POLITECNICO DI TORINO

Master Degree in Data Science and Engineering

Master Thesis

Enabling Unsupervised Domain Adaptation in Semantic Segmentation from Deep to Lightweight Model

Politecnico di Torino

Supervisors

Prof. BARBARA CAPUTO

Dr. ANTONIO TAVERA

Candidate

CLAUDIA CUTTANO

OCTOBER 2022

Abstract

Semantic segmentation is the task of detecting and classifying each pixel of an image with a semantic class (i.e, road, sidewalk, pedestrian, etc). It is a powerful tool for various applications, ranging from robotics to aerial image analysis, but its primary application is in autonomous driving, where recognizing the entire scene is critical to making the best driving decision.

One of the major challenges with semantic segmentation is the shortage of large pixel-by-pixel annotated datasets, which are costly in terms of both human and financial resources. A common approach is to train models using synthetic datasets composed of simulated images and pixel-accurate ground truth labels, and then utilize Domain Adaptation (DA) approaches to bridge the gap between such artificial domains (referred to as source) and the real ones (to which we refer as target).

Deep learning networks are nowadays built on the notion of "infinite resources", resulting in larger and more accurate deep models (in terms of learnable parameters), but at the expense of efficiency. This model complexity affects both the inference time and the hardware requirements. Furthermore, data can be confidential or secret, reason why some companies opt to supply only a pretrained model without data access. With this in mind, we suggest extending existing DA techniques in order to harness the fine-grained information of these complex and deep networks while training efficient and lighter models on new domains.

We present a benchmark in which we examine the ability of three traditional DA settings to transfer knowledge from a Deep to a Lightweight model: (i) Adversarial, (ii) Self-Learning (SSL), and (iii) the combination of Adversarial+SSL.

Given the promising results obtained in the Self-Learning setting, we focus on existing SSL-based DA techniques to further improve the performance of the lightweight network. We extend the Cross-Domain mixing technique proposed in the DAFormer paper by introducing a novel cross-domain mixing via instance selection combined with a dynamic weighted segmentation loss. All of the experiments were carried out by applying the two conventional synthetic-to-real protocols: (i) Synthia to Cityscapes and (ii) GTA5 to Cityscapes.

Acknowledgements

Sin dalla prima lezione con la professoressa Barbara Caputo, ormai quasi più di un anno e mezzo fa, ho desiderato poter fare la tesi con lei. Voglio quindi ringraziare immensamente la mia relatrice per avermi dato questa opportunità.

Un ringraziamento di cuore va ad Antonio Tavera, che mi ha pazientemente aiutata e supportata in tutto il percorso della tesi. Nessun grazie può esprimere tutta la mia gratitudine per la sua disponibilità e gentilezza, e sono certa di poter dire che non avrei potuto avere supervisor migliore.

In questi anni, ho avuto la fortuna di avere accanto persone speciali. Alcune di queste le ho conosciute lungo il percorso, altre mi accompagnano da più tempo, qualcuna di loro da sempre. Sarebbero molte le persone che vorrei ringraziare, ma mi limiterò a qualche nome.

Ad Alessandro, una delle persone più care che ho. In tutti questi anni, le nostre vite si sono sempre mosse parallelamente nella stessa direzione, intersecandosi inaspettatamente in un susseguirsi di eventi. Spero che, qualsiasi sia la direzione che il futuro ci riserverà, troveremo sempre il modo di incontrarci.

Ad Isabella, per avermi regalato leggerezza e tante risate.

A Nicola e Niccolò, per tutti i dilemmi informatici e le confessioni notturne che abbiamo condiviso. Siete i migliori amici nerd che potessi desiderare.

A Gianni, per avermi pazientemente ospitata come "terza quasi-coinquilina" in questi anni, non ti ringrazierò mai abbastanza.

A Francesco, che la vita mi ha dato la fortuna di conoscere. Abbiamo sempre scherzato sul fatto che meriteresti una laurea ad honorem in "Machine Learning" per quanto ti ho riempito la testa di idee, dubbi e ragionamenti in questi anni e, in fondo, molto di questo traguardo lo devo a te.

Ai miei genitori, per la fiducia riposta in me.

Per ultima, ma la più importante, a mia sorella Erika.
Ti ho accompagnata per mano tutta la vita, e ora che sei diventata grande riguardo indietro e mi accorgo che non ho ricordo in cui tu non ci sia.
Da quando ho memoria, ti sei sempre cacciata in qualche guaio e io ho sempre fatto l'impossibile per tirartene fuori. E' sempre stato il tuo sport preferito, farmi arrabbiare, e il mio sport preferito, lasciartelo fare.
Ora che sei grande devi essere forte, ma ricorda sempre che, quando la testa sembra non voler collaborare, ci sarò sempre io a stringerti la mano.
Questo traguardo lo dedico interamente, e completamente, a te.

Table of Contents

Chapter 1

Introduction

One of the most active field of research in Artificial Intelligence is Computer Vision (CV). We can describe Computer Vision as the interdisciplinary field that enables machines to observe and understand the real world, emulating the capabilities of the human visual system with sensors, data and algorithms rather than retinas, optic nerves and a visual cortex.
This artificial vision allows machines to monitor the surrounding environment and eventually take autonomous decisions without human intervention, based on the digital images collected and processed. Given this premise, it is not surprising that Computer Vision attracted the attention of several industries and has been already integrated in many sectors, such as healthcare, manufacturing and robotics. However, between the innumerable applications of CV, self-driving cars is probably the most active one, for what concerns investment and research.

The broad field of Computer Vision can be subdivided in several sub-tasks. The classical problem is *image classification*, where the intent is to identify which

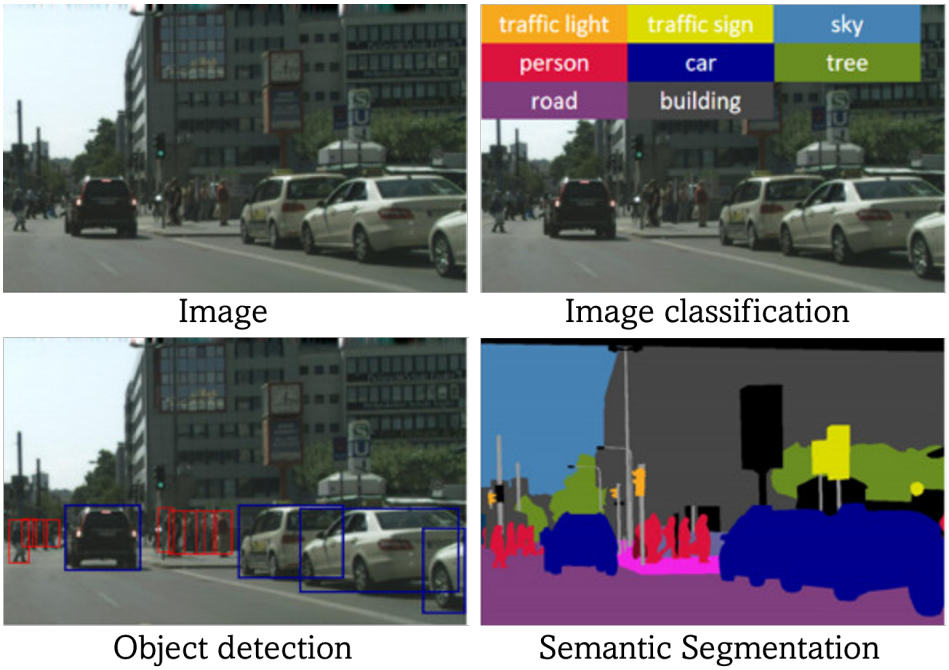


Figure 1.1: Computer Vision can be subdivided in several tasks, including Image Classification (on the top right), Object Detection (on the bottom left) and Semantic Segmentation (on the bottom right). Images borrowed from [1].

objects exist in an image by assigning it one or more category labels. For example, in fig 1.1, the classification algorithm is telling us that several object categories, such as buildings, trees and cars, are depicted in the image.
Object detection takes a step further, identifying not only the presence of certain objects but also detecting their location. In fig 1.1, the detection algorithm locates some objects in the image with annotated rectangles.
The next step is *semantic segmentation*, that consists is partitioning the image space into mutually exclusive subsets, therefore locating the objects but also accurately delineating their boundaries. This task has great potential since makes it possible to obtain fine grained understand of images.

If on one hand the availability of Image Classification datasets with millions of labeled images pushed the improvements in Computer Vision, on the other side Semantic Segmentation task lacked for many time of appropriate annotated datasets.

Clearly, pixel-wise labeling requires a huge human effort, especially in the annotation of object boundaries. For example, the annotation of a single image from the Cityscapes [2] dataset takes up to 90 minutes [3]. For this reason, usually, the more detailed the semantic labeling is, the smaller the size of dataset ("curse of dataset annotation" [4]).
However, 3D games engine have enabled the creation of very large datasets, referred to as *synthetic datasets*, composed of highly realistic images along with pixel-accurate ground-truth labels. Despite being characterized by high fidelity of

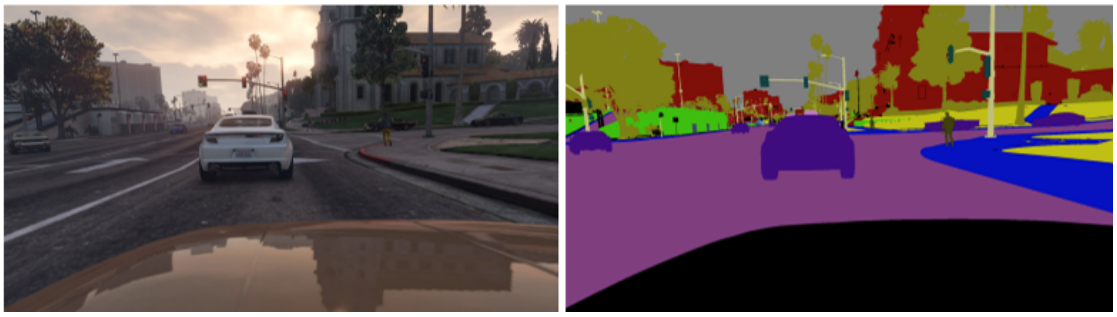


Figure 1.2: On the right, an example of synthetic image from the GTA5 [5] dataset, on the left, its ground truth label.

appearance and detailed environments, synthetic datasets alone cannot be effectively used to tackle real world semantic segmentation tasks, due to the domain shift between the artificial images and the real ones. Intuitively, if we train a model on the artificial images extracted from the GTA5 [5] video game, we cannot expect the model to perform in the same way in a real context, thought the images may be realistic.
To bridge the gap between the different domains, several Domain Adaptation (DA) techniques have been studied and proposed in recent years, making the models more and more able to generalize on target domains characterized by different data distributions. In parallel, more and more complex architectures have been designed, making it possible to obtain remarkable results in Semantic Segmentation.

The combination of these two aspects, combined with the today's high performance of graphic cards and processors, creates a strong starting point for the development of autonomous driving systems based on image segmentation, where data collected from multiple sensors, such as cameras, LiDAR and GPS, are modeled to make driving decisions. Making machines capable to emulate drivers capabilities is not trivial. Actions that from human perspective are immediately recognizable, such as the prediction of vehicle driving or pedestrian trajectory, represent instead a big challenge for autonomous systems.

With the outstanding progresses in Computer Vision, the gap between human drivers and machines is gradually decreasing. However, what makes humans different from machines is not only the capability of observe and recognize multiple situations, but also the *time* necessary to perform this operation (i.e. the inference time).
To understand this concept, we suppose to design a powerful autonomous driving system able to perfectly recognize car surrounding. If a pedestrian suddenly crosses the street and the model takes more than some milliseconds to recognize the presence of the human on the road, the car cannot brake in time. This example explains why the overall performance of the system cannot be simply described by the segmentation precision, but should also take into account the inference time. In other words, to make autonomous systems safely usable in real-world applications, they need to run real-time.
Furthermore, often the segmentation precision is proportional to the complexity of the model. The computing resources needed inside the vehicle might need manufacturers to make special modifications to the car to make it compatible with the hardware requirements, increasing the design and production costs. Reducing the complexity of the models could potentially broad the application conditions, enabling the diffusion also on systems with limited computational resources. This reasoning highly resembles the logic of IoT, where the implementation of lightweight models on edge devices makes it possible to rapidly process data directly on low-cost boards.
The necessity for faster and lighter networks led researchers to design new architectures, characterized by limited complexity and latency.
In this work, we aim at training while transferring knowledge from a deep Semantic Segmentation model to a lightweight network, characterized by limited complexity and inference time, balancing the trade-off between number of learnable parameters, latency and performances.

We start from the benchmarking of existing Domain Adaptation techniques, traditionally applied on top of powerful but slow Deep Learning networks, and investigate how to extend these algorithms to transfer knowledge and train our lightweight model (see Chapter 3).
A straightforward approach consists in directly adopt the aforementioned Domain Adaptation techniques to train the lighter model. However, we imagine a different scenario, especially from companies point of view, in which in phase of deployment we want to use the lightweight model, but, during training, we actually have access to a complex but slower network, which guarantees high performances in the semantic segmentation task. In this scenario, we also start imagining the possibility, for companies, of having at disposal a pretrained deep network, but without having access to the original source data used in phase of training, for various reasons

which range from costs to privacy constraints.

We suggest to exploit the fine grained knowledge of the deep network to train the lighter model, reducing as much as possible the lightweight network dependence from the source dataset. In particular, we propose a benchmark in which we test the capability of the existing Domain Adaptation techniques (i.e. Adversarial, Self-Learning and Adversarial+Self-Learning) to transfer knowledge from a Deep to a Lightweight model. This set of experiments demonstrates, on the one hand, the unimpressive outcomes of the Adversarial technique, and, on the other, the potential of the Self-Learning settings.

Given that, we further improve the performances of our lightweight network by means of a new Instance-based Cross-Domain mixing strategy and by re-weighting the loss on the basis of the per-class confidence of the lighter network step after step (see Chapter 4).
All of the experiments were carried out by applying the two conventional synthetic-to-real protocols: GTA5 $\rightarrow$ Cityscapes and SYNTHIA $\rightarrow$ Cityscapes.

