$$

where $z_{S\to T}$ refers to the latent space extracted by E_T from $x_{S\to T}$ and vice versa.

3.2.4 Symmetric cross-entropy

Finally, we impose that the segmentation predicted for the translated image is consistent with the one predicted for the original one through a symmetric cross-entropy loss, which is made of two contributions. For the $S \to T$ case, the first contribution assumes that $M(x_{S\to T})$ is the ground truth label and tries to align $M(x_S)$ with it. The second contribution assumes that $M(x_S)$ is the ground truth label and tries to align

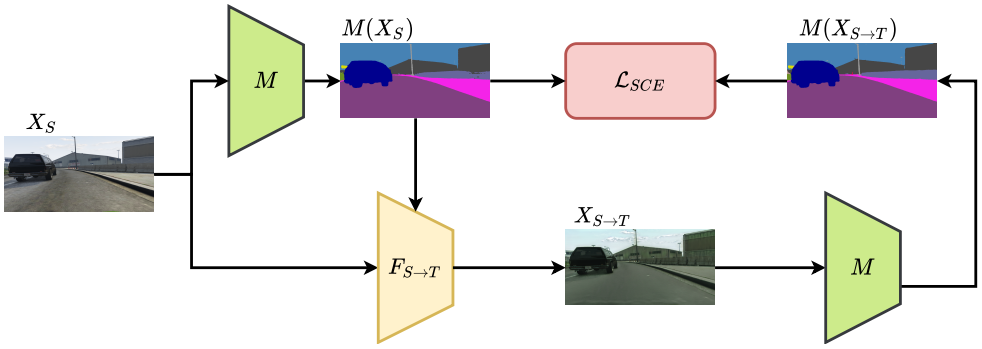


Figure 3.4: Pipeline for symmetric cross-entropy. The segmentor M is used to extract a Semantic Segmentation from both the source and translated image. The two segmentations are then compared in two cross-entropy terms.

$M(x_{S\to T})$ with it. The $T \to S$ case is symmetrical to the first one.

$$\begin{aligned}\mathscr{L}^S_{SCE} &= -\mathbb{E}_{x_S\sim X_S}\left[M(x_{S\to T})\log M(x_S)\right] - \mathbb{E}_{x_S\sim X_S}\left[M(x_S)\log M(x_{S\to T})\right] \\ \mathscr{L}^T_{SCE} &= -\mathbb{E}_{x_T\sim X_T}\left[M(x_{T\to S})\log M(x_T)\right] - \mathbb{E}_{x_T\sim X_T}\left[M(x_T)\log M(x_{T\to S})\right]\end{aligned} \tag{3.7}$$

3.2.5 Semantically adaptive generator

Recent generator architectures [57, 84, 85] make use of AdaIN to remove the source style and inject the target one (see Section 2.4.3). However, we observe that the global denormalization performed by AdaIN might be suboptimal for the image translation task. This is why we designed our generator to adaptively denormalize each pixel based on its semantics [4].

We use M to extract a segmentation map $M(x) \in \mathbb{R}^{B\times C\times H\times W}$ from the input image x, where C is the number of classes. Given an input activation $f \in \mathbb{R}^{B\times C'\times H'\times W'}$, $M(x)$ is resized to $H'\times W'$ and fed to the SPADE layer, which outputs $\gamma, \beta \in \mathbb{R}^{B\times C'\times H'\times W'}$. We then normalize f by using Instance Normalization and use γ and β to denormalize

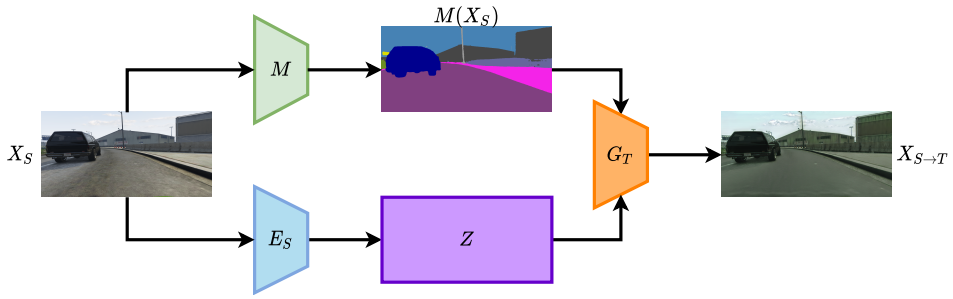


Figure 3.5: Pipeline for source-to-target translation with the semantically adaptive generator. The generator takes the segmentation prediction of M as semantic guidance, using it to adaptively denormalize each region based on its class.

it:

$$y_{b,c,h,w} = \gamma_{b,c,h,w} \frac{f_{b,c,h,w} - \mu_{b,c}}{\sigma_{b,c}} + \beta_{b,c,h,w} \tag{3.8}$$

With the semantically adaptive generator, each of the processes above implicitly take as input a semantic guidance that aids the image generation. In Figure 3.5, for example, we illustrate how the translation process becomes with semantic guidance. Table 3.1 provides all the details about the final system for our image-to-image translation architecture.

3.2.6 Representation of the semantic input

The image synthesis network of SPADE would take as input a *one-hot* encoding of the ground truth Semantic Segmentation. Here, instead, we choose to represent the semantic guidance as the unnormalized output of M for every translation that we perform. This is a consequence of the cycle consistency constraints.

We have to perform both the $S \rightleftarrows T$ and $T \rightleftarrows S$ cycles, which is why we have to train both G_S and G_T by feeding them semantic maps aligned with the input images. In UDA problems, we do not have access to Y_T, which is why we use $M(X_T)$ for the $T \rightleftarrows S$ cycle.

Encoder								
Kernel size	Stride	Input channels	Output channels	Output upsampling	Residual	Activation function	Normalization	Spectral normalization
7	1	3	64	-	-	ReLU	IN	✓
4	2	64	128	-	-	ReLU	IN	✓
4	2	128	256	-	-	ReLU	IN	✓
3	1	256	256	-	✓	ReLU	IN	✓
3	1	256	256	-	✓	ReLU	IN	✓
3	1	256	256	-	✓	ReLU	IN	✓
3	1	256	256	-	✓	ReLU	IN	✓
Generator								
Kernel size	Stride	Input channels	Output channels	Output upsampling	Residual	Activation function	Normalization	Spectral normalization
3	1	256	256	-	✓	ReLU	IN+SPADE	✓
3	1	256	256	-	✓	ReLU	IN+SPADE	✓
3	1	256	256	-	✓	ReLU	IN+SPADE	✓
3	1	256	256	✓	✓	ReLU	IN+SPADE	✓
5	1	256	128	✓	-	ReLU	LN	✓
5	1	128	64	-	-	ReLU	LN	✓
7	1	64	3	-	-	Tanh	-	✓
Discriminator (x3)								
Kernel size	Stride	Input channels	Output channels	Output upsampling	Residual	Activation function	Normalization	Spectral normalization
4	2	3	64	-	-	$\text{LReLU}_{0.2}$	-	✓
4	2	64	128	-	-	$\text{LReLU}_{0.2}$	-	✓
4	2	128	256	-	-	$\text{LReLU}_{0.2}$	-	✓
4	2	256	512	-	-	$\text{LReLU}_{0.2}$	-	✓
1	1	512	1	-	-	-	-	✓

Table 3.1: **Detailed architecture of encoders, generators and discriminators in the image-to-image translation step.** The architectures follow the schemes adopted by CycleGAN and UNIT. *Output upsampling* indicates that we use a $2\times$ nearest-neighbor upsampling of the output feature maps. *Residual* indicates that the layer is actually a residual block, not a simple convolutional one. $\text{LReLU}_{0.2}$ indicates the Leaky Rectified Linear Unit with slope $\alpha = 0.2$.

However, we note that the refined output classes predicted by M are far from the ground truth and cannot give an accurate conditioning, especially in the target domain when the segmentation is still in the initial training phases. Because of this, we choose to use as semantic guidance the unnormalized output of M. This representation has the advantage of carrying the confidence of the prediction, which could potentially be used by the SPADE layers to avoid denormalizing a region with the incorrect class (*e.g.* on the borders of objects, where the segmentation tends to fail more easily).

In the $S \rightleftarrows T$ cycle, we could use Y_S as semantic guidance, but this would lead to inconsistent input distributions for the SPADE layers, which is why we adopt $M(X_S)$ as semantic guidance in this case too.

3.3 Semantic Segmentation

When the task network is given by a Semantic Segmentation model M, features are aligned during training by combining supervision on $X_{S\to T}$, self-supervision on X_T and adversarial learning. Therefore, the loss function used when training the segmentor is given by

$$\mathscr{L} = \lambda_{seg}\mathscr{L}_{seg} + \lambda_{SSL}\mathscr{L}_{SSL} + \lambda_{adv}\mathscr{L}_{adv} \tag{3.9}$$

3.3.1 Supervised training

The main supervision for the segmentation task is given by training the network on $(X_{S\to T}, Y_S)$, where $X_{S\to T}$ are images translated from the synthetic to the real domain. This is formulated as the common cross-entropy loss:

$$\mathscr{L}_{seg} = -\mathbb{E}_{x\sim X_{S\to T}, y\sim Y_S} \sum_{k=1}^{K} \mathbf{1}_{[k=y]} \log(M(x)_k) \tag{3.10}$$

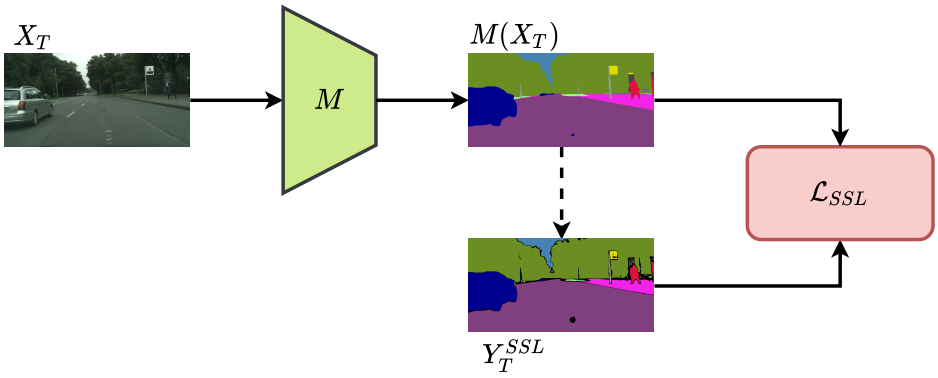


Figure 3.6: Pipeline for self-supervised learning. Before training, the predictions for the target domain are thresholded to construct the pseudo labels Y_T^{SSL}. Low-confidence predictions (in black) are set to the ignored class and do not contribute to the final objective during training.

3.3.2 Self-supervised training

We also adopt self-supervision to improve the adaptation model [76]. To this end, we compute $M(X_T)$ and use the high confidence predictions as labels, creating Y_T^{SSL}:

$$Y_T^{SSL} = \begin{cases} \arg\max_{1 \le k \le K} M(X_T)_k, & \text{if } M(X_T)_k \ge th_k^{SSL} \\ -1, & \text{otherwise} \end{cases} \tag{3.11}$$

where K is the number of classes, -1 is the index ignored and th_k^{SSL} is the confidence threshold, which we use to filter the uncertain predictions. We select a different threshold for each class k as the median score for that class across the whole dataset. When computing the median for class k, we consider only the scores where the predicted class is k.

This makes us able to compute a cross-entropy loss also on the target dataset:

$$\mathscr{L}_{SSL} = -\mathbb{E}_{x \sim X_T, y \sim Y_T^{SSL}} \sum_{k=1}^{K} \mathbf{1}_{[k=y]} \log(M(x)_k) \tag{3.12}$$

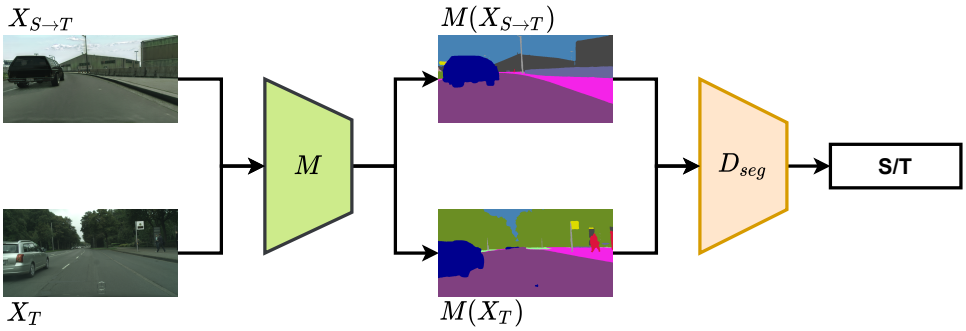


Figure 3.7: Pipeline for adversarial training. The segmentor outputs predictions for both domains, which are fed to the segmentation discriminator that acts as domain classifier. The discriminator tries to distinguish the target segmentations from the source ones. By learning how to trick the discriminator, the generator (*i.e.* the segmentor) will output more realistic and aligned semantic maps.

3.3.3 Adversarial training

Supervision on pixel-level aligned images and self-supervision on target images are not enough to learn a full model. This is why we also make use of adversarial training by feeding the semantic maps to a discriminator D_{seg}, which has to distinguish the maps predicted by M for S and T (see Figure 3.7) . This leads to the formulation of a GAN objective (similar to (2.12)):

$$\mathscr{L}_{adv,seg} = \mathbb{E}_{x_T\sim X_T}[\log(D_{seg}(M(x_T)))] + \mathbb{E}_{x_{S\to T}\sim X_{S\to T}}[\log(1 - D_{seg}(M(x_{S\to T})))] \tag{3.13}$$

This loss enforces an output space alignment [72], which means that M has to learn how to predict semantic maps with distributions that are aligned regardless of the input domain. We provide details about the architecture of D_{seg} in Table 3.2, where we do not use residual connections nor normalization layers.

D_{seg}				
Kernel size	Stride	Input channels	Output channels	Activation function
4	2	N_C	64	$\text{LReLU}_{0.2}$
4	2	64	128	$\text{LReLU}_{0.2}$
4	2	128	256	$\text{LReLU}_{0.2}$
4	2	256	512	$\text{LReLU}_{0.2}$
4	2	512	1	Sigmoid

Table 3.2: **Detailed architecture of discriminator in the adversarial training for Semantic Segmentation.** $\text{LReLU}_{0.2}$ indicates the Leaky Rectified Linear Unit with slope $\alpha = 0.2$. N_C indicates the number of classes considered for segmentation.

3.4 Semantically adaptive image-to-image translation for domain adaptation of Semantic Segmentation

By combining pixel-level and feature-level alignment, we obtain a full system for domain adaptation of Semantic Segmentation, as depicted by Figure 3.8. This system is designed to enforce cooperation between its two subparts. Pixel-level alignment reduces the input gap between domains, aiding feature-level alignment by making it easier to extract similar features. Feature-level alignment actually improves the segmentor performance on both domains, making it a better semantic guidance for the image translation process.

3.5 Object detection

We adopt Faster R-CNN [33] as reference Object Detection model (see Section 2.2.3). When training it as task network M, we combine supervision on X_S and adversarial training with X_T.

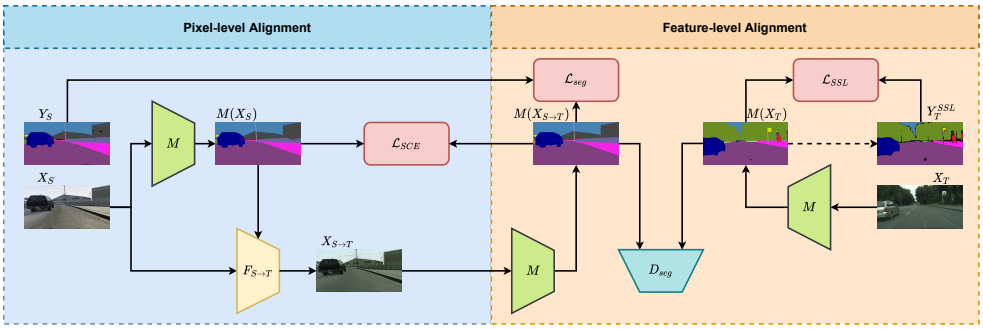


Figure 3.8: Full system employed when adapting Semantic Segmentation. It comprises two main parts: pixel-level alignment (*i.e.* image-to-image translation) and feature-level alignment (*i.e.* adversarial training and self-supervised learning). Cycle consistency and reconstruction parts are omitted for simplicity.

3.5.1 Supervised training

The main supervision for the detection task is given by training the network on (X_S, Y_S), where Y_S represents the ground-truth bounding boxes as classes U_S and coordinates B_S. The loss function for the task is then given by a combination of classification and regression losses:

$$\mathscr{L}_{det} = \mathscr{L}_{cls} + \mathscr{L}_{reg} \tag{3.14}$$

$$\mathscr{L}_{cls} = -\mathbb{E}_{x\sim X_S, u\sim U_S} \sum_{k=1}^{K} \mathbf{1}_{[k=u]} \log(M_u(x)_k) \tag{3.15}$$

$$\mathscr{L}_{reg} = -\mathbb{E}_{x\sim X_S, u\sim U_S, b\sim B_S} \sum_{k=1}^{K} \mathbf{1}_{[k=u]} smooth_{L_1}(M_b(x) - b) \tag{3.16}$$

where $M_u(x)$ and $M_b(x)$ are the predicted class and coordinates respectively.

D_{det}			
Input channels	Output channels	Activation function	Dropout
1024	1024	ReLU	0.5
1024	1024	ReLU	0.5
1024	1	Sigmoid	-

Table 3.3: **Detailed architecture of discriminator in the adversarial training for Object Detection.**

3.5.2 Adversarial training

In order to align the predicted boxes across the two domains, we also employ a domain classifier when training the Object Detection model (as in Section 3.3.3). Faster R-CNN uses 1D features to predict the boxes classes and coordinates. Therefore, the discriminator used to classify their domain is a multi-layer perceptron (*i.e.* composed by fully-connected, or linear, layers). Its architecture is described in Table 3.3 and it is optimized with the following objective (from (2.12)):

$$\mathscr{L}_{adv,det} = \mathbb{E}_{x_T \sim X_T}[\log(D_{det}(M(x_T)))] + \mathbb{E}_{x_S \sim X_S}[\log(1 - D_{det}(M(x_S)))] \quad (3.17)$$

where $M(x)$ are the features extracted from the image x and used by the output layer to predict $M_u(x)$ and $M_b(x)$. In order to get a correctly balanced adversarial training, all region proposals are taken into account during this process, without performing any non-maximum suppression or sampling to balance the positive-negative ratio.

3.6 Instance segmentation

We adopt Mask R-CNN [38] as reference Instance Segmentation model. Mask R-CNN is a simple and straightforward extension of Faster R-CNN with the addition of a mask branch (see Section 2.2.4). When training it as task network M, we combine supervision on X_S and adversarial training with X_T.

3.6.1 Supervised training

The main supervision for the Instance Segmentation task is given by training the network on (X_S, Y_S), where Y_S represents the ground-truth instances as classes U_S, box coordinates B_S and masks V_S. The loss function for the task is then given by a combination of classification, regression and segmentation losses:

$$\mathscr{L}_{ins} = \mathscr{L}_{det} + \mathscr{L}_{mask} \tag{3.18}$$

$$\mathscr{L}_{mask} = -\mathbb{E}_{x\sim X_S, v\sim V_S}[y\log(M_{vk}(x)) + (1-y)\log(1-M_{vk}(x))] \tag{3.19}$$

where $M_vk(x)$ is the mask predicted for the corresponding ground-truth class k and $\mathscr{L}_{det}$ is the objective of Object Detection in (3.14).