Chapter 2

Related works

2.1 Artificial Intelligence, Machine Learning & Deep Learning

Alan Turing, in its paper "Computing Machinery and Intelligence" [6] (1950), posed a simple question:

> "Can machines think?"

Naturally, even if most people still think at sentient robots when hearing the terms *Artificial Intelligence*, machines cannot consciously *think*, but could be programmed to *replicate* human intelligence. As a result, we could obtain machines able to learn with experience and perform human-like tasks, improving the quality of our lives on several fronts.
For instance, AI has many applications in the healthcare sector, where it can be used to aid in surgical procedures or to dose drugs. In the self-driving industry, AI utilizes data collected from sensors, radars and cameras to drive vehicles to destinations, preventing collisions. Also the financial industry makes use of Artificial Intelligence, for example to spot either anomalies or longer-term trends that otherwise would not have been detected by standard reporting methods.

AI is an interdisciplinary science consisting of multiple approaches, but the greatest achievements have been made in the fields of Machine Learning and Deep Learning.

In 1959, the American pioneer in the field of computer gaming and artificial intelligence Arthur Samuel defined the field of Machine Learning (ML) as:

> "the field of study that gives computers the ability to learn without being explicitly programmed."

In particular, instead of using hand-coded set of instructions to accomplish a specific task, Machine Learning combines statistics with experience to train algorithms able to draw inference from data. In an iterative process, large volume of structured data, known as *training data*, are fed to the ML models, that are automatically trained by optimizing an objective function, which varies on the algorithm and the task. At the end of the learning process, known as *training*, the model can be used in inference on new unseen data, based on the knowledge extracted from the training examples.

Deep Learning is a sub-field of Machine Learning that uses Artificial Neural Networks, complex algorithms inspired by the structure of the brain.
With respect to traditional Machine Learning methods, Deep Learning algorithms are highly scalable, that implies that their performance keep increasing if we increase the training data in input. This scalability, enhanced by the availability of large amount of data and the increasing computational power of computers, has made Deep Learning a revolutionary player in the area of Artificial Intelligence.

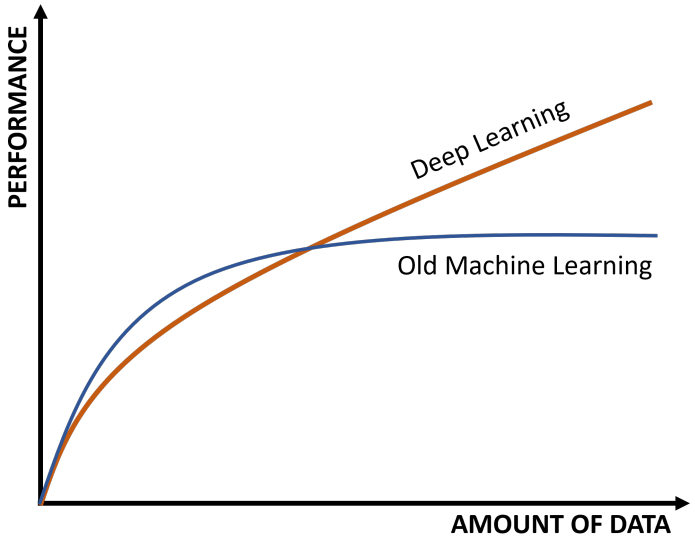


Figure 2.1: When the amount of training data is small, traditional Machine Learning algorithms equal (or even outperform) Deep Learning models. On the contrary, when the amount of data increases, old ML models performances start saturating until they reach a plateau, while DL models performances keep increasing.

2.2 Neural Networks

In the human brain, the information received by our senses is transmitted to the neurons and here processed. These neurons are interconnected each others in a

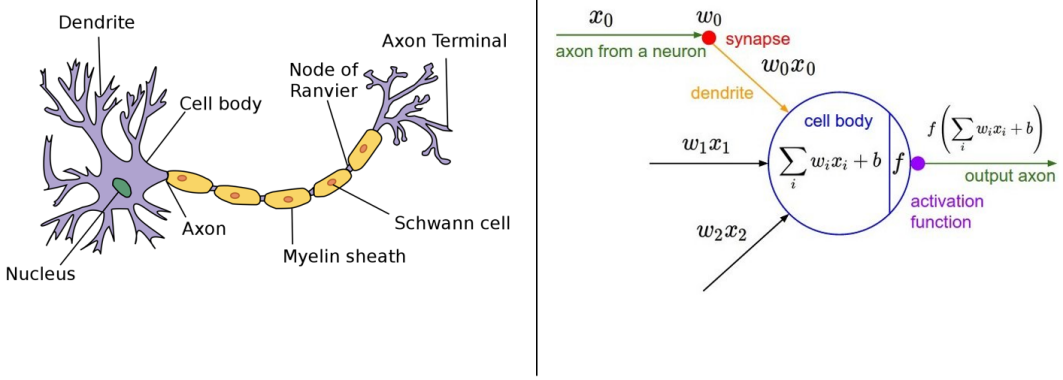


Figure 2.2: The structure of a biological and artificial neuron, respectively on the left and on the right.
A neuron receives signals from the other neurons thanks to special branches called dendrites. The cell body contains the nucleus and controls the cell's activity. Through the axon, the neuron sends out spikes of electrical activity, than can inhibit or excite the activity of a connected neuron.

complex interconnection network, that allows the brain to perform highly complex computations.
Artificial Neural Networks mimic human brain structure, with neurons and links that are artificially created on a computer. Specifically, the neural network structure can be modelled as a directed graph, where neurons are the nodes and the edges represent the links between them.
Since Neural Network are *multi-layer perceptrons*, it is worth starting with the concept of *perceptron*.

2.2.1 Perceptron

A perceptron is a binary classifier that takes several inputs $x_1, x_2, ..., x_n$ and produces a single output value.
In particular, to weight the importance of each input, a set of parameters, called *weights*, is introduced. Each input x_i is multiplied by the corresponding weight

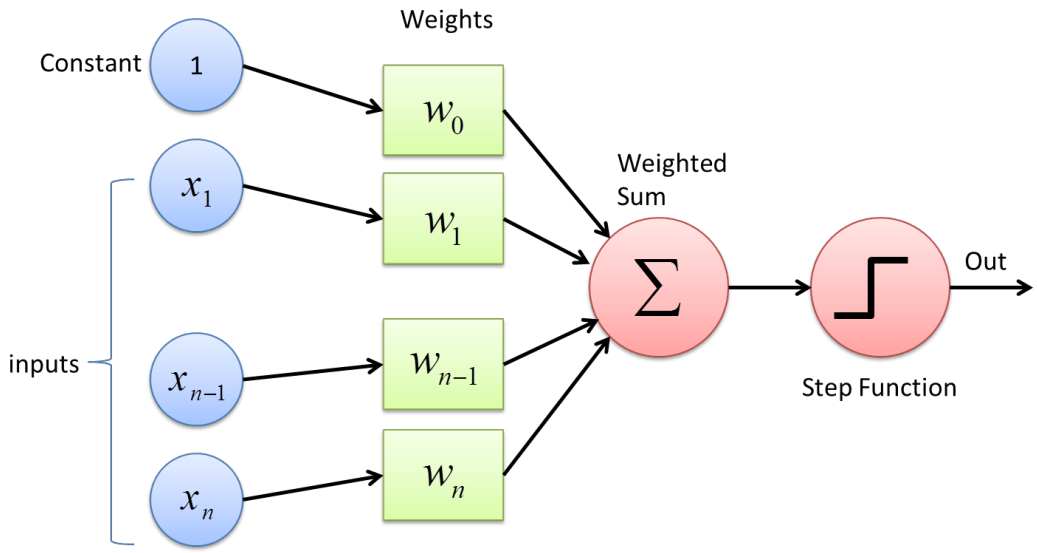


Figure 2.3: Graphical structure of a perceptron. Inputs are multiplied by a set of parameters, called weights, and then summed up. Through the step function, the output is reduced to a binary value, that can be either 0 or 1.

w_i, to obtain the weighted sum u:

$$u = \sum_{i=1}^{n} w_i * x_i$$

Therefore, the weights are real numbers that quantify the importance of the respective input to the output.
To determine the final output, the weighted sum is compared with a threshold value: if u is less than the threshold the output is set to 0, otherwise is set equal 1.

$$output = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} w_i * x_i > threshold \\ 1 & \text{if } \sum_{i=1}^{n} w_i * x_i < threshold \end{cases}$$

Therefore, it is possible to modify the decision making function by varying the threshold value.
Introducing another parameter, the *bias* b = - *threshold*, and replacing the weighted sum formulation with the dot product between the weight and input vectors, the perceptron rule can be replaced with the more lightweight:

$$output = \begin{cases} 0 & \text{if } w_i * x_i + b > 0 \\ 1 & \text{if } w_i * x_i + b < 0 \end{cases}$$

The perceptron architecture represents a **single-layer neural network** and is particularly useful when the data are linearly separable. However, in real-case scenarios, data are usually non-linearly separable, making it necessary to have more complex structures characterized by:

- Multiple layers, consisting of several perceptrons
- Non linear activation functions

2.2.2 Multi-Layer Perceptrons

Multi-layer percetrons are architectures composed of one or more layers, where each layer is obtained by stacking several perceptrons on each other.
As we can see in fig. 2.4, MLP are composed of 3 parts:

- The input layer: composed of input neurons, takes raw data in input without performing any operation.
- The hidden layer: the intermediate layer(s), not exposed on the outside.
- The output layer: the fixed length output vector of the network.

For example, in fig. 2.4 the number of layers is 4, since the network contains 2 hidden layers.
In this architecture, we feed the input into the network, and the intermediate

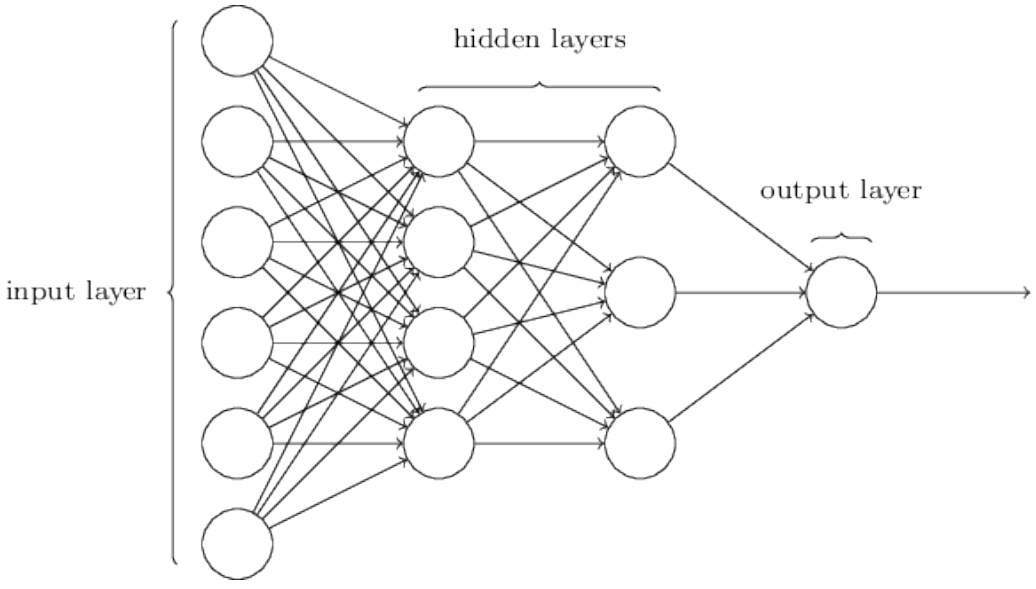


Figure 2.4: Graphical structure of a Multi-Layer Perceptron.

result is computed layer by layer, until it reaches the output layer. The information always flows forward, without any loop in the network. For this reason, it is referred to as *Feedforward Neural Network*.

2.2.3 Non linear Activation functions

In the perceptron structure presented in 2.2.2, the activation function adopted is the so-called Heaviside step function. This function simply returns 0 and 1 for negative and positive inputs, respectively:

$$f(x) = \begin{cases} 0 & \text{if } x > 0 \\ 1 & \text{if } x < 0 \end{cases}$$

Despite working reasonably well for linearly separable data, this function has several drawbacks:

- To obtain high quality predictions, a small change in the weights of the network should result in a small variation of the output. This small change in the output is not achievable with a function that can only assume 2 values.
- Neural Network are trained using gradient descent with backpropagation, which requires differentiable activation functions. Heaviside step function is non differentiable at $x = 0$, while the derivative in all the other points is equal to 0, therefore blocking the learning process.

To face these issues, one could suggest to simply use the *identity* (or *linear*) function as activation:

$$f(x) = x$$

In fact, adopting the linear function, a small change in weights would correspond in a small variation in the output.
However, despite being differentiable in all its points, it cannot be effectively used in the backpropagation algorithm, because its derivative is a constant, independently from the input. In addition, it can be proven that a multi-layer perceptron containing linear activation function in all neurons can be reduced to a 2-layer input-output model, whatever was its original depth.
Therefore, to enhance the representational power of neural networks, it is crucial to insert *non linear activation functions*.

Sigmoid Function The Sigmoid Function, often referred to as *Logistic Function*, takes any real value in input and produces an output value in the range [0,1].

Figure 2.5: Sigmoid or Logistic

Mathematically, the Sigmoid Function implements:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

This function is historically popular, especially because it can be intuitively used to represent probabilities. However, it has some drawbacks.
As we can see inf fig. 2.5, when the input value x is very small or really big the Sigmoid Function will output values close to 1: in these regions, the gradient will tend to 0. This implies that even a large modification in the parameters cannot change the output much, making the training process slower and slower. This phenomenon is referred to as *vanishing gradient* problem.
Other minor problems are the fact that sigmoid outputs are not zero-centered and that the exponential computation is an expensive operation.

Tanh Function The Tanh Function (or *hyperbolic* function) solves the not-zero centered problem of the Sigmoid Function by producing output values in the range $[-1,1]$:

$$tanh(x) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Having a zero-centered range leads to faster convergence and allows the network to better model inputs with strongly negative, positive or neutral values.
Despite the different span range, its curve is very similar to the one characterizing

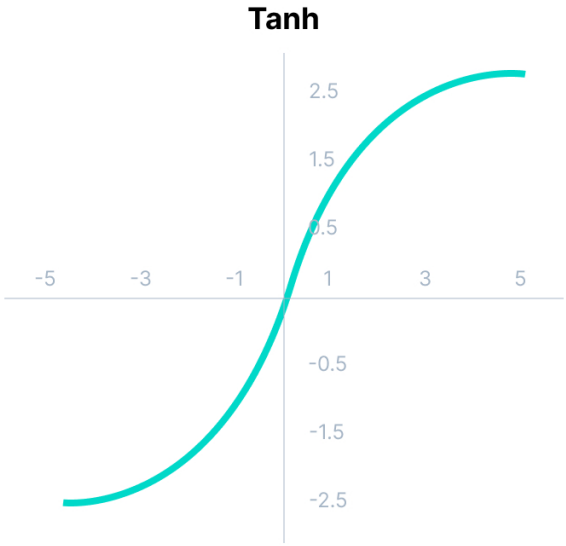


Figure 2.6: Tanh or Hyperbolic

the Sigmoid function. As a consequence, also the hyperbolic activation suffers from the vanishing gradient problem, as well as the computational complexity of the exponential function.

ReLU ReLU (*Rectified Linear Unit*) are the most common activation functions used in neural networks. They return the same value received in input, except when the values are negative. In this cases, they return 0.
Mathematically:

$$f(x) = max(0, x)$$

Despite their simplicity, they are very effective.
First of all, since only a portion of neurons are activated at a time, they are extremely computationally efficient, especially when compared with the exponential functions previously analyzed.
Then, they solve the vanishing gradient problem, because their output is proportional to the node activation, for positive inputs.
In addition, given their non saturating property, ReLU accelerate the gradient descent toward the global minimum of the loss function, therefore speeding up the convergence.

Despite these advantages, ReLU has an important limitation. As we can see in fig. 2.7, since the negative inputs are mapped to 0, the derivative of this function on the negative part of the graph is 0. This implies that, during the backpropagation,

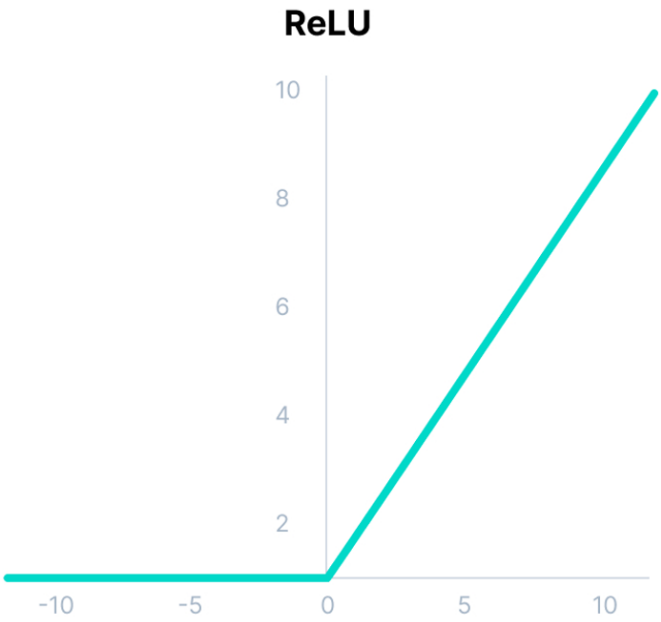


Figure 2.7: Rectified Linear Unit (ReLU)

some neurons are never activated, therefore their weights never updated, causing the existence of *dead neurons*. This phenomenon is referred to as *Dying ReLU problem*.

Leaky ReLU Leaky ReLU solve the dying ReLU problem by inserting a small positive slope in the negative region of the graph.
Mathematically, Leaky ReLU implement:

$$f(x) = max(0.01 * x, x)$$

Besides the advantages of the ReLU, this function enables the gradient backpropagation also for the negative values, eliminating the existence of dead neurons.

Parametric Leaky ReLU In the Leaky ReLU activation function, the small value determining the slope for the negative region is fixed. This has shown a negligible increase in the accuracy of neural networks.
Parametric Leaky ReLU introduce the parameter α to determine the slope of the activation function in the negative part of the graph:

$$f(x) = max(\alpha * x, x)$$

This parameter is learnt during the backpropagation at a negligible increase in the cost of training.

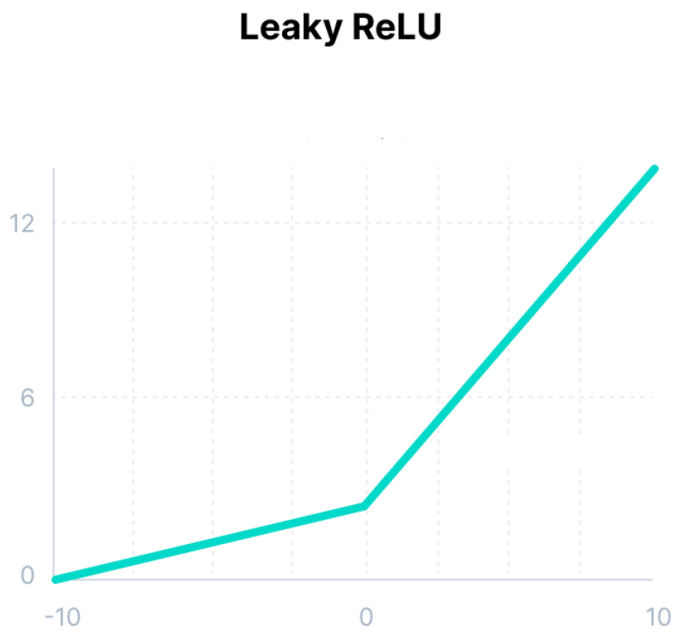


Figure 2.8: Leaky ReLU

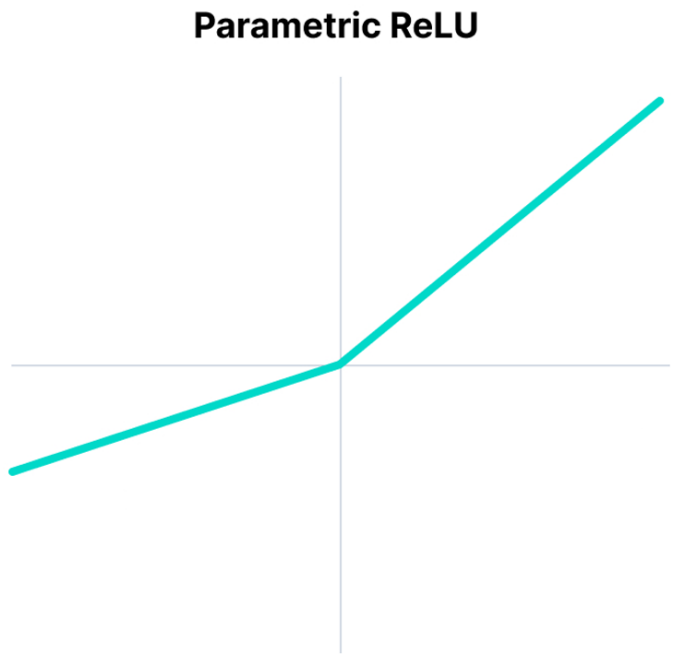


Figure 2.9: Parametric Leaky ReLU

2.2.4 Gradient Descent Algorithm

Now, we describe how the training process works, more specifically how the weights of neural networks are updated iteration by iteration.

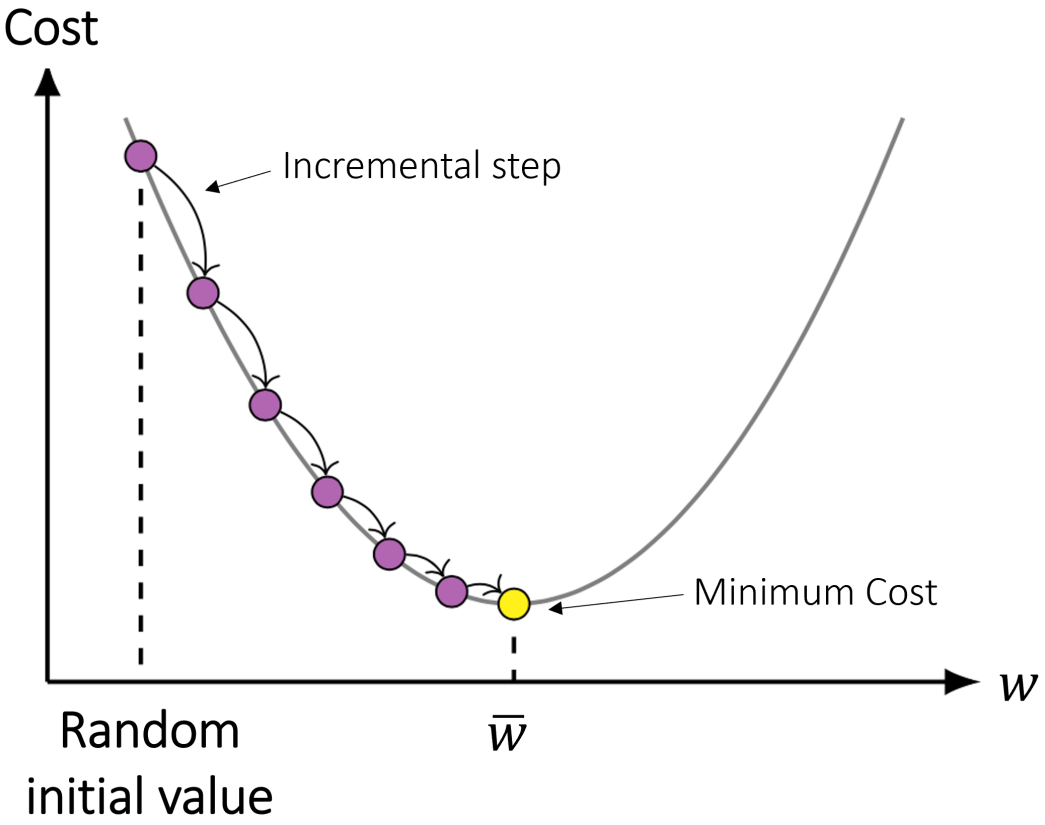


Figure 2.10: The cost function measures how bad the network is modeling the relationship between the input and the output. Gradient Descent can be imagined as the direction to reach the minimum error: at each step the set of weights moves towards the minimum point, making jumps proportional to a parameter called *learning rate*.

Typically, when a neural network is designed, its weights are randomly initialized. We can then feed our input to the network and obtain a prediction, in the so-called *forward propagation* process. Clearly, initially, we cannot expect high quality results, but we want to update iteratively the network until it converges to a optimal configuration able to output satisfying predictions.
Therefore, we develop a *cost function* (often referred to as *loss*), that penalizes

outputs far from the expected values. In particular, since each output can be represented as a composition of functions, the cost function can be directly expressed as a function of the weights of the network. As a consequence, it is possible to optimize the weights by minimizing the cost function, exploiting their relationship. More specifically, we can compute the vector of partial derivatives of the cost function $f(w)$ with respect to each learnable parameter $\nabla f(w) = \left(\frac{\partial f(w)}{\partial w[1]}, ..., \frac{\partial f(w)}{\partial w[d]}\right)$ and apply the so-called *gradient descent* algorithm to move toward the global minimum of the loss function by backpropagating those gradients.
In particular, we exploit the recursive application of the chain rule in the backpropagation of the error, which enables the efficient calculation of the derivatives of composed functions by exploiting the derivative of the parent function.
Mathematically, at each iteration we take a step in the direction of the negative of the gradient at the current point:

$$w^{(t+1)} = w^{(t)} - \mu \nabla f(w^{(t)})$$

where μ is the learning rate parameter, controlling the step size.

2.3 Convolutional Neural Networks

Great advances are being made in the field of Computer Vision, especially thanks to the introduction of special Neural Network called *Convolutional Neural Networks* (CNN).
These family of architectures are designed to process data having grid pattern (i.e. images) to perform several tasks, such as Image Analysis & Classification, Video Recognition or Image Segmentation.
At this point, one could ask why we cannot simply apply to images the multi-layer perceptrons architecture previously discussed. To address this concept, we should analyze the structure of an RGB image (see fig. 2.11).
An image can be visualized as a matrix of values, each representing a pixel. Taking as example a image with size $200 \times 200 \times 3$, a single fully-connected neuron would require $200 * 200 * 3 = 120.000$ weights. Considering a MLP with 40.000 hidden units, the number of parameters would increase up to ≈ 5 bilion parameters.
Besides being computationally expensive, the huge number of weights results in overfitting, making the network completely unable to generalize. In addition, this fully connectivity does not allow the network to capture the spatial dependencies and correlations between pixels.

Convolutional Neural Networks rearrange the internal structure of the neurons to take advantage of the images input structure (see fig. 2.12). In particular, neurons are arranged in 3 dimensions, so that each layer of the network receives a

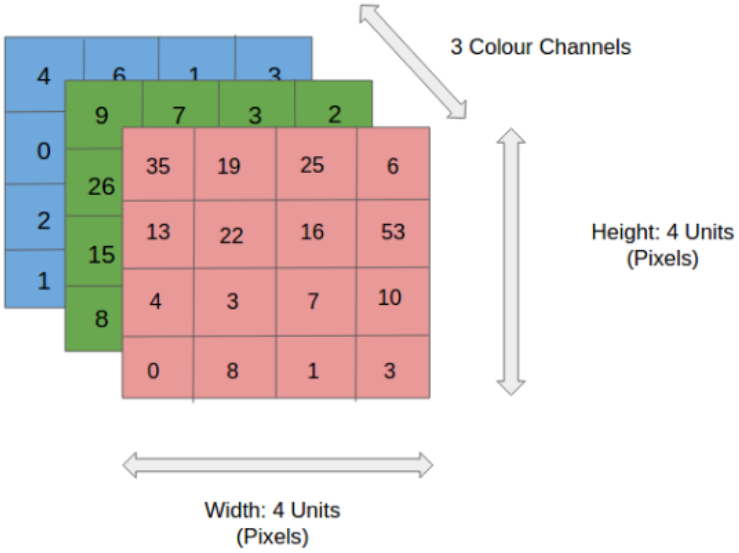


Figure 2.11: Matrix structure of an RGB image.