3.6.2 Adversarial training

In order to align the predicted masks across the two domains, we also employ a domain classifier when training the instance model (as in Section 3.3.3). Mask R-CNN uses a small segmentation decoder in its mask branch. Therefore, the discriminator used to classify their domain is smaller than the one employed in Semantic Segmentation. Its architecture is detailed in Table 3.4 and it is optimized with the following objective (from (2.12)):

$$\mathscr{L}_{adv,ins} = \mathbb{E}_{x_T\sim X_T}[\log(D_{mask}(M(x_T)))] + \mathbb{E}_{x_S\sim X_S}[\log(1-D_{mask}(M(x_S)))] \tag{3.20}$$

D_{ins}				
Kernel size	Stride	Input channels	Output channels	Activation function
4	2	N_C	64	$\text{LReLU}_{0.2}$
4	2	64	128	$\text{LReLU}_{0.2}$
4	2	128	1	Sigmoid

Table 3.4: **Detailed architecture of discriminator in the adversarial training for Instance Segmentation.** $\text{LReLU}_{0.2}$ indicates the Leaky Rectified Linear Unit with slope $\alpha = 0.2$.

where $M(x)$ are the features extracted from the image x and used by the output layer to predict $M_v(x)$. In order to get a correctly balanced adversarial training, all region proposals are taken into account during this process, as for Object Detection.

3.7 Panoptic segmentation

As described in Section 2.2.5, Panoptic Segmentation [42] is a task that in some way merges instance and Semantic Segmentation. Likewise, we perform Panoptic Domain Adaptation by merging the methods developed for Instance and Semantic Segmentation. We adopt Panoptic FPN [43] as reference model, which constructs the panoptic output as post-processing of instance masks and semantic map.

3.7.1 Supervised training

The main supervision for the Panoptic Segmentation task is given by training the network on (X_S, Y_S), where Y_S represents the ground-truth semantic map and instances. The loss function for the task is then given by a combination of semantic and instance losses:

$$\mathscr{L}_{pan} = \mathscr{L}_{seg} + \mathscr{L}_{ins} \tag{3.21}$$

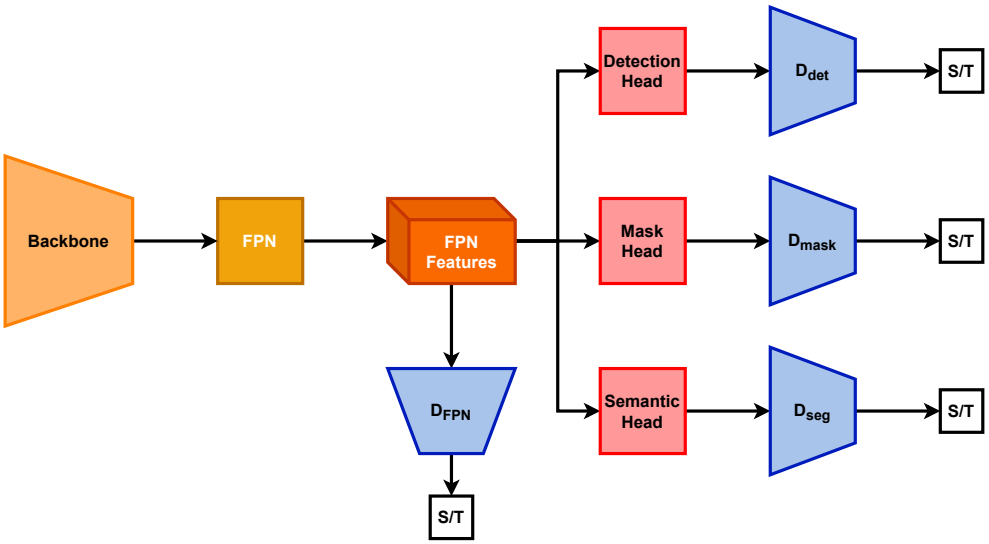


Figure 3.9: Pipeline for full adversarial training of Panoptic FPN. Each of the outputs necessary to construct the Panoptic Segmentation is adapted separately. Moreover, the features extracted from the FPN module are adapted by a separate discriminator, named D_{FPN}

where $\mathscr{L}_{seg}$ is the objective of Semantic Segmentation in (3.10) and $\mathscr{L}_{ins}$ is the objective of Instance Segmentation in (3.18).

3.7.2 Adversarial training

The construction of the panoptic output is not directly differentiable and works with the refined outputs, which is why it is not possible to adapt panoptic features like it is done for the subtasks. Moreover, this would be suboptimal, since any region overwritten by the other output would be hidden to the adaptation process.

To enforce the adaptation of both tasks simultaneously, we implement an FPN discriminator. This discriminator is, as usual, a domain classifier, but takes as input each feature map extracted by the Feature Pyramid Network (FPN) of Panoptic FPN. FPN processes the features extracted from the backbone and outputs a feature map

D_{FPN}				
Kernel size	Stride	Input channels	Output channels	Activation function
1	1	256	512	ReLU
1	1	512	1	Sigmoid

Table 3.5: **Detailed architecture of discriminator in the adversarial training for Feature Pyramid Network.** $\text{LReLU}_{0.2}$ indicates the Leaky Rectified Linear Unit with slope $\alpha = 0.2$.

for each scale, all having the same number of channels. This allows us to use the same discriminator to classify their domain. The FPN discriminator is important to strengthen the alignment, since it operates on deep features. We provide details about its architecture in Table 3.5 and it is optimized with the following objective (from (2.12)):

$$\mathscr{L}_{adv,fpn} = \mathbb{E}_{x_T \sim X_T}[\log(D_{FPN}(M_{FPN}(x_T)))] + \mathbb{E}_{x_S \sim X_S}[\log(1 - D_{FPN}(M_{FPN}(x_S)))] \tag{3.22}$$

where $M_{FPN}(x)$ are the FPN features extracted from the image x.

Note that all the discriminators presented so far can be combined in panoptic adaptation, which is why we present an ablation study about their use in Section 4.5.1.

Chapter 4

Experiments

We present our experimental results for the synthetic to real adaptation using various dataset settings and architectures. First we present the datasets we used in our experiments and the reasons behind their choice. Then we show the results of adapting Semantic Segmentation using both feature-level alignment and semantically adaptive translation. Finally we extend the feature alignment for Semantic Segmentation to panoptic domain adaptation and provide an ablation study of the methods presented for it. We use PyTorch [86] to implement our methods and perform all our experiments.

4.1 Datasets

Here we present the datasets used for our work. In order to compare with the state of the art of domain adaptation for Semantic Segmentation, we adopt the same synthetic-to-real settings of related works: GTA5 [87] to Cityscapes [2] and SYNTHIA [88] to Cityscapes. Since there are no other works about panoptic domain adaptation, we choose to adapt VIPER [89] to Cityscapes, since GTA5 [87] has no instance annotations. In the following we present the main features of these datasets.

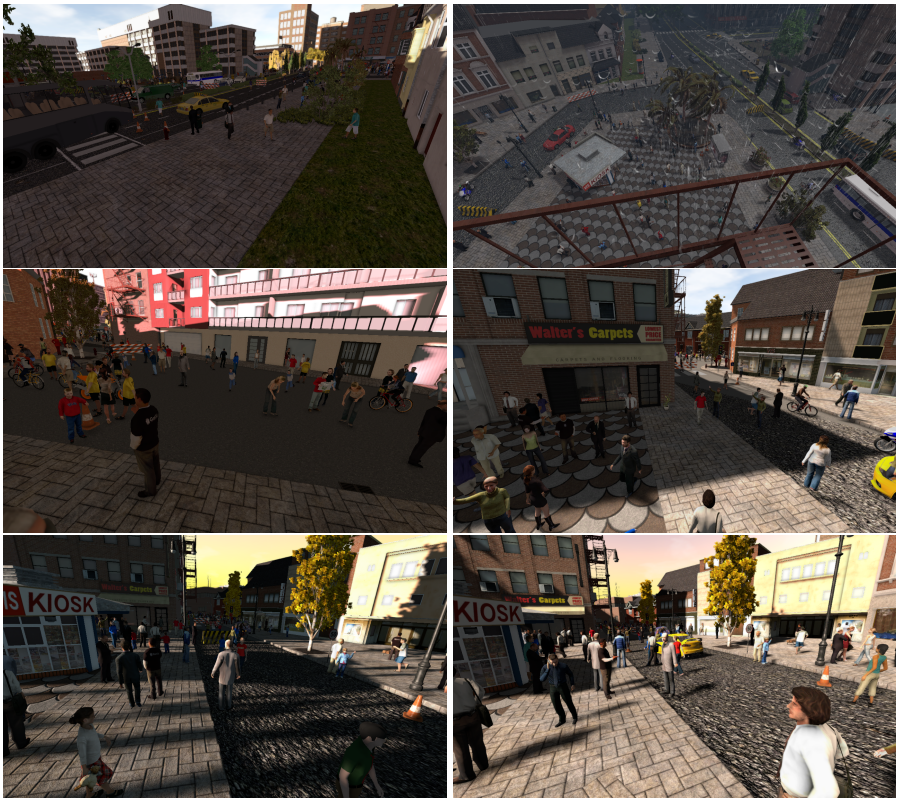


Figure 4.1: Example of images taken from SYNTHIA [88].

4.1.1 SYNTHIA

SYNTHIA [88], which stands for SYNTHetic collection of Imagery and Annotations, is a synthetic dataset that simulates urban environments with many classes and instances. However, SYNTHIA is not a simulator: each image is generated as a random perturbation of the environment and there is no temporal consistency. Such perturbation is able to create a good heterogenous dataset, with different lighting and weather conditions, which is very important for deep learning systems. The images have a resolution of 960 × 720 pixels and horizontal field of view of 100 degrees. One subset of SYNTHIA is SYNTHIA-Rand-Cityscapes, which provides semantic

labels for 9000 images aligned with a subset of the Cityscapes classes, which makes it a good choice for domain adaptation. The only missing classes are *terrain*, *truck* and *train*. As can be seen in Figure 4.1, SYNTHIA lacks of photo-realism, since its textures and details are very far from real ones.

4.1.2 GTA5

Figure 4.2: Example of images taken from GTA5 [87].

We refer to GTA5 as the dataset created by [87] using the famous videogame. Given the possibility of driving in realistic scenarios, the videogame has been used to capture images of urban scenes and the engine has been exploited to extract the Semantic Segmentation of each frame. The dataset consists of 24966 non-consecutive frames with a resolution of 1914×1052 pixels. GTA5 provides labels for all the classes used for the evaluation of Cityscapes. Moreover, the photo-realism of its mod-

ern computer graphics makes it a good choice for domain adaptation, since its shapes and textures are more similar to that of real objects, as can be seen in Figure 4.2.

4.1.3 VIPER

VIPER [89] is an extension of GTA5 that sets up the possibility to benchmark many perception algorithms using the same dataset. In fact, VIPER provides video sequences completely labelled with Panoptic Segmentation, optical flow and visual odometry, which may be easily used as training sets for the corresponding tasks. Summing all sequences, VIPER is comprised of 253814 images, split as 134097 for training, 49815 for validation and 69902 for testing. The testing labels are unavailable to keep the benchmark fair. Besides using a resolution of $1920x1080$ pixels per for the images, the dataset keeps the same features of GTA5, like photo-realism, but also increases its variety providing more scenarios with different weather conditions and daytime.

4.1.4 Cityscapes

Cityscapes [2] is one of the most famous real datasets for Semantic Segmentation in urban driving. It is recorded across 50 cities in Europe and provides 5000 images with fine annotations and 20000 images with coarse annotations, all having a resolution of 2048x1024 pixels. The labels are both class-wise and instance-wise, making it also suited for the introduction of the panoptic benchmark. Cityscapes provides annotations for 34 classes, but only 19 are evaluated. This is because many classes have few instances, which makes them also harder to detect for models that learn from data (*e.g.* CNNs). Besides the segmentation annotations, the dataset also provides the vehicle odometry, GPS coordiantes, temperature, stereo images, pre-computed depth maps and HDR images. Like the related work, we adopt Cityscapes as target real dataset for all our experiments, but we do not use any of the labels it provides during training. We only use its images in the adaptation process and the labels from the validation set to evaluate our method.

Figure 4.3: Example of images taken from Cityscapes [2].

4.2 Metrics

Here we present the metrics we measure in our experiments. They are the common choices for the tasks at hand and we adopt them to compare with the related work.

4.2.1 Intersection over Union

Given a prediction x and a ground-truth y, Intersection over Union (IoU) is measured as follows:

$$IoU(x,y) = \frac{TP}{TP+FP+FN} \tag{4.1}$$

where TP are true positives, FP are false positives and FN are false negatives. The way these quantities are computed changes based on the task. For example,

in Semantic Segmentation they depend on the match between the predicted and the ground-truth class for a pixel. Usually the evaluated quantity is the mean IoU (mIoU), which is the mean of the IoUs for each class. In Object Detection, instead, the calculation is easier, since intersection and union can be easily computed as areas in the image.

4.2.2 Panoptic Quality

Given a prediction x and ground truth y, Panoptic Quality (PQ) [42] is measured as follows:

$$PQ = \underbrace{\frac{\sum_{(x,y)\in TP} IoU(x,y)}{TP}}_{\text{Segmentation Quality (SQ)}} \cdot \underbrace{\frac{TP}{TP+\frac{1}{2}FP+\frac{1}{2}FN}}_{\text{Recognition Quality (RQ)}} = \frac{\sum_{(x,y)\in TP} IoU(x,y)}{TP+\frac{1}{2}FP+\frac{1}{2}FN} \tag{4.2}$$

Segmentation Quality evaluates the IoU of matched segments, while Recognition Quality is the $F1$ score and evaluates how many matches were found. In order to understand the $F1$ score, we also give the definition of *Precision* and *Recall*:

$$Precision = \frac{TP}{TP+FP} \tag{4.3}$$

$$Recall = \frac{TP}{TP+FN} \tag{4.4}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.5}$$

In other words the $F1$ score is the harmonic mean of *Precision* and *Recall*, which evaluate respectively the number of true positives *w.r.t.* the number of predicted positives and *w.r.t.* the number of actual positives.

4.2.3 Inception score

In order to evaluate the visual quality in the results of image synthesis or translation, a commonly adopted metric is the Inception Score (IS) [90]. It simultaneously measures two things:

- How distinctly each image looks like something.
- The variety of the generated images.

This is computed through the scores predicted by the Inception [26] classifier pretrained on ImageNet [23]. By running the network on the generated dataset, we compute two things:

- The label distribution of each image.
- The marginal distribution.

In order to have a distinct object in an image we expect the produced scores to be focused on a particular class. In order to obtain high variance, instead, we want the marginal distribution to be uniform. A good measure to combine these two factors is therefore the distance between the two distributions. The IS, in fact, has been formulated as the exponential of the Kullback–Leibler divergence [91] [92] between label and marginal distributions:

$$\mathrm{IS} = exp(\mathbb{E}_x[\mathrm{KL}(p(y|x)p(y))]) \tag{4.6}$$

It has been noted [93], however, that relying on IS has some drawbacks, mostly due to the limitations of its training data. For example, if we try to generate classes not present in the training data, they will get low scores despite having high quality, since Inception in unable to classify them as distinct classes.

4.2.4 Frechét Inception Distance

Frechét Inception Distance (FID) [94] is a measure of distance between datasets. As IS it is based on the Inception [26] model, which is used to extract deep features

from the 2048 channels pool3 layer. The features are extracted from both real and generated images and modeled as two multivariate Gaussians $X_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ and $X_g \sim \mathcal{N}(\mu_g, \Sigma_g)$. Then, FID is the Frechét distance between the two Gaussians:

$$\text{FID} = \|(\mu_r - \mu_g)\| + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}) \tag{4.7}$$

where Tr is the trace of the trace of a matrix. FID is very useful to measure the quality of image translation: given the real domain, we can see how close a translated one is by measuring the FID between them, which we do in Section 4.3.6.

4.3 Semantically adaptive image-to-image translation for domain adaptation of Semantic Segmentation

We perform domain adaptation of Semantic Segmentation using two synthetic to real dataset settings: GTA5 [87] to Cityscapes [2] and SYNTHIA [88] to Cityscapes. We evaluate the mean intersection-over-union (IoU) on the Cityscapes validation set and show how our method outperforms the current state-of-the-art by adopting the same segmentation models. Finally, we conduct an ablation study to highlight the value of our contributions. See Section 2.5.2 for details about the methods we compare with.

4.3.1 Segmentation network

We choose to adapt two segmentation networks: DeepLabV2 [9] with ResNet101 [27] and FCN8s [29] with VGG16 [25]. Both networks are trained on images downsampled to 1024x512 with batch size 1.

We initialize the segmentation networks from [76] to speed up the training process. However, in order to show the independence of our work from this initialization, we also conduct one experiment by training the network from scratch. In the GTA5 to Cityscapes setting, we train DeepLabV2 initialized only by the commonly used ImageNet [23] pretraining, which we find to be in line with the results that we get by initializing it with [76].

Both DeepLabV2 and FCN8s are trained on images downsampled to 1024x512 with batch size 1. We set $\lambda_{seg} = 1$ and $\lambda_{SSL} = 1$ for both networks. For DeepLabV2 we set $\lambda_{adv} = 10^{-3}$, while for FCN8s we set $\lambda_{adv} = 10^{-4}$.

DeepLabV2 is trained using SGD as optimizer with a 'poly' learning rate policy:

$$lr_i = lr_0 \cdot \left(1 - \frac{i}{i_{max}}\right)^{\gamma} \tag{4.8}$$

where lr_i is the learning rate at iteration i. Here we set the decay $\gamma = 0.5$, the maximum number of iterations to $i_{max} = 250000$ and the initial learning rate to $lr_0 = 2.5 \cdot 10^{-4}$.

FCN8s is trained using Adam [95] as optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$. The learning rate follows a 'step' policy:

$$lr_i = lr_0 \cdot \gamma^{\lfloor \frac{i}{\omega} \rfloor} \tag{4.9}$$

where the initial learning rate is $lr_0 = 10^{-5}$, the drop is $\gamma = 0.1$ and the step size is $\omega = 5000$.

The discriminator for the adversarial training is similar to [96] and is trained using Adam with $lr_0 = 10^{-4}$ for DeepLabV2 and $lr_0 = 10^{-6}$ for FCN8s. In both cases we set $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

4.3.2 Translation network

For the translation part, we describe the architecture of the encoders, generators and discriminators.

The encoder is made by few downsampling blocks, followed by residual blocks for further processing of the latent code and they all use IN [21]. Symmetrically, the generators take in the latent code and process it with residual blocks, where IN and SPADE are combined to normalize the feature maps. These are followed by upsampling blocks with Layer Normalization [22]. We found LN to better preserve the style in the generated activations.

In each domain we have discriminators for multiple scales [53], each being a

Patch Discriminator [52, 97]. The GAN [46] objective we choose is the one proposed in LSGAN [48]. We apply Spectral Normalization [98] to all the models described here.

When training the translation model we resize the input images to 1024x512 and take 512x512 random crops out of them. We use Adam [95] as optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$. We apply TTUR [94] and set the initial learning rate to be 10^{-4}. The learning rate is scheduled to decay to 0 after 1000000 iterations with a 'poly' scheduling where the power is 0.9. The batch size is 1 for all the experiments. The loss weights are set to $\lambda_{recon} = 10$, $\lambda_{GAN} = 1$, $\lambda_{CC_l} = 10$, $\lambda_{CC_H} = 1$, $\lambda_{SCE} = 10$.

We present some visual results of how the translation happens with DeepLabV2 in Figure 4.4 for GTA5 to Cityscapes and in Figure 4.5 for SYNTHIA to Cityscapes.