3D input and outputs a 3D output volume of neuron activations.

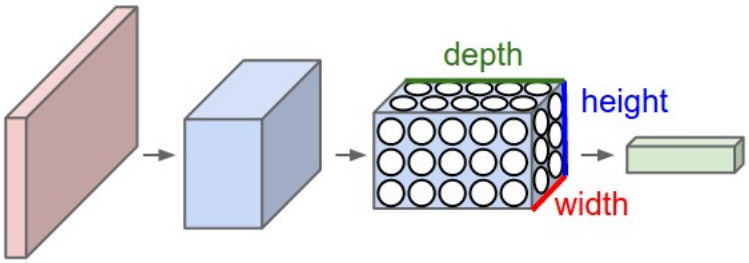


Figure 2.12: 3-Dimensions structure of neurons of Convolutional Neural Networks.

More specifically, the layers composing CNN are essentially of 3 types: Convolutional Layers, Pooling Layers and Fully Connected Layers.

2.3.1 Convolutional Layer

Convolution operation consists in sliding (or, more precisely, *convolve*) a small filter (referred to as **kernel**) across the input image, computing an element-wise dot product between the filter and the input area of the image and then summing the result to produce the corresponding output value in the so-called **feature map**. This process can be repeated by applying an arbitrary number of kernels, each one producing a different feature map, as depicted in fig. 2.14. In this sense, each

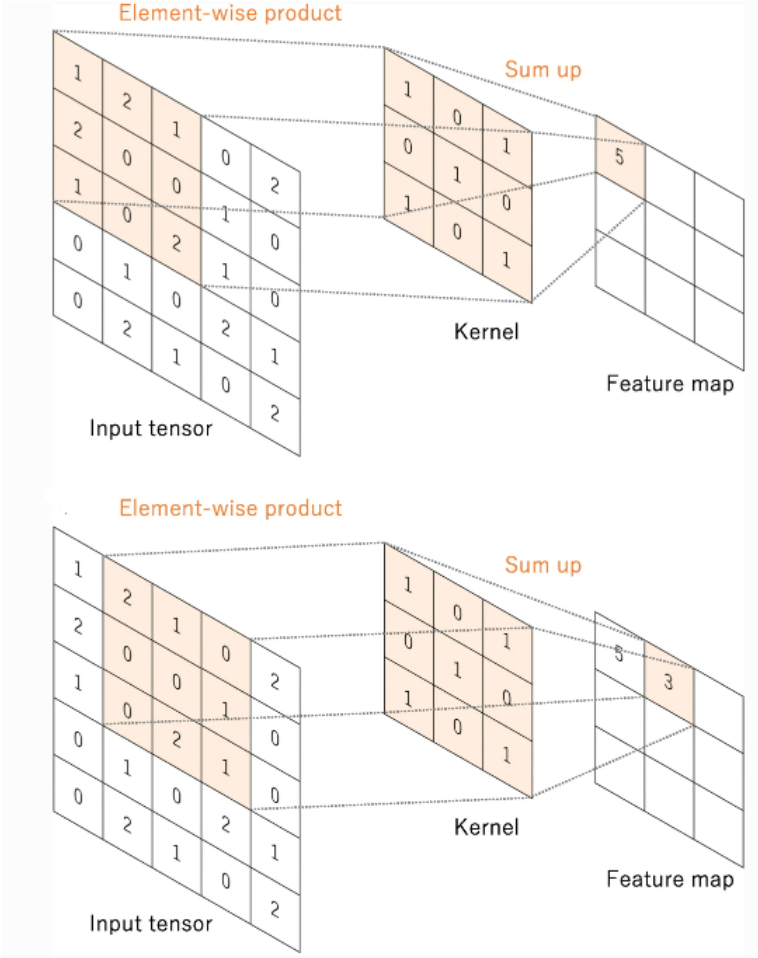


Figure 2.13: An example of 3×3 convolution. The same kernel slides across the input tensor: elements are multiplied pixel-wise and then summed to obtain, at each location, the resulting value of the output feature map. Image borrowed from [7].

kernel can be understood as a feature extractor, looking for a particular feature in the image.
The first layer of the CNN extracts low-level features, such as edges, lines and colors. Going deeper in the networks, convolutional layers are able to capture features of higher and higher level, gradually recognizing bigger shapes and eventually objects.

This weight sharing approach has several advantages, that we can illustrate through some examples.
First, with respect to the fully connected structure, the number of weights are far

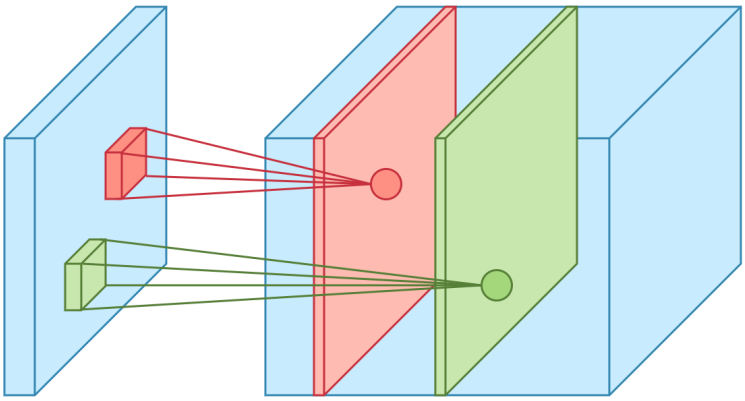


Figure 2.14: Different learnable kernels produce different feature maps.

fewer. Taking the same example as before, we consider a CNN with 40.000 hidden units and we apply a 10×10 filter to the same $200 \times 200 \times 3$ image. The total number of parameters drastically decreases from 5 billions to 12 millions.
Another advantage is that we exploit the fact that statistics, in different locations, are similar. This hypothesis allows the network to successfully use the same feature extractor in different parts of the input. We take as example a kernel detecting edges. If this edge detector is useful in a certain part of the image, it will be probably useful also in another part of the image.
Last point, convolutional neural networks benefit of translation invariance. This means that if we feed an image containing a car, and another image containing the same car but shifted of some pixels, the output feature maps won't be so different. This happens because kernels travel across the entire image and this operation is repeated in all the layers, making the neural network more robust to small shifts.

2.3.2 Pooling Layer

In the Pooling Layers the spatial size of the feature maps is reduced by performing a downsampling operation. Since the output at each location is replaced by deriving a summary of the nearby values, this layer has no learnable parameters.
This operation is essential to enhance the translation invariance to shifts and distorsions of the network and, at the same time, to limit overfitting.
The most used pooling operation is called **Max Pooling**. It extracts the maximum value from each patch, and discard all the other values.
In the commonly used 2x2 filter with stride = 2, 75% of the activations are discarded, width and height are halved, while the number of channels remain unchanged.
Examples of pooling operations are described in fig. 2.15.

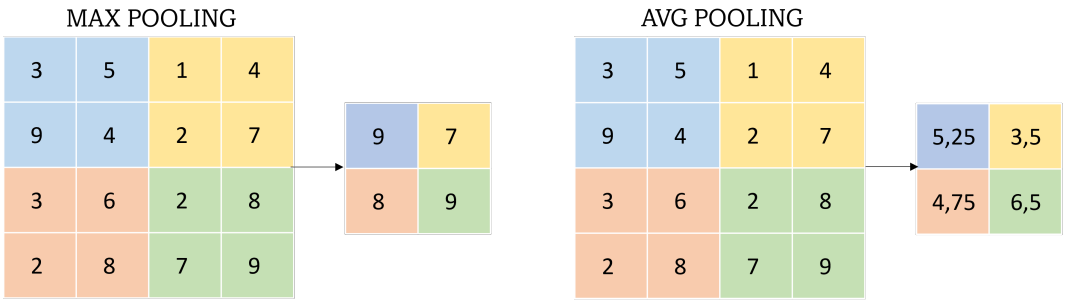


Figure 2.15: On the left, example of Max-Pooling layer with 2×2 filter: the mximum value is extracted from each patch. On the right, an example of Average Pooling layer with 2×2 filter: each patch is replaced with the average of its values.

2.3.3 Fully Connected Layers

After the last convolutional or pooling layer, feature maps are usually flattened and connected to a Fully Connected Layer (e.g. *dense layer*), to produce the final output of the network. Therefore, similarly to standard Neural Networks, each neuron of the Fully Connected Layer is connected, by a learnable weight, to all the activations in the previous layer.
Usually, the activation function of this last layer is different from the others, according to the task of the network. For example, if the network is trained on a multi-class classification problem, an appropriate activation is the softmax function, which normalize the output values to obtain probabilities.

Putting together the pieces, we obtain the overall architecture of a Convolutional Neural Network, as shown in fig. 2.16.

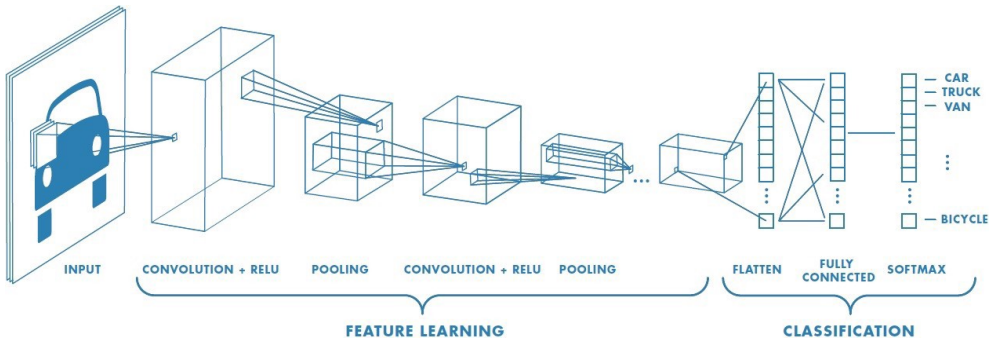


Figure 2.16: Convolutional Neural Network.

2.4 Semantic segmentation

Given a predefined set of classes, semantic segmentation aims at correctly assigning each pixel of an image to its category label, by partitioning the visual space into mutually exclusive subsets. From another perspective, the semantic segmentation task can be conceived as a **classification problem at pixel level**.
Given this fine grained understanding of images, this task has many real world applications, such as self-driving vehicles, medical image diagnosis, aerial image processing and agriculture precision.

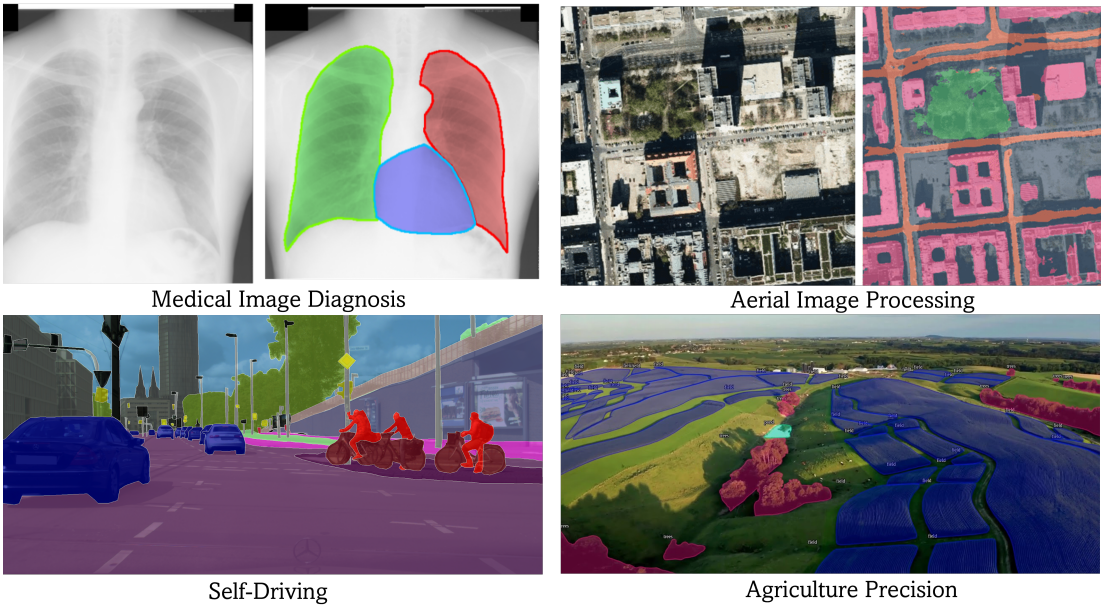


Figure 2.17: Real-world applications of Semantic Segmentation.