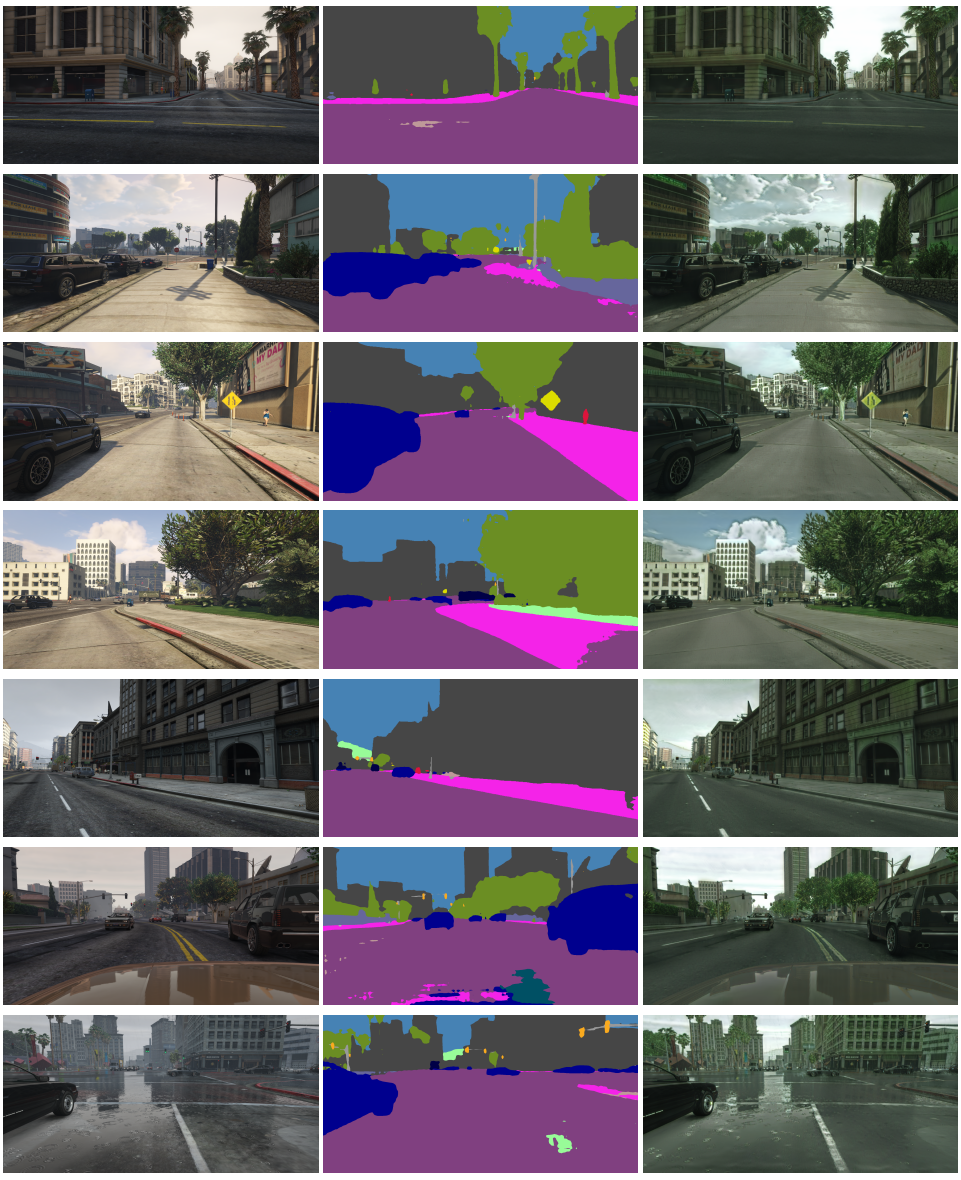
4.3.3 Bidirectional learning

Pixel-level and feature-level alignment are not performed in an end-to-end fashion. Besides being highly expensive in terms of memory requirements, we found this approach to be very unstable and it did not lead to good results.

We iteratively recreate $X_{S \to T}$ when M stops improving on the target dataset. Before each training of the segmentation network, we also generate new pseudo-labels Y_T^{SSL}. We found this procedure to significantly improve the final mIoU compared to a single iteration of pixel-level and feature-level alignment.

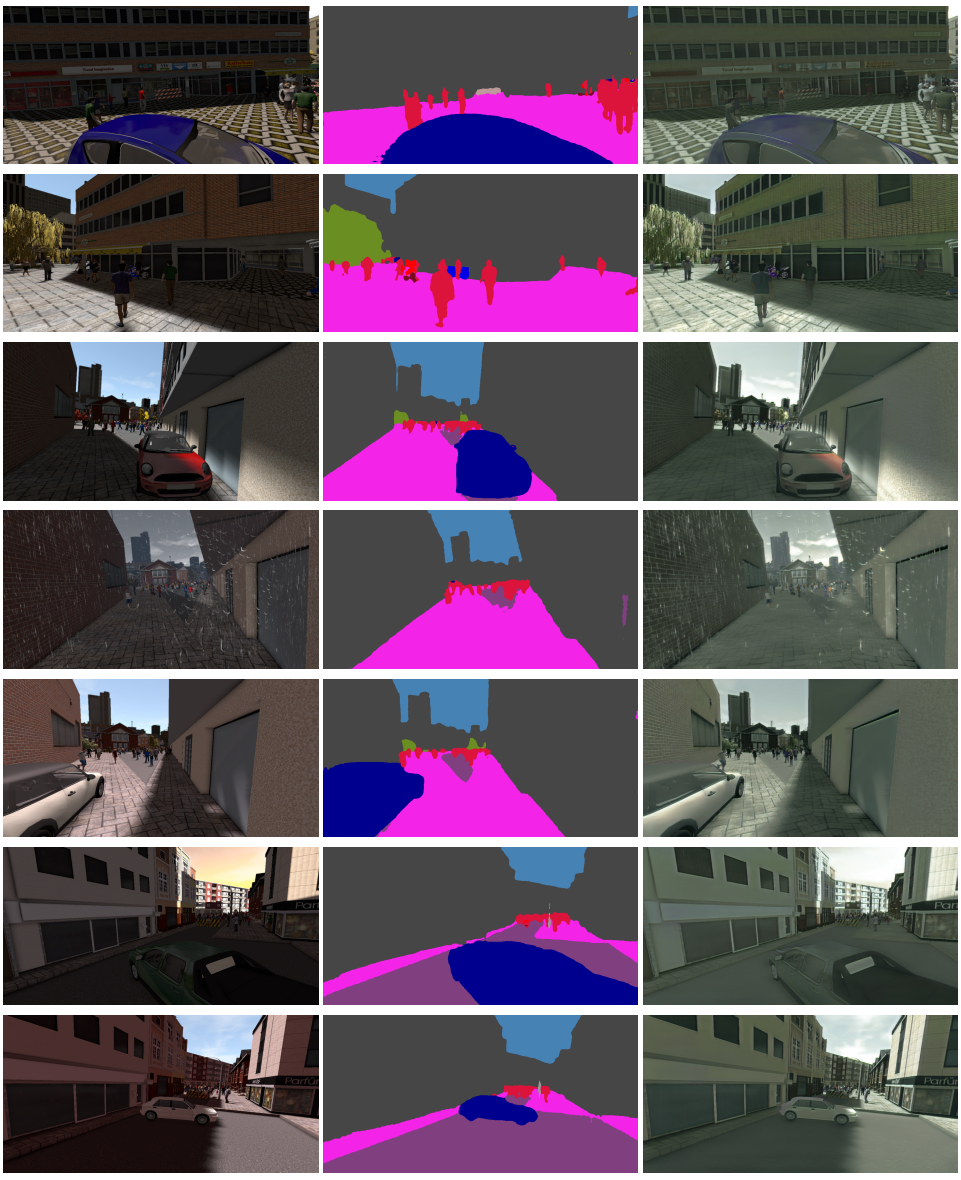
4.3.4 Comparison with State of the Art

GTA5 to Cityscapes For the GTA5 [87] to Cityscapes [2] task, we evaluate on all the 19 classes used in the Cityscapes benchmark since the datasets are fully compatible. Some visual results for this setting are presented in Figure 4.6. In this case, the upper bounds in terms of mIoU are 65.1 for DeepLabV2 [9] and 60.3 for FCN8s [29], which are the results achievable by training with the target labels. In Table 4.1 we compare our results with the related work. In terms of mIoU, we get respectively +1.9% and +4.2% over the state-of-the-art with the two networks.



GTA5 sample GTA5 pred GTA5→CS

Figure 4.4: **Translations from GTA5 to Cityscapes.** We take a sample X_S from GTA5, get the predicted segmentation using M, and generate $X_{S\to T}$. We present the results obtained with DeepLabV2 used as semantic guidance.



SYNTHIA sample SYNTHIA pred SYNTHIA→CS

Figure 4.5: **Translations from SYNTHIA to Cityscapes.** We take a sample X_S from SYNTHIA, get the predicted segmentation using M, and generate $X_{S \to T}$. We present the results obtained with DeepLabV2 used as semantic guidance.

GTA5 → Cityscapes																				
Method	Arch.	road	sidewalk	building	wall	fence	pole	light	sign	veget	terrain	sky	person	rider	car	truck	bus	train	mbike	bike	mIoU
Cycada [78]	D	86.7	35.6	80.1	19.8	17.5	**38.0**	39.9	**41.5**	82.7	27.9	73.6	**64.9**	19	65.0	12.0	28.6	4.5	31.1	**42.0**	42.7
AdaptSegNet [72]	D	86.5	25.9	79.8	22.1	20.0	23.6	33.1	21.8	81.8	25.9	75.9	57.3	26.2	76.3	29.8	32.1	**7.2**	29.5	32.5	41.4
DCAN [79]	D	85.0	30.8	81.3	25.8	21.2	22.2	25.4	26.6	83.4	36.7	76.2	58.9	24.9	80.7	29.5	42.9	2.5	26.9	11.6	41.7
CLAN [74]	D	87.0	27.1	79.6	27.3	23.3	28.3	35.5	24.2	83.6	27.4	74.2	58.6	28.0	76.2	33.1	36.7	6.7	31.9	31.4	43.2
BDL [76]	D	91.0	**44.7**	84.2	34.6	**27.6**	30.2	36.0	36.0	85.0	43.6	83.0	58.6	31.6	83.3	**35.3**	**49.7**	3.3	28.8	35.6	48.5
Ours	D	**91.2**	43.3	**85.2**	**38.6**	25.9	34.7	**41.3**	41.0	**85.5**	**46.0**	**86.5**	61.7	**33.8**	**85.5**	34.4	48.7	0.0	**36.1**	37.8	**50.4**
Curriculum [71]	F	74.9	22.0	71.7	6.0	11.9	8.4	16.3	11.1	75.7	13.3	66.5	38.0	9.3	55.2	18.8	18.9	0.0	16.8	16.6	28.9
CBST [75]	F	66.7	26.8	73.7	14.8	9.5	**28.3**	25.9	10.1	75.5	15.7	51.6	47.2	6.2	71.9	3.7	2.2	5.4	18.9	32.4	30.9
Cycada [78]	F	85.2	37.2	76.5	21.8	15.0	23.8	22.9	21.5	80.5	31.3	60.7	50.5	9.0	76.9	17.1	28.2	4.5	9.8	0.0	35.4
DCAN [79]	F	82.3	26.7	77.4	23.7	20.5	20.4	**30.3**	15.9	80.9	25.4	69.5	52.6	11.1	79.6	24.9	21.2	1.3	17.0	6.7	36.2
LSD [99]	F	88.0	30.5	78.6	25.2	23.5	16.7	23.5	11.6	78.7	27.2	71.9	51.3	19.5	80.4	19.8	18.3	0.9	20.8	18.4	37.1
CLAN [74]	F	88.0	30.6	79.2	23.4	20.5	26.1	23.0	14.8	81.6	34.5	72.0	45.8	7.9	80.5	**26.6**	29.9	0.0	10.7	0.0	36.6
CrDoCo [80]	F	89.1	33.2	80.1	26.9	25.0	18.3	23.4	12.8	77.0	29.1	72.4	**55.1**	20.2	79.9	22.3	19.5	1.0	20.1	18.7	38.1
BDL [76]	F	89.2	40.9	81.2	29.1	19.2	14.2	29.0	19.6	83.7	35.9	80.7	54.7	23.3	**82.7**	25.8	28.0	2.3	25.7	19.9	41.3
Ours	F	**91.1**	**46.4**	**82.9**	**33.2**	**27.9**	20.6	29.0	**28.2**	**84.5**	**40.9**	**82.3**	52.4	**24.4**	81.2	21.8	**44.8**	**31.5**	**26.5**	**33.7**	**46.5**

Table 4.1: Results of adapting GTA5 [87] to Cityscapes [2]. D stands for DeepLabV2 [9] with ResNet101 [27], while F stands for FCN8s [29] with VGG16 [25] as backbone network.

SYNTHIA → Cityscapes																			
Method	Arch.	road	sidewalk	building	wall	fence	pole	light	sign	veget	sky	person	rider	car	bus	mbike	bike	mIoU
AdaptSegNet [72]	D	79.2	37.2	78.8	-	-	-	9.9	10.5	78.2	80.5	53.5	19.6	67.0	29.5	21.6	31.3	45.9
CLAN [74]	D	81.3	37.0	80.1	-	-	-	16.1	13.7	78.2	81.5	53.4	21.2	73.0	32.9	22.6	30.7	47.8
BDL [76]	D	86.0	46.7	80.3	-	-	-	14.1	11.6	79.2	81.3	54.1	27.9	**73.7**	42.2	25.7	45.3	51.4
Ours	D	**87.7**	**49.7**	**81.6**	-	-	-	**19.3**	**18.5**	**81.1**	**83.7**	**58.7**	**31.8**	73.3	**47.9**	**37.1**	**45.7**	**55.1**
FCNsITW [70]	F	11.5	19.6	30.8	4.4	0.0	20.3	0.1	11.7	42.3	68.7	51.2	3.8	54.0	3.2	0.2	0.6	20.2
Curriculum [71]	F	65.2	26.1	74.9	0.1	0.5	10.7	3.5	3.0	76.1	70.6	47.1	8.2	43.2	20.7	0.7	13.1	29.0
CBST [75]	F	69.6	28.7	69.5	**12.1**	0.1	25.4	11.9	13.6	**82.0**	**81.9**	49.1	14.5	66.0	6.6	3.7	32.4	35.4
DCAN [79]	F	79.9	30.4	70.8	1.6	**0.6**	22.3	6.7	23.0	76.9	73.9	41.9	16.7	61.7	11.5	10.3	38.6	35.4
CLAN [74]	F	80.4	30.7	74.7	-	-	-	1.4	8.0	77.1	79.0	46.5	8.9	73.8	18.2	2.2	9.9	39.3
CrDoCo [80]	F	**84.9**	32.8	**80.1**	4.3	0.4	**29.4**	14.2	21.0	79.2	78.3	50.2	15.9	69.8	23.4	11.0	15.6	38.2
BDL [76]	F	72.0	30.3	74.5	0.1	0.3	24.6	10.2	25.2	80.5	80.0	54.7	**23.2**	72.7	**24.0**	7.5	44.9	39.0
Ours	F	79.1	**34.0**	78.3	0.3	**0.6**	26.7	**15.9**	**29.5**	81.0	81.1	**55.5**	21.9	**77.2**	23.5	**11.8**	**47.5**	**41.5**

Table 4.2: Results of adapting SYNTHIA [88] to Cityscapes [2]. D stands for DeepLabV2 [9] with ResNet101 [27], while F stands for FCN8s [29] with VGG16 [25] as backbone network.

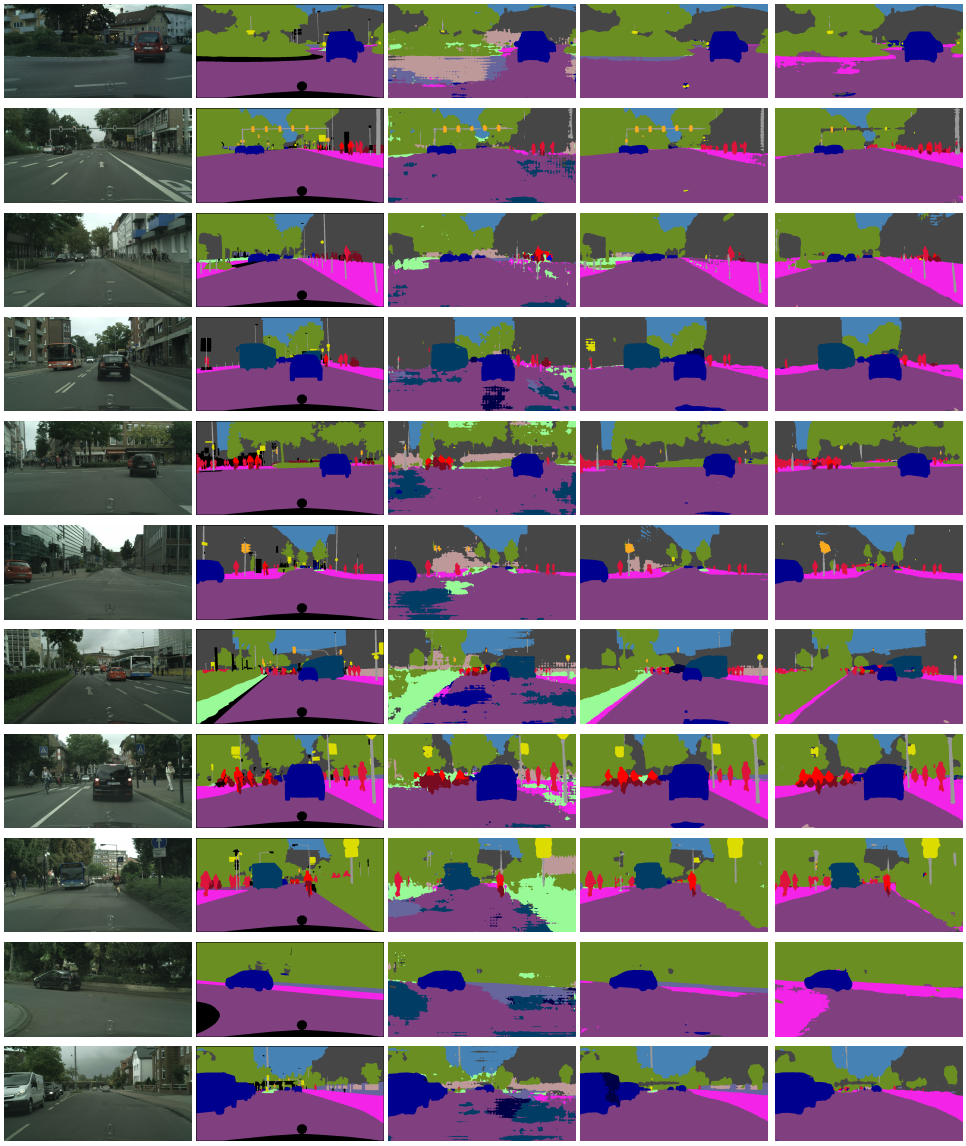


Figure 4.6: **Segmentation results.** We take a sample X_T from the Cityscapes validation set and get the prediction using M. Here we show the different results obtainable with M being DeepLabV2. First we show the results obtained with M trained with no adaptation on GTA5, then the results obtained by adapting GTA5 and SYNTHIA.

SPADE	$\mathscr{L}_{SCE}$	mIoU	Gain	Gap to UB
		49.2	15.6	15.9
✓		49.5	15.9	15.6
	✓	49.5	15.9	15.6
✓	✓	50.4	16.8	14.7

Table 4.3: **Ablation study**. We report the mIoU, the gain *wrt* the lower bound (*i.e.* training naively on source), the gap *wrt* the upper bound (*i.e.* training on target).