With the growing popularity of Deep Learning, several architecture have been proposed to deal with this powerful but challenging task.
The first attempts consisted in simply adapting existing classification networks (such as VGG [8] and AlexNet [9]) by fine-tuning the fully connected layers. However, the use of fully connected layers makes the training phase very time consuming, and cause the model to overfit on the training data. In addition, these traditional CNN networks fail in combining the semantic-aware features extracted from the deeper layers with the spatial details elaborated by the shallow layers.
To solve these issues, researchers proposed a new architecture, shown in fig. 2.18, named Fully Convolutional Networks (FCN) [10], that replaces the fully connected layers with convolutional layers. In particular, given not-fixed sized images in input,

they produce segmented outputs having the same size of the input.
Since the network produces progressively decreasing feature maps, this method

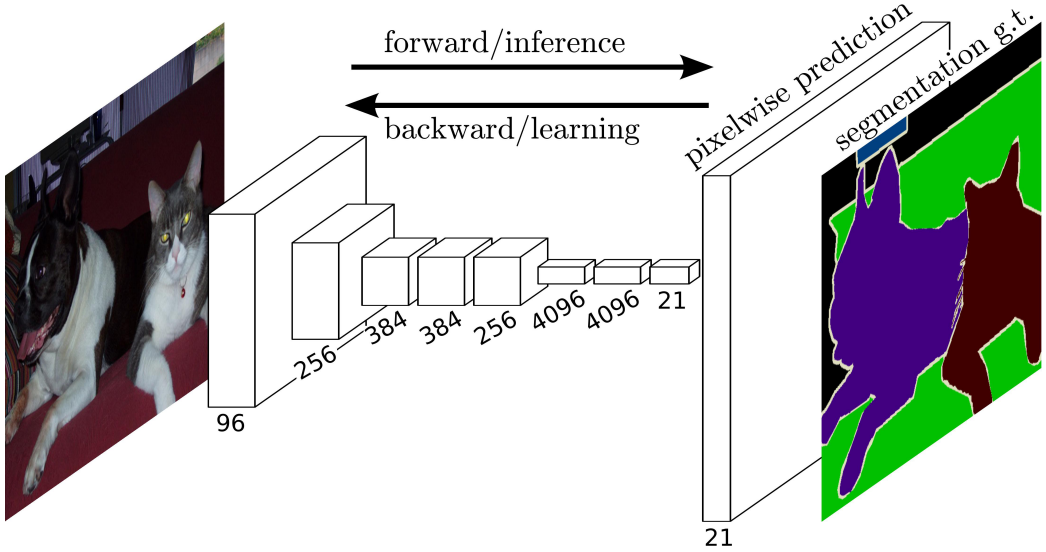


Figure 2.18: Architecture of Fully Convolutional Networks [10].

requires an upsampling strategy to produce outputs with the same size of the input, called deconvolution. In addition, to enhance the fusion of features extracted from different layers, skip connections are introduced in the architecture. These connections are able to bypass the pooling layers and to forward the localized informations extracted from the first layers into to deeper layers of the network, preserving precious details especially for the segmentation of the image at border objects.
This approach evolved in the encoder-decoder architecture. The encoder (or contraction part) uses convolutions to compress the information content of an input image, which is gradually recovered by the decoder (or expansion part), that upscales the encoded features back to the original resolution. The pioneer of this study is Ronneberger, who proposed U-NET [11], shown in fig. 2.19. In this network, each layer of the decoder is directly linked, through the skip connection, to the same layer in the encoder part, resulting in a symmetrical architecture, which guarantees precise localization.
DeconvNet [12] and the corresponding lightweight SegNet [13] propose a new strategy, as shown in fig. 2.20, to upsample the low resolution encoded input in the decoder part. In particular, they perform non-linear upsampling by using the pooling indices (i.e. the location of maximum values) computed and stored in the corresponding encoder part, therefore eliminating the upsampling learning.
Another family of models, called Pyramid Network Based Models, take advantage

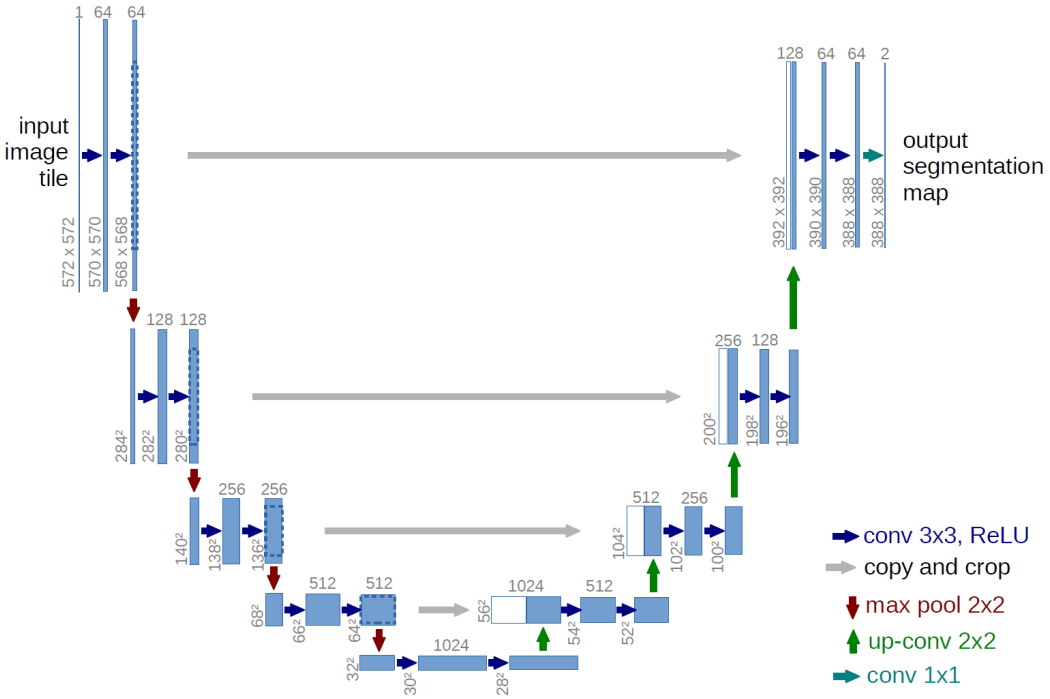


Figure 2.19: Architecture of U-NET [11].

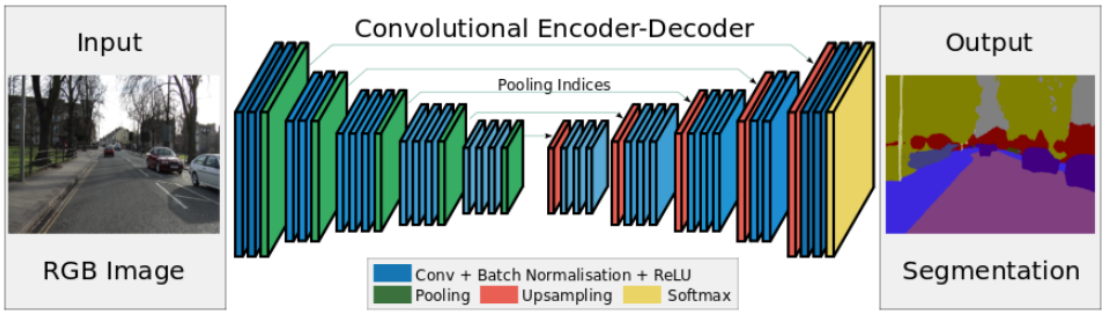


Figure 2.20: Architecture of SegNet [13]

of multi-scale analysis to deal with the challenging task of classifying objects having different scale, in particular small objects. Feature Pyramid Network (FPN) [14] combines the usual bottom-up convolutional network with a top-down pathway and lateral connections. The top-down pathway is used to convert the semantic rich layer into high resolution layers, while the lateral connections helps locating the reconstructed objects in the space (see fig. 2.21).
The advanced Pyramid Scene Parsing Network (PSPN) [15], in fig. 2.22, captures

both local and global context information. It feeds the feature maps extracted from a residual network to a pyramid pooling module to distinguish input at different scales, by pooling them at 4 different scales. To retain both global and local context, the resulting feature maps are upsampled and concatenated to the original feature maps, before generating the pixel-wise prediction.

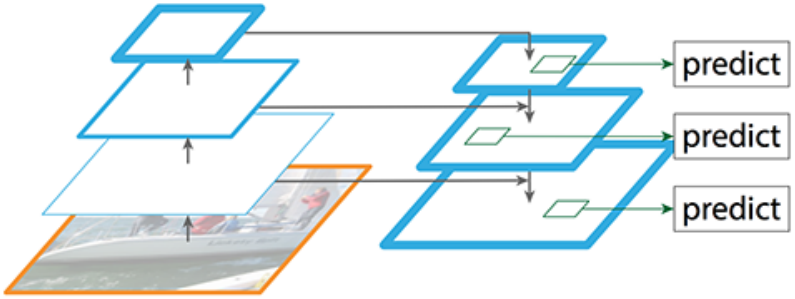


Figure 2.21: Architecture of Feature Pyramid Network [14]

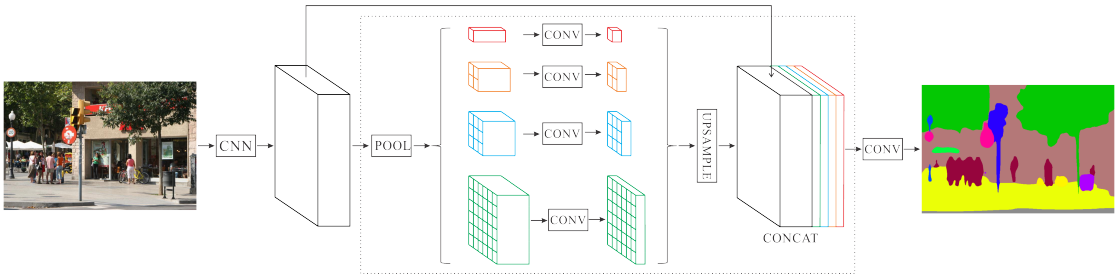


Figure 2.22: Architecture of advanced Pyramid Scene Parsing Network [15]

To enlarge the receptive field of convolutional filters, the Dilated Convolutional Models introduce a new parameter into their filters: the dilatation rate. This parameter allows the network to face the decreasing resolution of the input caused by max pooling and striding, without impacting on the computational cost. In this set of architectures, some of the most important are the DeepLab family [16], densely connected Atrous Spatial Pyramid Pooling (DenseASPP) [17], multi scale context aggregation [18] and the efficient neural network ENet [19].

2.4.1 Architectures

2.4.1.1 DeepLabV2

DeepLab [16] is a semantic segmentation model, designed and open-sourced by Google in 2016. Since then, this family of models has constantly evolved, and DeepLabV2, DeepLabV3 [20] and DeepLabV3+ [21] have been released. Despite being now far from state-of-the-art performance, DeepLabV2 represented a turning point in semantic segmentation, reason why it has been adopted in this work.

The key characteristic of this family of models is the introduction of the dilation rate parameter in their filters, the so-called **atrous convolution**.
Before going into the details of atrous convolution, we should make a digression about the concept of *receptive field*. Receptive field is defined as the size of input region of the feature map that produces each output element.
For example, in fig. 2.23 the receptive field for the layer 2 is 3x3, as each element in

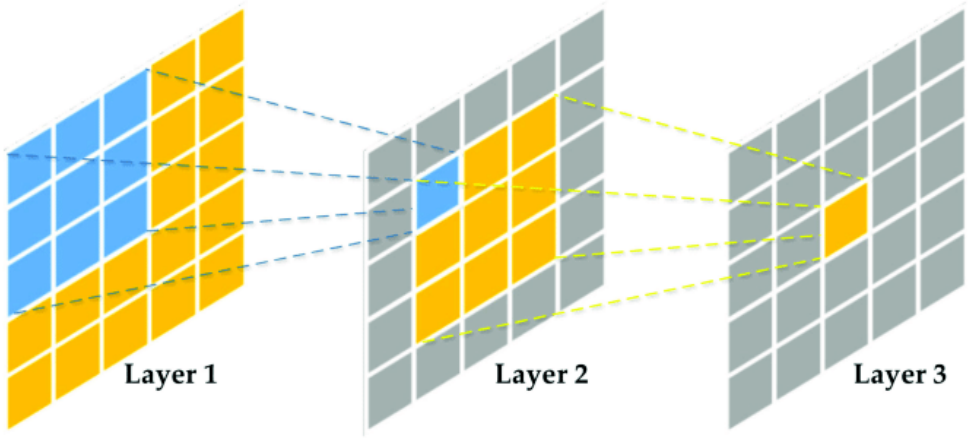


Figure 2.23: Receptive field.

the output feature map sees 3x3 input elements. However, the concept of receptive field is relative, since elements on the feature map of a certain layer can see different areas on the previous layers [22].
Since Deep-CNN use repeated combinations of standard convolutions and max-pooling layers, there is a gradual reduction in the spatial resolution of the feature maps, that impacts negatively on the semantic segmentation objective. Chen et Al. [16] proposed to enlarge the receptive field to permit dense feature extraction, by introducing the *dilation rate* parameter in the convolutions.
This parameter defines the space between values in the kernel, while setting equal to zero the values in between.

Mathematically, considering the 1-D signal $x[i]$, the filter $w[k]$ of length K and the dilation rate r, the output $y[i]$ of the atrous convolution can be defined as:

$$y[i] = \sum_{(k=1)}^{K} x[i + r * k]w[k]$$

When setting $r = 1$, we return to the standard convolution definition.
Moving to 2D-space, it is possible to enlarge the kernel size of a K x K filter to $K_e = k + (k - 1)(r - 1)$ without increasing neither the number of parameters or the number of operations per position. For example, a 3x3 kernel with dilatation rate equal to 2 will have the same receptive field of a 5x5 kernel, while using only 9 parameters instead of 25. This ensures an efficient trade-off between localization (small receptive field) and context assimilation (large field of view).

Returning to the DeepLabV2 architecture, it is composed of 2 modules, the backbone (or feature extractor) and the head.

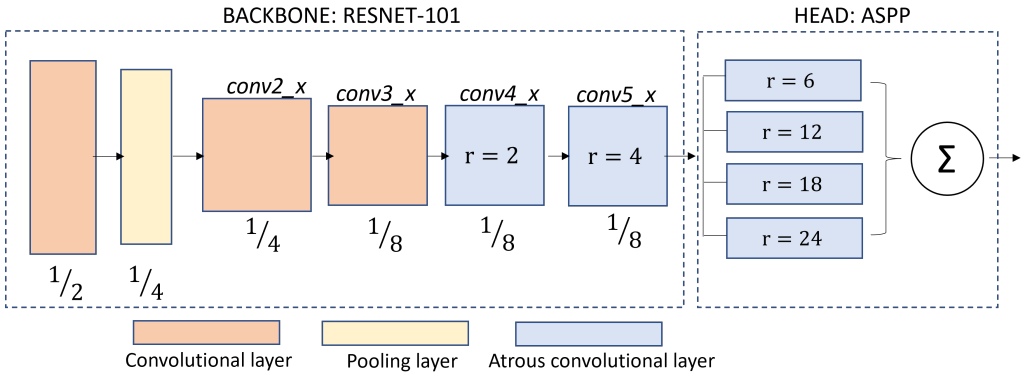


Figure 2.24: DeepLabV2 architecture.