SYNTHIA to Cityscapes SYNTHIA [88] has been adopted in the past by the other works for its overlapping with 16 of the Cityscapes classes. For the SYNTHIA to Cityscapes task we compare our results with the state-of-the-art in Table 4.2 and present some visual results in the supplementary material. For a fair comparison, the results of the DeepLabV2 architecture are limited to the 13 classes adopted by the other works [72, 74, 76]. The upper bounds in terms of mIoU are 71.7 for DeepLabV2 and 59.5 for FCN8s. In the case of DeepLabV2 we surpass the current state-of-the-art in mIoU by +3.7%. For FCN8s, instead, we get +2.5% on the mIoU.

4.3.5 Ablation study

In order to weight our contribution, we perform an ablation study of the proposed method (see Table 4.3). For each experiment, we report 3 values: the mIoU; the gain *wrt* the lower bound, which is a naive training on the source dataset; the remaining gap *wrt* the upper bound, which is the result for training with target labels (called oracle prediction).

The experiments are conducted with DeepLabV2 [9] for the GTA5 [87] to Cityscapes [2] task, for which the lower bound is 33.6 and the upper bound is 65.1.

We first show the baseline results that we get by using the generator with no Symmetric Cross-Entropy $\mathscr{L}_{SCE}$ and no semantic guidance. In this setting, the resid-

ual blocks of the generator use IN [21] layers and the image-to-image translation is completely unrelated to the Semantic Segmentation. Secondly, we add the semantic guidance with the SPADE [54] layer. This setting can still benefit from the semantic guidance in the translation, but loses the ability to enforce the cross-domain consistency for the segmentation task. Then we swap back the SPADE layer with IN and enable $\mathscr{L}_{SCE}$. This setting resembles the one used in [80], where the architecture of CycleGAN [55] is replaced by ours. Finally, we show that the best results are achieved by the combination of the two elements, which completely bridges the translation and segmentation tasks and is the final setting of our work.

We can see that when we remove SPADE or $\mathscr{L}_{SCE}$ the mIoU drops, suggesting that they both have an important contribution to get the best result.

4.3.6 Image translation quality

We also report the quality of the images generated by our image-to-image translation model. In Table 4.4 we report the Inception Score (IS) [90] of the images $X_{S\to T}$ and the Fréchet Inception Distance (FID) [94] with the Cityscapes training set. Although the IS of the produced images is low in every setting, the FID results indicate that the semantic guidance induced by DeepLabV2 is the one that best visually aligns the synthetic domain to Cityscapes. The images translated from SYNTHIA, however, have a much greater distance from Cityscapes than the ones translated from GTA5, regardless of the network used as semantic guidance. We note that this is possibly due to the bigger initial gap in visual appearance between the two domains, since the FID between the original SYNTHIA and Cityscapes is 156.92, while the FID between the original GTA5 and Cityscapes is only 62.42.

4.3.7 Fake segmentation

The results of Figure 4.4 and Figure 4.5 show how our image-to-image translation method is able to visually align the synthetic to the real domain. However, it is hard to see what is the contribution given by the Semantically Adaptive Generator (see Section 3.2.5). Therefore, we perform a particular experiment, we take two

Setting	Network	IS	FID
GTA5 → Cityscapes	DeepLabV2	4.9	27.9
GTA5 → Cityscapes	FCN8s	4.8	40.3
SYNTHIA → Cityscapes	DeepLabV2	5.0	100.8
SYNTHIA → Cityscapes	FCN8s	4.9	113.7

Table 4.4: **Image quality evaluation**. We report the Inception Score (IS) [90] and the Fréchet Inception Distance (FID) [94] of the images generated in each setting of our experiments.

different synthetic samples X_S^1 and X_S^2. We then use M to get the predicted segmentation $M(X_S^1)$ and use it as semantic guidance for the translation of X_S^1 to get $X_{S\to T} = F_{S\to T}(X_S^1, M(X_S^2))$. The result emphasizes the effect of the semantic guidance in our image-to-image translation method. This process can be seen in Figure 4.7.

4.4 Panoptic domain adaptation

We perform panoptic domain adaptation using the VIPER [89] to Cityscapes [2] setting. We evaluate the Panoptic Quality (PQ) on the Cityscapes validation set and show how our method is able to improve the baseline model trained naively on VIPER. Therefore we conduct an ablation study to highlight the value of our contributions. See Sections 3.3, 3.5, 3.6 and 3.7 for details about the methods.

4.5 Panoptic network

We adopt Panoptic FPN [43] as Panoptic Segmentation architecture. We employ ResNet50 [27] pretrained on ImageNet [23] as backbone. The training uses the fol-

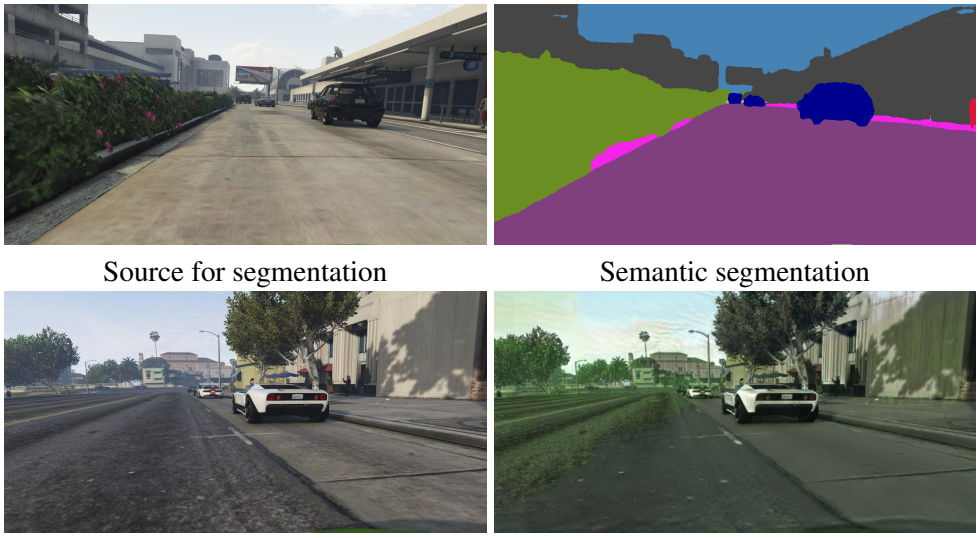


Figure 4.7: **Fake segmentation for image-to-image translation.** We take two different samples (a) and (c) from GTA5. We then use M to get the predicted segmentation (b) which acts as semantic guidance for the translation of another image (incoherent with the semantics). The result (d) emphasizes the effect of the semantic guidance in our image-to-image translation method.

lowing learning rate policy:

$$lr_i = \begin{cases} lr_0 \cdot \frac{i}{\omega_0}, & 0 \leq i < \omega_0 \\ lr_0, & \omega_0 \leq i < \omega_1 \\ \gamma \cdot lr_0, & \omega_1 \leq i < \omega_2 \\ \gamma^2 \cdot lr_0, & \omega_2 \leq i < i_{max} \end{cases} \tag{4.10}$$

In order to stabilize the adversarial training for domain adaptation, we lower the training batch size (32 in the original paper [43]) to 1 and scale all other hyperparameters accordingly by a factor of 32. In particular we downscale the initial learning rate lr_0 to $3.125 \cdot 10^{-4}$, we upscale the maximum number of iterations i_{max} to 2080000 and the learning rate milestones ω_1 to 1280000 and ω_2 to 1760000. We start the training with a warmup phase of $\omega_0 = 1000$ iterations, where the learning rate is linearly increased from 0 to lr_0. Then the training continues with a decay of $\gamma = 0.1$ after two milestones. To ensure that such rescaling of hyperparameters is a good heuristic, we train Panoptic FPN on Cityscapes and obtain PQ=56.77, which is in line with the result using the official hyperparameters (PQ=57.7 [43]).

Input images from both domains are resized to a random resolution that goes from 1024×512 to 4096×2048 pixels, which may possibly break the aspect ratio. After resizing, we take a random crop of 1024×512 pixels from each image. This data augmentation process is fundamental for the detection pipeline to learn object appearings at different scales.

The discriminators employed for adversarial training follow the same policy we adopt for Panoptic FPN, but are set to use $lr_0 = 1.24 \cdot 10^{-4}$.

4.5.1 Ablation study

In Sections 3.3, 3.5, 3.6 and 3.7 we presented various methods that are useful for adversarial training in Panoptic Segmentation. In Table 4.5, we take various combinations of the presented discriminators, where the final adversarial loss is given by a sum of the single ones, and measure the PQ of each result to find the best combination. The results are compared with the baseline, which is a trivial training of Panoptic

D_{det}	D_{mask}	D_{seg}	D_{FPN}	PQ_A	PQ_T	PQ_S	$mIoU_{seg}$	mAP_{det}	mAP_{ins}
Baseline				21.4	20.4	23.9	29.4	13.9	11.0
✓				25.4	21.7	30.0	35.3	15.1	11.8
	✓			24.5	21.3	28.3	34.2	14.6	11.4
		✓		29.5	21.0	36.0	38.8	13.7	11.2
			✓	28.9	**22.6**	35.1	38.6	**15.8**	12.3
✓	✓			27.3	21.1	32.5	37.7	14.7	11.4
		✓	✓	31.3	21.5	38.5	**42.4**	15.1	11.3
✓	✓	✓		29.5	19.7	36.7	40.1	14.3	10.6
	✓	✓	✓	**31.7**	22.5	38.6	41.8	15.6	**12.7**
✓	✓	✓	✓	31.0	21.9	**38.8**	42.0	14.8	12.2
Oracle				60.3	53.7	65.2	76.7	42.7	36.8

Table 4.5: **Ablation study**. We report the PQ for all classes (PQ_A), the PQ for only things classes (PQ_T), the PQ for only stuff classes (PQ_S), the mean Intersection over Union of the Semantic Segmentation ($mIoU_{seg}$), the mean Average Precision of the bounding boxes (mAP_{det}) and the mean Average Precision of the instance masks (mAP_{ins}).