Backbone: ResNet101 In the original paper [16], they re-purpose two popular classification networks (VGG [8] and ResNet101 [23]) as backbones of DeepLabV2. Since they obtained better results with ResNet-101, we adopt this backbone in our DeepLabV2.
In particular, they used an adapted version of ResNet101, repurposed for the semantic segmentation task, by making these modifications:

- All the fully connected layers have been transformed into convolutional layers.
- The feature resolution has been increased using atrous convolution on the last 2 convolutional layers (*conv*4_*x* and *conv*5_*x*), reducing the degree of signal downsampling.

Head: Atrous Spatial Pyramid Pooling Handling scale variability (e.g. objects at multiple scales) is a challenging task in semantic segmentation. To deal with this problem, they introduced the Atrous Spatial Pyramid Pooling (ASPP) module.
The proposed solution consists in applying multiple atrous convolutions to the last feature map, each with a different dilation rate, and then fuse together the outputs.

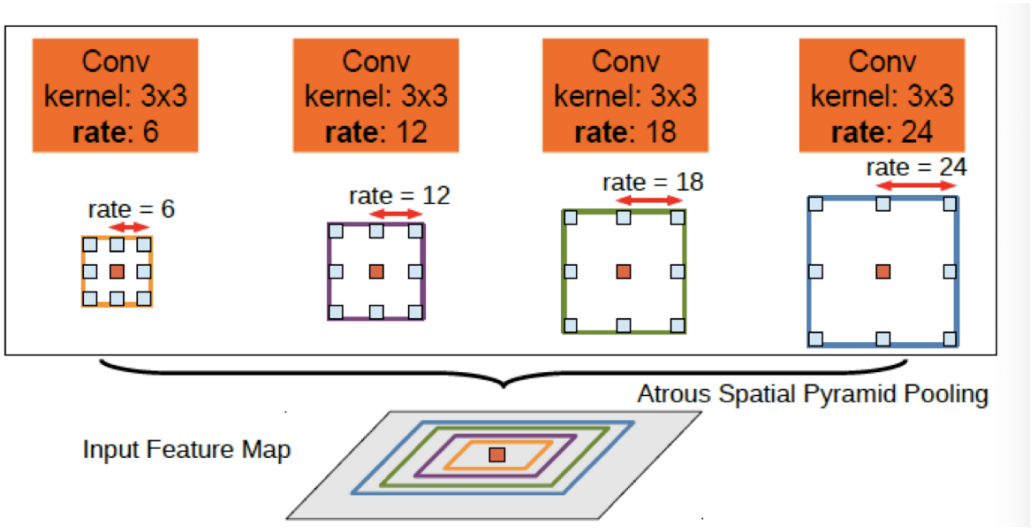


Figure 2.25: Atrous Spatial Pyramid Pooling (ASPP).

This solution allows the network to recognize both large and small objects, and to robustly segment objects at multiple scales.

2.4.1.2 MobileNetV2

In 2017, a group of researchers from Google presented MobileNet [24], a new network designed for mobile devices, optimized to obtain high accuracy while keeping as low as possible the number of parameters and operations. The innovation in this network consists in the introduction of a new layer, the inverted residual and linear bottleneck layer, built on depth-wise separable convolutions.

Depth-wise separable convolution Depth-wise separable convolutions are factorized convolutions that can approximate standard convolutions while significantly decreasing the computational complexity.
To understand how they work, we have a look to the standard convolution operation. Given the input tensor $L_i : h_i \times w_i \times c_{in}$, it applies a convolutional kernel $K \in R^{k \times k \times c_{in} \times c_{out}}$ to produce the $h_j \times w_j \times c_{out}$ output tensor L_j. The corresponding

computational cost is:

$$h_i \times w_i \times c_{in} \times c_{out} \times k \times k$$

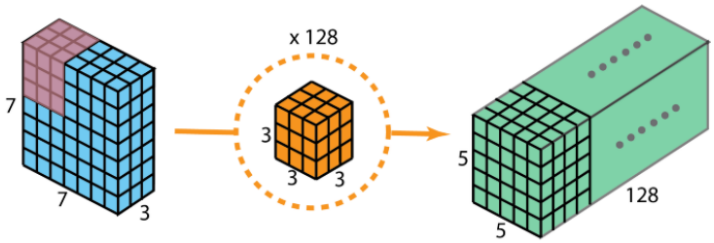
Depthwise Separable convolutions split convolutions into 2 consecutive operations:

- Depth-wise Convolution applies a separate filter per input channel, with total complexity $h_i \times w_i \times c_{in} \times k \times k$
- Point-wise Convolution applies a 1×1 convolution to combine the different channels, with complexity $h_i \times w_i \times c_{in} \times c_{out}$

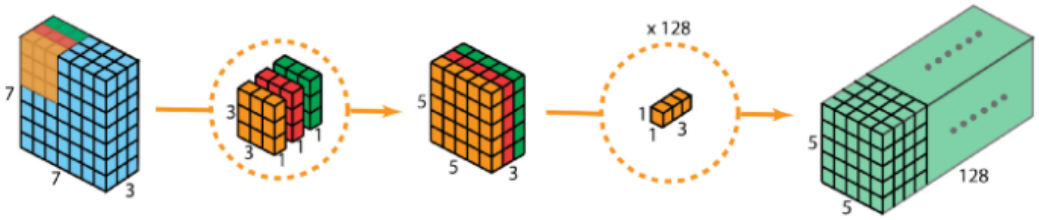
Therefore, the overall complexity decreases to:

$$h_i \times w_i \times c_{in} \times (k^2 + c_{out})$$

To better get this reduction, we can take as example an input feature map having $c_{in} = 32$, and apply a convolution with $k = 3$ and $c_{out} = 64$. For standard convolution, the number of operations per single output location is 18432, while this number reduces to 2336 (8x reduction) when applying depthwise-separable convolution.



a) Standard Convolution.



b) Depthwise Convolution.

Figure 2.26

Linear Bottleneck Layer Starting from the premise that layer activations (e.g. the feature maps) could be embedded in low-dimensional subspaces (1), MobileNetV1 introduced the width multiplier parameter to reduce the dimensionality

of its layers.
However, the presence of non-linearity activations inevitably causes an information loss, as they collapse some channels (2). To retain as much information as possible, a solution consists in increasing the number of channels (3), so that the part of the information lost on one channel could still be preserved in another one.

Based on hypothesis (1) and (2), MobileNetV2 introduced *linear bottlenecks* to fully preserve the information contained in the low-dimensional feature maps, by simply discarding the last non-linearity activation of each residual block.
Instead, to deal with the information loss on the intermediate activation layers of the residual blocks, the number of channels of the feature maps are expanded by the *expansion factor* parameter, relying on hypothesis (3).
Therefore, the operating of each residual block can be essentially decomposed in the following steps:

- The low-dimensional feature maps are expanded by using a 1x1 convolution and moved into the higher dimensional space, more suitable for non-linear activation. The ReLU6 activation function is applied
- A depth-wise 3x3 convolution is performed on the higher-dimensional tensor. The ReLU6 activation function is applied.
- The resulting feature map is projected back to the low-dimensional space using another 1x1 convolution. In this step, no activation function is applied.

Inverted Residual Residual blocks connect the beginning and the end of convolutional blocks with skip connections.
Based on the hypothesis that bottlenecks contain all the necessary information (1), in MobileNetV2 the shortcuts are inserted between the low-dimensional feature maps, rather than between the expansion layers.

The overall structure of the inverted residual and linear bottleneck layer is shown in 2.27.

MobileNetV2 for Semantic Segmentation For the task of Semantic Segmentation, in the original paper[24], authors adopted DeepLabV3 head on MobileNetV2 feature extractor.
However, in this work, we choose to build DeepLabV2 head (described in paragraph 2.4.1.1) on top of MobileNetV2 in order to have more similar and comparable architectures.

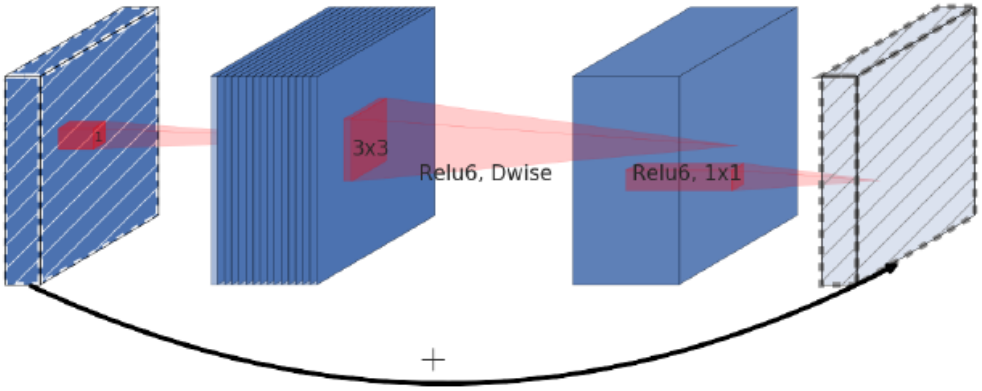


Figure 2.27: MobileNetV2: inverted residual and linear bottleneck layer.

2.5 Unsupervised Domain Adaptation

Generally, Deep Learning methods rely on a large quantity of labeled data. However, data collection is a very expensive and time consuming process. This is especially true for semantic segmentation datasets, where each pixel of images should be manually labeled. For example, the annotation of a single image from the Cityscapes [2] dataset takes up to 90 minutes [3].
A naive solution consists in directly applying models trained on a large-scale labeled domain to the target domain. However, this approach results in poor performances, caused by the domain shift (or dataset bias) between the two domains.
We can understand this phenomenon taking into consideration a real-world example. We want to build a model and use it on Italian roads (i.e. the target domain), but we lack labeled data for this domain. Given the availability of annotated data from American roads, we decide to use them as source data to train the model. When testing the model on the American road images, we will obtain highly accurate results, but when switching to Italian roads we will see a significant drop in the performances, caused by the gap between the 2 domains.
Another option is finetuning the pretrained models in the target domain, but this still requires a relatively large quantity of labeled target images, not always available.
Domain Adaptation (DA) is a form of Transfer Learning (TL) that consists in learning a model from a source domain that can generalize well to a different, but related, domain. These techniques attempt to bridge the gap between domains, transferring the knowledge from the label-rich domain to the target one, characterized by the same (or similar) semantic information.
This approach is very promising in the autonomous driving scenario, where the traffic data obtained with different sensors is particularly difficult to label [3] and

where the training of models requires a huge amount of data derived from different scenarios and climate conditions. The success and the level of fidelity reached by 3D graphical engines have made possible the adoption of synthetic datasets, artificially created by a software. These datasets are very powerful tool, since they can depict images from a multitude of weather and environmental conditions and, at the same time, guarantee perfect and effortless labeling. On the other hand, they have made it necessary to explore new techniques to face the drop of performance caused by the gap between virtual and real-world images. This problem is the so-called synthetic-to-real Domain Adaptation.
Formally, the domain adaptation setting can be subdivided in 3 categories, according to the number of labeled target samples available. In particular, denoting as N_T the number of source samples and as N_{TL} the number of labeled ones, we can categorize DA as:

- Unsupervised DA, when $N_{TL} = 0$
- Fully supervised DA, when $N_{TL} = N_T$
- Semi supervised DA, otherwise

We pose ourselves in the unsupervised DA scenario, assuming we can only access the labeled source domain and the unlabeled target samples and we analyze the main technique of DA, following [3].

2.5.1 Adversarial based Domain Adaptation

Adversarial Generative Models Generative Adversarial Networks (GAN) [25] were proposed by Ian Goodfellow in 2014 and were immediately considered as "the most interesting idea in the last 10 years in ML."
GAN are deep-learning-based generative model composed of a generator g and a discriminator d. The first sub-module, the generator, takes random noise $\mathbf{z}$ and *generates* a virtual sample. In this generative process a fixed-length random vector is converted into a multi-dimensional vector representing a new image.
The second sub-module, the discriminator, is asked to distinguish (or to *discriminate*) between real images and generated ones, where the real samples come from the source dataset and the fake ones from the generator.
The learning process is based on a competitive scenario, where the generator must compete against an adversary: the discriminator tries to maximize the binary probability of correctly classify whether the input sample comes from the generator or the real dataset, while the generator tries to generate more and more realistic images to maximize the probability of d to make a mistake.
Therefore, they are playing the so-called zero-sum game. Zero-sum implies that when the discriminator d correctly classifies the incoming image, it is rewarded

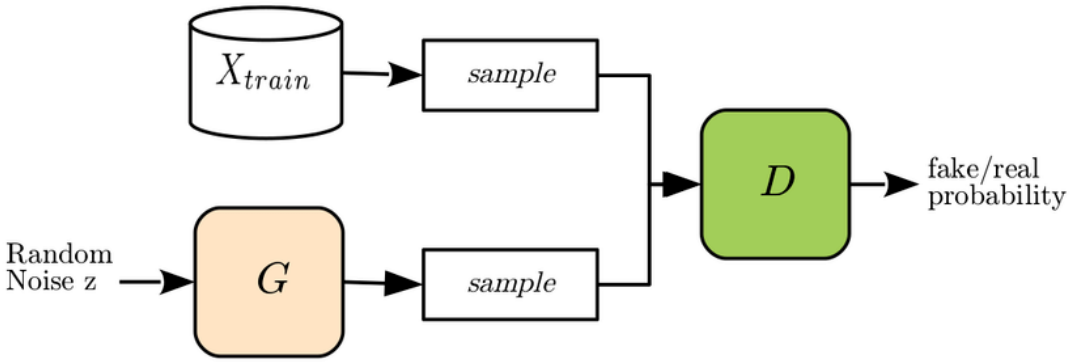


Figure 2.28: Generative adversarial Networks (GAN) [25].

with no modification in its parameters and the generator g is penalized by a large update in its weights. On the contrary, when the generator produces a image able to fool the discriminator, its weights remain unchanged while the discriminator undergoes a significant modification.
Ideally, the limit is reached when the discriminator is not able to distinguish between real and fake images, and assigns a 50% probability to each input sample.