FPN on VIPER, and with the oracle, which is the result obtained by training simply on Cityscapes.

We perform our tests using the Cityscapes validation set where we report the mean PQ of all classes, but also the mean of only things and only stuff. In order to evaluate also the single tasks and see how each configuration performs on them, we also measure the mean Intersection over Union of the semantic maps, the mean Average Precision of the detected boxes and the mean Average Precision of the instance masks. Note that in VIPER the classes *wall* and *pole* are not labelled, while *train* is available but there are no instances and *rider* is labelled as *person*, which is why we ignore these classes when evaluating on Cityscapes.

From this study we can see how discriminators designed for a specific task also aid (or hinder) other tasks. This is why in order to adapt the features for that task, the network consequently modifies the features used for the others. For example, the basic detection discriminator D_{det} improves all scores *w.r.t.* the baseline results.

We can notice that the best combination for the overall PQ is given by adapting Semantic Segmentation, instance masks and FPN features. Using all discriminators actually lowers the final score, but a more careful tuning of hyperparameters and architectures might change this result.

Our best result has a gain of 10.7 PQ *w.r.t.* the baseline, but is still missing 29.3 points to reach the upper bound. In the following chapter we provide some suggestions on how to improve the panoptic domain adaptation with self-training and image-to-image translation, but leave this to future research.

4.6 Discussion

The experiments conducted on UDA for Semantic and Panoptic Segmentation show how perception systems based on deep learning can reach high accuracy in real environments even when no real label is used to train them.

In particular, UDA methods for Semantic Segmentation can be considered mature and are well consolidated into three main categories: self-supervised learning, adversarial training and image-to-image translation. We can see how these can be considered as separate modules once implemented, but their cooperation is fundamental to increase model accuracy (*i.e.* self-supervision if effective only if the annotator model has already a good accuracy, *e.g.* because of adversarial training).

Our experiments on Semantic Segmentation show how this applies to image-to-image translation. By adopting a baseline model as semantic guidance, we are able to develop a new image-to-image translation system, and its output is shown to be particularly useful to retrain the segmentation model and adapt it to the target domain. We have shown how our contribution to the pixel-level adaptation improves the mIoU of the analyzed models and surpasses the state-of-the-art methods.

The pixel-level domain gap is the main reason behind the low target accuracy, which is why we believe that research should focus on the possibility of generating more and more realistic images to be exploited in training. This is also why we designed an image generator that is particularly suited for cooperation with a Semantic Segmentation model.

The same reasoning can be extended to UDA for Panoptic Segmentation, which still lacks a full pipeline since the task introduction in [42]. Therefore we only developed feature-level adaptation through adversarial training, which has shown promising results, but leave the experiments on self-supervision and pixel-level adaptation to future work. In the following we also give future directions on how to carry on this work, and suggest some possible implementations of these ideas.

Chapter 5

Conclusion

In our work we investigated methods to simplify the development of perception systems in autonomous driving. Considering the limits and high cost of annotating real images, we researched on how to train Deep Convolutional Neural Networks on synthetic images (*e.g.* from simulators). Given the difference between the synthetic and real domain, we focused on methods that are able to solve Unsupervised Domain Adaptation, both at pixel level and feature level.

For semantic segmentation, we combined Semantically Adaptive Image-to-image Translation [4] with adversarial feature-level alignment and self-supervision to surpass the state-of-the-art with various architectures and dataset settings.

For panoptic segmentation, we developed adversarial training methods to perform feature-level alignment, which show how to reduce the domain shift for such a complex task.

5.1 Future directions

In the following we propose future directions to further improve this work and to overcome the UDA problem.

We notice that Semantically Adaptive Image-to-image Translation has still to reach its full potential. Current implementation synthesizes different normalization

statistics based on the processed region. However, recent methods of image-to-image translation rely on a clear distinction between style and content [57], where style is modeled as these normalization statistics. Therefore we propose another implementation of SPADE, where per-class normalization statistics are sampled as a per-class style code. This would allow multimodal image-to-image translation, where each sampling would translate each class differently (*e.g.* translate the same image with different kinds of target roads).

Moreover, Panoptic Domain Adaptation could be improved with pixel-level alignment. Just like the Semantically Adaptive method we propose, it could be possible to implement Panoptically Adaptive Image-to-image translation. Besides adaptively denormalizing each region based on its class, the Panoptic Segmentation model could be exploited to apply different translations to different instances. This could be done, for example, by feeding SPADE [54] with an instance boundary mask, but its construction needs to be differentiable.

Finally, Panoptic Domain Adaptation could be improved by creating pseudo-labels for each subtask. We already developed a solution for semantic maps in Section 3.3.2, which can be easily adopted for the panoptic task. We propose to adopt a similar solution for instance masks, but we first need to define which instances are reliable. To this end, we propose to employ SRRS for the bounding boxes of the detection branch and keep as pseudo-labels only those instances with a score higher than a threshold. This should also be employed to select the pseudo-ground-truth for object detection. Using the outputs of the detection branch should provide more reliable bounding boxes than the ones extracted from mask boundaries. With these methods, self-supervision could provide a great boost to Panoptic Domain Adaptation.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[2] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[4] Luigi Musto and Andrea Zinelli. Semantically adaptive image-to-image translation for domain adaptation of semantic segmentation. In *BMVC 2020-British Machine Vision Conference*, 2020.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[6] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016.

[7] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 472–480, 2017.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[10] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[11] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[12] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[13] George Cybenko. Approximation by superpositions of a sigmoidal function. math cont sig syst (mcss) 2:303-314. 2:303–314, 12 1989.

[14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[18] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[19] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[21] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[22] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[30] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[32] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[33] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[34] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[35] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[36] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[37] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[38] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[39] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask scoring r-cnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6409–6418, 2019.

[40] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 9157–9166, 2019.

[41] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact++: Better real-time instance segmentation. *arXiv preprint arXiv:1912.06218*, 2019.

[42] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9404–9413, 2019.

[43] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019.

[44] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[45] Lorenzo Porzi, Samuel Rota Bulo, Aleksander Colovic, and Peter Kontschieder. Seamless scene segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8277–8286, 2019.

[46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[47] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 214–223, 2017.

[48] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[49] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[50] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[51] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Advances in neural information processing systems*, pages 469–477, 2016.

[52] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[53] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[54] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019.

[55] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[56] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *Advances in neural information processing systems*, pages 700–708, 2017.

[57] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 172–189, 2018.

[58] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[59] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.

[60] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6924–6932, 2017.

[61] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1180–1189. JMLR. org, 2015.

[62] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4068–4076, 2015.

[63] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. Learning transferable features with deep adaptation networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 97–105. JMLR. org, 2015.

[64] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[65] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. In *Advances in Neural Information Processing Systems*, pages 136–144, 2016.

[66] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176, 2017.

[67] Qingchao Chen, Yang Liu, Zhaowen Wang, Ian Wassell, and Kevin Chetty. Reweighted adversarial adaptation network for unsupervised domain adaptation.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7976–7985, 2018.

[68] Bo Geng, Dacheng Tao, and Chao Xu. Daml: Domain adaptation metric learning. *IEEE Transactions on Image Processing*, 20(10):2980–2989, 2011.

[69] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision*, pages 443–450. Springer, 2016.

[70] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv preprint arXiv:1612.02649*, 2016.

[71] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2020–2030, 2017.

[72] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7472–7481, 2018.

[73] Yi-Hsin Chen, Wei-Yu Chen, Yu-Ting Chen, Bo-Cheng Tsai, Yu-Chiang Frank Wang, and Min Sun. No more discrimination: Cross city adaptation of road scene segmenters. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1992–2001, 2017.

[74] Yawei Luo, Liang Zheng, Tao Guan, Junqing Yu, and Yi Yang. Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2507–2516, 2019.

[75] Yang Zou, Zhiding Yu, BVK Vijaya Kumar, and Jinsong Wang. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training.

In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 289–305, 2018.

[76] Yunsheng Li, Lu Yuan, and Nuno Vasconcelos. Bidirectional learning for domain adaptation of semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6936–6945, 2019.

[77] Aysegul Dundar, Ming-Yu Liu, Ting-Chun Wang, John Zedlewski, and Jan Kautz. Domain stylization: A strong, simple baseline for synthetic to real image domain adaptation. *arXiv preprint arXiv:1807.09384*, 2018.

[78] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[79] Zuxuan Wu, Xintong Han, Yen-Liang Lin, Mustafa Gokhan Uzunbas, Tom Goldstein, Ser Nam Lim, and Larry S Davis. Dcan: Dual channel-wise alignment networks for unsupervised scene adaptation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 518–534, 2018.

[80] Yun-Chun Chen, Yen-Yu Lin, Ming-Hsuan Yang, and Jia-Bin Huang. Crdoco: Pixel-level domain transfer with cross-domain consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1791–1800, 2019.

[81] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Domain adaptive faster r-cnn for object detection in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3339–3348, 2018.

[82] Seunghyeon Kim, Jaehoon Choi, Taekyung Kim, and Changick Kim. Self-training and adversarial background regularization for unsupervised domain adaptive one-stage object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6092–6101, 2019.

[83] Han-Kai Hsu, Chun-Han Yao, Yi-Hsuan Tsai, Wei-Chih Hung, Hung-Yu Tseng, Maneesh Singh, and Ming-Hsuan Yang. Progressive domain adaptation for object detection. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 749–757, 2020.

[84] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[85] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[87] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016.

[88] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016.

[89] Stephan R Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2213–2222, 2017.

[90] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[91] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[92] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[93] Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

[94] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[95] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[96] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[97] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision*, pages 702–716. Springer, 2016.

[98] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[99] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. Learning from synthetic data: Addressing domain shift for semantic

segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3752–3761, 2018.