Adversarial Discriminative Models To minimize the divergence between source and target domains, neural networks should learn domain invariant representations. In fact, once the distributions in the 2 domains are aligned, the classifier trained on the source data can be directly applied in the target domain. On this basis, adversarial discriminative models aims at learning features that are both domain-invariant and discriminative.
Inspired by the Generative Adversarial Nets [25], Ganin et al. [26] proposed the Domain-Adversarial Neural Network (DANN), illustrated in fig. 2.29.
The overall architecture is composed of 3 modules:

- The feature extractor G_f: given an input image, coming from the source or target domain, extracts the features *f*. It acts as *generator* in the adversarial network.

- The domain classifier G_d: receives the features f extracted from G_f and distinguishes them as coming from the source or target domain. It acts as *discriminator* in the adversarial network.

- The label predictor G_y: receives the features f extracted from G_f and performs the classification

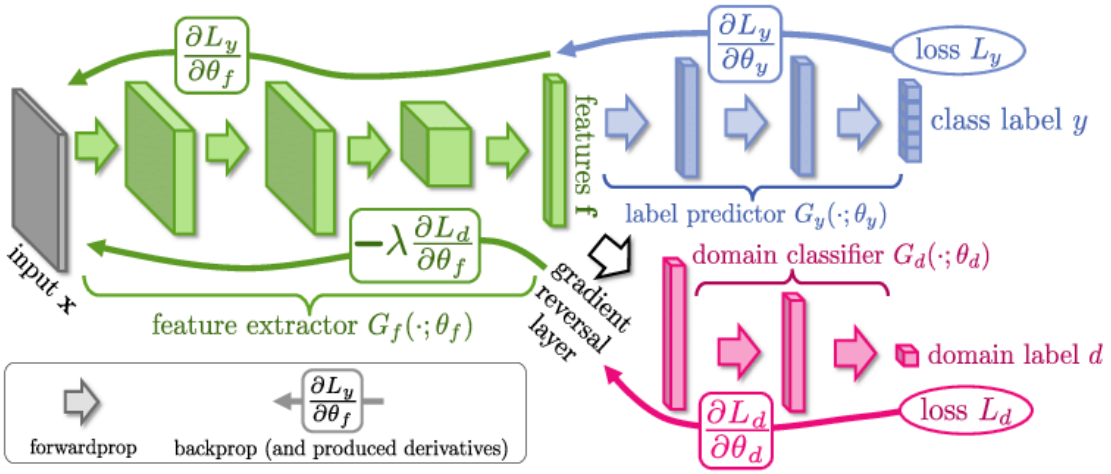


Figure 2.29: Domain-Adversarial Neural Network (DANN) [26].

The feature extractor G_f is optimized to minimize the loss of the label predictor G_y, while maximizing the loss of the domain classifier G_d. This dual optimization is achieved by the introduction of the *gradient reversal layer* (GRL), inserted between the feature extractor G_f and the domain classifier G_d. During the forward pass, it acts as an identity operation, but during the backpropagation it multiplies the gradient by -1 before passing it to the feature extractor, therefore encouraging G_f to generate features more and more similar.

Tzeng et Al. proposed Adversarial Discriminative Domain Adaptation (ADDA) [27], in which they split the optimization process by considering indipendent source and target mappings.
The architecture is composed of:

- The feature extractor M_s: takes the source images x_s as input and extracts the features $M_s(x_s)$
- The feature extractor M_t: takes the target images x_t as input and extracts the features $M_t(x_t)$
- The discriminator D: receives the features extracted from M_s and M_t and distinguishes them as coming from the source or target domain
- The classifier C: performs the classification on the incoming features M_s and M_t

There is a pre-training phase where M_s learns a discriminative representation using the labeled images from the source domain. Then, the target model M_t is

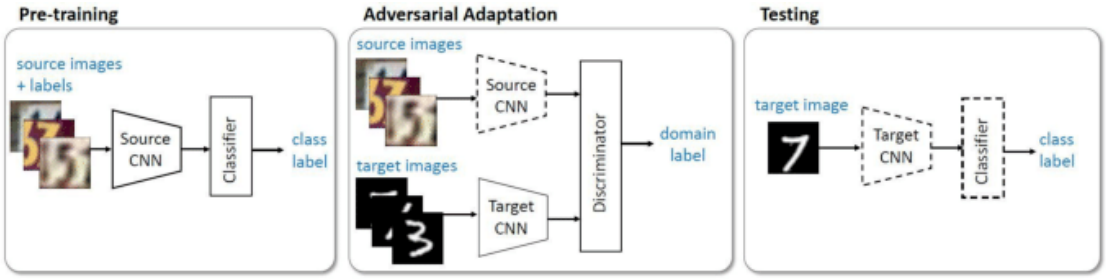


Figure 2.30: Adversarial Discriminative Domain Adaptation (ADDA) [27],

initialized with the parameters of the pre-trained-in-source model M_S and the data are mapped into the same space by using a domain-adversarial loss to train M_t:

$$L_{adv_M}(\mathbf{X_s}, \mathbf{X_t}, D) = -\mathbb{E}_{x_t \sim \mathbf{X_t}}\Big[\log D(M_t(\mathbf{x_t}))\Big]$$

At the end of training phase, M_s is discarded: the target input image are mapped through the target encoder M_t in the domain invariant feature space and classified by the source classifier C. The whole pipeline is described in fig. 2.30.

Tsai et al. extends these approaches to address the pixel-level prediction task in semantic segmentation, by proposing their work "Learning to Adapt Structured Output Space for Semantic Segmentation" [28] in 2018.

They explore the idea that even images very different in appearance in the visual

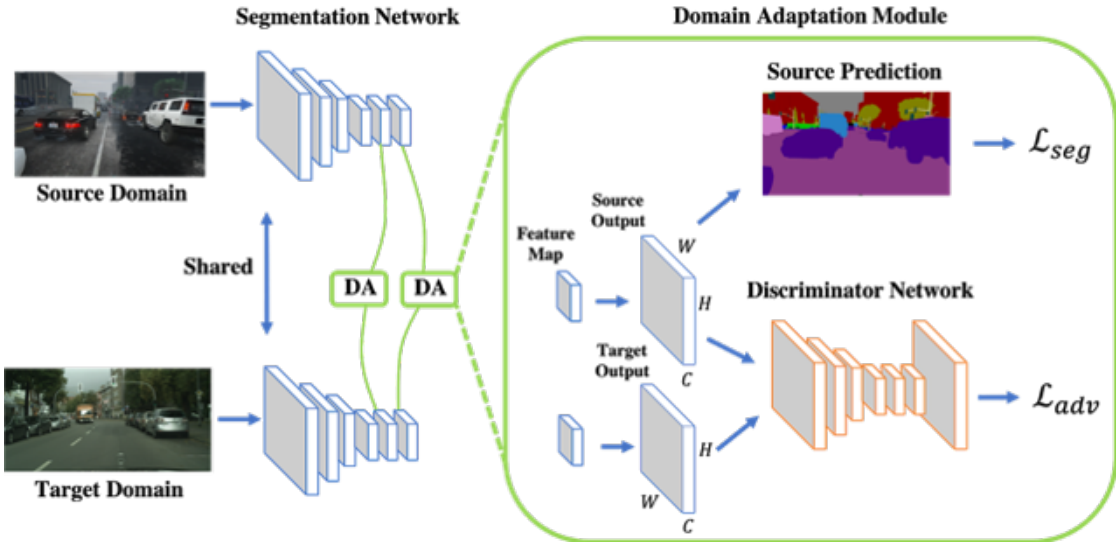


Figure 2.31: Architecture proposed in Learning to Adapt Structured Output Space for Semantic Segmentation [28].

space share many similarities in the segmentation output. Therefore, they pose the pixel-level domain adaptation problem in the output space, trying to directly align the predicted label distributions of source and target images.
The adversarial architecture is composed of:

- The generator G: the segmentation model, that takes in input an image and predicts the segmented output result.
- The discriminator D: receives the segmented output from G and determines whether it corresponds to a source or target image.

The generator is trained with the segmentation loss L_{seg} on the labeled source data, while the discriminator learns how to distinguish between source and target images with the discriminator loss L_d. To encourage the segmentation network to learn domain invariant features, a second loss on the generator is introduced: the adversarial loss L_{adv}. In particular, the generator G tries to *fool* the discriminator, passing him the segmented output of a target image $G(X_t)$ but labeled as source. Minimizing L_{adv} encourages the segmentation network G to generate similar segmentation distributions in both the domains.

2.5.1.1 ADVENT

Models trained exclusively on source domains tend to produce *over-confident* predictions on source-like images and *under-confident* predictions on target ones. Visualizing the entropy maps of the predictions (in fig. 2.32), it is evident that this phenomenon results in low-entropy for source-like images and high-entropy for images coming from target.
Starting from this considerations, Vu et Al. propose ADVENT [29], in which they attempt to bridge the gap between domains by forcing low-entropy on target predictions through a unified adversarial training framework.
In particular, since models trained on source images naturally produce low-entropy predictions for source-like images, minimizing the distance between source and target in the weighted self-information space implies an indirect entropy minimization of the predictions on target images.
Denoting as F the segmentation network, x an input image and $F(\mathbf{x}) = \mathbf{P_x} = \left[\mathbf{P_x}^{(h,w,c)}\right]_{h,w,c}$ the C-dimensional segmentation map, each pixel-wise vector behaves as a discrete distribution over classes. If scores are evenly distributed over classes, it means that the uncertainty, and consequently the entropy, for the pixel is high. On the contrary, if one class stands out, the entropy associated to the pixel is low. This concept can be mathematically expressed through the weighted self-information maps $\mathbf{I_x}^{(h,w)}$:

$$\mathbf{I_x}^{(h,w)} = -\mathbf{P_x}^{(h,w)} \cdot \log \mathbf{P_x}^{(h,w)}$$

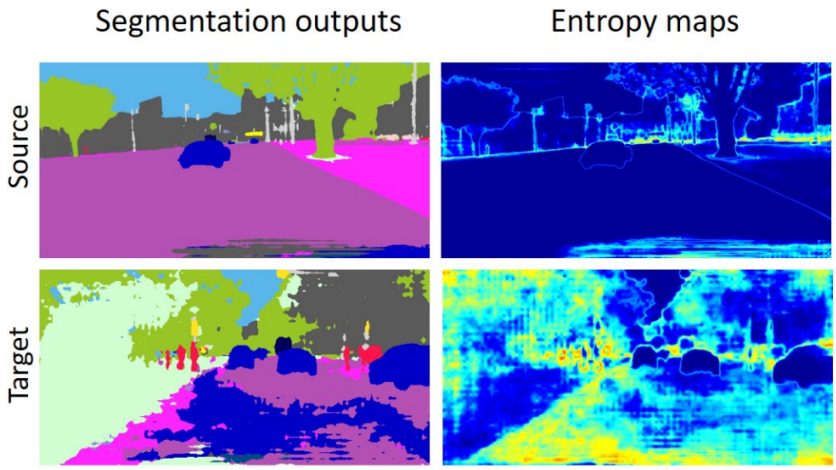


Figure 2.32: A model trained without adaptation produces highly confident prediction for a source image (on the top left), to which corresponds a low-entropy map (on bottom right). On the contrary, the model produces under-confident prediction for a target sample (on the top left) which results in a high-entropy map (on the bottom right).

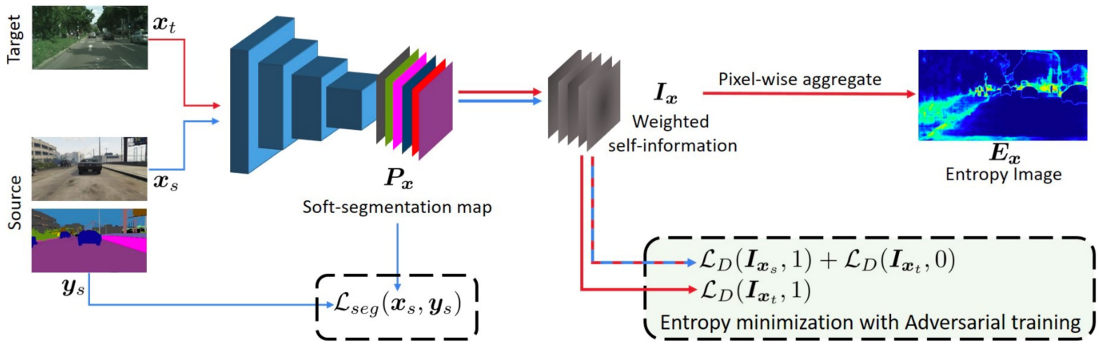


Figure 2.33: ADVENT [29] overview.

I_x is fed as input to a fully convolutional discriminator D, trained to classify whether the input is coming from the source domain X_s or from the target domain X_t, labeled respectively as 1 and 0, with a cross-entropy domain classification loss $\mathcal{L}_D$. The training objective for the discriminator is:

$$\min_{\theta_D} \frac{1}{|X_s|} \sum_{x_s} \mathcal{L}_D(\mathbf{I}_{\mathbf{x_s}},1) + \frac{1}{|X_t|} \sum_{x_t} \mathcal{L}_D(\mathbf{I}_{\mathbf{x_t}},0)$$

At the same time, the segmentation network F is trained with a standard cross-entropy loss $\mathcal{L}_{seg}$ on the source domain. Jointly, an adversarial loss on F forces the alignment between the source and target domain. In particular, the segmentation

network tries to fool the discriminator by passing it the weighted self-information maps of target images but labeled as source. The adversarial loss encourages the segmentation network to produce low entropy predictions also on the target domain, minimizing the distance between source and target in the weighted self-information space. Combining the segmentation loss and the adversarial loss, the optimization problem on the segmentation network F can be formulated as:

$$\min_{\theta_F} \frac{1}{|X_s|} \sum_{x_s} \mathcal{L}_{seg}(\mathbf{x_s}, \mathbf{y_s}) + \frac{\lambda_{adv}}{|X_t|} \sum_{x_t} \mathcal{L}_D(\mathbf{I_{x_t}}, 1)$$

where λ_{adv} is a parameter weighting the contribution of the adversarial loss.

2.5.2 Self Training based Domain Adaptation

A promising approach in DA is Self-Learning. It consists in training a model using the labeled samples from the source domain and then applying the resulting model to generate artificial pseudolabels for some of the unlabeled images. These predictions are then used as ground truth labels for the target images and used to re-train the model in a following iteration. This procedure can be repeated iteratively, typically until no more unlabeled data are available.
Following the notation of [30], we denote the set of labeled samples as $S = (\mathbf{x_i}, y_i)_{1 \leq i \leq m}$, and the set of unlabeled samples as $X_U = (\mathbf{x_i})_{m+1 \leq i \leq m+u}$. At each iteration, a set X_T of unlabeled samples is removed from X_U and corresponding pseudo-labels $\tilde{y}$ are generated. These pseudo-labeled samples are added to the labeled points set and a new supervised classifier h is trained over $S \cup X_T$. In particular, h is trained to minimize the regularized empirical loss:

$$\frac{1}{m} \sum_{(\mathbf{x_i}, y_i) \in S} \ell(h(\mathbf{x}), y) + \frac{\gamma}{|X_T|} \sum_{(\mathbf{x}, \tilde{y}) \in X_T} \ell(h(\mathbf{x}), \tilde{y}) + \lambda \|h\|^2 \tag{2.1}$$

where γ is an hyperparameter that controls the impact of the pseudo-labels in learning and λ is a regularization parameter.

This approach admits a multitude of variations and design decisions. Most important, it is necessary to determine which data select during the pseudo-labeling phase, since this choice highly affect the performance of the model. Usually, this criterion is based on the prediction confidence of the generated pseudo-labels, therefore the ranking of prediction probability for the unlabeled samples should reflect the true confidence ranking. Then, it is necessary to decide whether to re-use pseudo-labelled data in later stages of the training. Another decision affects the weight of the pseudo-labelled data. In fact, in the first stages, the generated

pseudo-labels are less reliable, therefore it is necessary to design a way to increase their weights during the training stages.
In addition, some approaches, instead of pre-computing the pseudo-labels offline and consequently repeat the training process, generate the pseudo-labels online.

To avoid training instabilities, many solutions have been proposed.
For instance, Pan et al. [31] hypothesize that updating the pseudo-labels at the end of one training stage leads the network to overfit on the noisy labels. Therefore they proposed to use representative prototypes to estimate the likelihood of pseudo-labels to facilitate online correction during training, by exploiting the distances from the prototypes (the centroids of the classes) to rectify the pseudo-labels.
Other approaches combine **consistency regularization** with **Data Augmentation**, based on the idea that predictions on unlabelled samples should be invariant to perturbations. An example is [32], that uses photometric noise, flipping and scaling to ensure consistency of the predictions across transformations. In particular, the set of augmentation is applied to each target image and the corresponding pseudo-label is generated by selecting confident pixels from the averaged map using thresholds based on running statistics.
Another Data Augmentation based method is FixMatch [33], that applies weak augmentations on the target samples and feeds the weakly augmented images into the model to obtain predictions. The pseudo-labels are generated by retaining

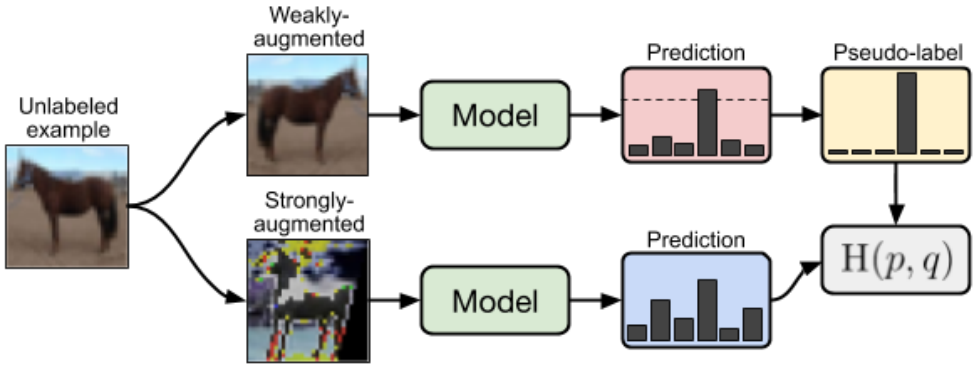


Figure 2.34: FixMatch [33] approach.

the classes predicted with probability above a certain threshold. Then, the same images are augmented with strong transformation and coupled with the previously generated pseudo-labels to train the network (see fig. 2.34).
A promising augmentation technique is **mixing** (or **domain-mixup**), that consists in combining pixels from different training images to produce new perturbed samples.
As the title of their paper states, French et Al. [34] affirm that "Semi-supervised

semantic segmentation needs strong, varied perturbations". In their work, they explore variants of the popular CutOut and CutMix augmentations, where a rectangular region is cut from one image and pasted into another one. The corresponding pseudo-label is generated by mixing, in the same way, their predictions. As result, strongly perturbed images are generated and used, along with their pseudo-labels, to train the model. Similar approaches are the one proposed by ClassMix [35], where half of the classes of the first image are selected and corresponding pixels are pasted into a second image (see fig. 2.35), and by Domain Adaptation via Cross-domain mixed Sampling (DACS) [36] (see section 2.5.2.3).

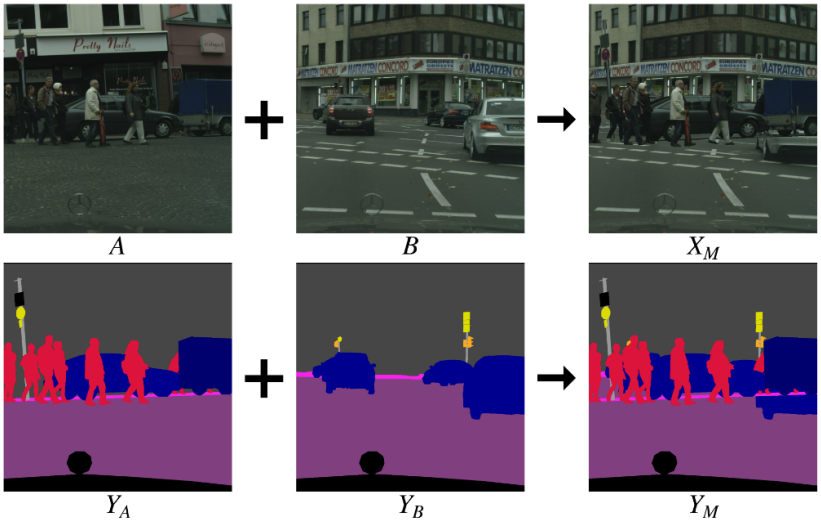


Figure 2.35: ClassMix [35]. Half of the classes of the first images are selected and corresponding pixels are copied and pasted into the second image.

A more sophisticated technique has been formulated by Zhou et al. [37]. They proposed end-to-end Context-Aware Mixup (CAMix), which exploits contexts as prior knowledge to mitigate the domain gaps and guide the context-aware domain-mixup. In particular, given a source and a target image as input, the contextual mask is generated, by leveraging spatial distribution of source and contextual relationships of target. Then, this mask is used to combine source and target images to produce mixed images, mixed pseudo-labels and mixed significance masks, where the latter are used to guide the significance-reweighted consistency loss (SRC) on the mixed images and mixed pseudo-labels.

A common problem in self training is the model bias towards easy classes, caused essentially by two factors. First, datasets are often unbalanced and follow long-tail distributions. Second, SSL-methods usually choose pseudo-labels with high prediction confidence, therefore penalising the performances for the hard

classes.
To address this issue, many strategies have been proposed.
For example, [38] design the self-motivated pyramid curriculum domain adaptation (PyCDA), where they combine curriculum domain adaptation (CDA) with self-training (ST). As shown in fig.2.36, they merge the estimated target label

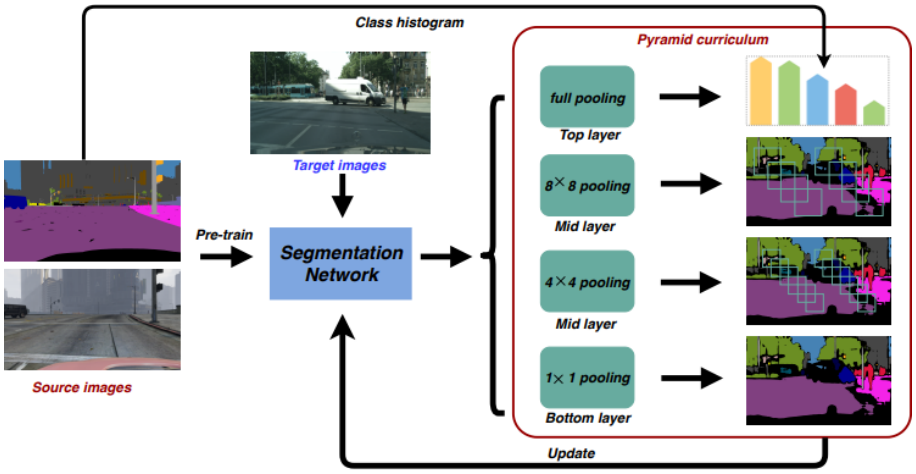


Figure 2.36: Self-motivated pyramid curriculum domain adaptation (PyCDA)[38].

distributions with the pseudo-labeled target pixels to construct a three-layers pyramid containing precious information regarding the target domain, mainly about the desired label distributions over the target domain images (over the top-layer), image regions (over the middle-layers), and pixels (over the bottom-layer).
Other techniques rely on re-weighting, such as Class-Balanced Self-Training (CBST) (see section 2.5.2.1) and Instance Adaptive Self-Training for Unsupervised Domain Adaptation (IAST) [39]. IAST uses an instance adaptive selector to adapt the pseudo-label threshold for each semantic class, gradually decreasing the proportion of hard classes, while a region guided regularization prevents the model to overfit on the pseudo-labels by smoothing the prediction results of the confident regions and sharpening the prediction of the ignored areas.
Other strategies apply class-based sampling to perform oversampling on the rarest classes. An example is DAFormer [40] (see section 2.5.2.4), that addresses the co-occurrence of rare and common classes in a single semantic segmentation sample by performing Rare Class Sampling.

2.5.2.1 Unsupervised Domain Adaptation for Semantic Segmentation via Class-Balanced Self-Training: CBST

Simply using the generated pseudo-labels to optimize the model is self-referential, because it is not possible to guarantee the correctness of the predictions.

Therefore, an effective solution is to follow the *self paced curriculum* scheme, that consists in consider the most confident predictions to update the model, and then explore the remaining pseudo-labels with less confidence. This approach is inspired by the human learning process, where the training proceeds from easy to more complex samples [41].

Self-Training with self-paced Learning We denote the s^{th} training source sample as I_s, the corresponding ground truth of the n^{th} pixel as $y_{s,n}$ and $p_n(w, I_s)$ the softmax output for the n-pixel of sample I_s. Similarly, I_t is the target sample with corresponding pseudo-label $\hat{y}_{t,n}$. C is the number of classes, $e^{(i)}$ a one hot vector and w the weights of the network.
With this notation, the self-paced curriculum learning can be described as:

$$\begin{aligned} \min_{\mathbf{w},\hat{\mathbf{y}}} \mathcal{L}_{ST}(\mathbf{w},\hat{\mathbf{y}}) = \quad & -\sum_{s=1}^{S}\sum_{n=1}^{N} \mathbf{y}_{s,n}^{T} \log(\mathbf{p}_n(\mathbf{w},\mathbf{I_s})) \\ & -\sum_{t=1}^{T}\sum_{n=1}^{N} \left[\hat{\mathbf{y}}_{t,n}^{T} \log(\mathbf{p}_n(\mathbf{w},\mathbf{I}_t)) + k|\hat{\mathbf{y}}_{t,n}|_1\right] \\ \text{s.t.} \quad & \hat{\mathbf{y}}_{t,n} \in \{\{\mathbf{e}^{(i)}|\mathbf{e}^{(i)} \in \mathbb{R}^C\} \cup \mathbf{0}\}, \forall t, n \\ & k > 0 \end{aligned} \tag{2.2}$$

When the pseudo-label $\hat{y}_{t,n}$ is assigned to 0, it is not considered in the learning process. To prevent the model to trivially ignore all the pseudo-labels, the L_1 regularization term is introduced as a negative sparse. This term is controlled by the hyperparameter k, which determines the amount of ignored pseudo-labels.

Class-Balanced Self-Training Since it is common to obtain higher prediction scores for the easy-to-transfer classes, setting a unique hyperparameter k to control the amount of selected pseudo-labels will encourage the bias toward the initially well-transferred classes, causing the model to ignore the hard classes.
To face this problem, Zou et Al. propose Class-Balanced Self-Training [42], in which they suggest the introduction of a class-balanced self-training framework, where the amount of selected pseudo-labels is controlled by a different hyperparameter k_c

for each class:

$$\begin{aligned}\min_{\mathbf{w},\hat{\mathbf{y}}} \mathcal{L}_{CB}(\mathbf{w},\hat{\mathbf{y}}) = \quad & -\sum_{s=1}^{S}\sum_{n=1}^{N} \mathbf{y}_{s,n}^{T} \log(\mathbf{p}_n(\mathbf{w},\mathbf{I}_s)) \\ & -\sum_{t=1}^{T}\sum_{n=1}^{N}\sum_{c=1}^{C} \left[\hat{\mathbf{y}}_{t,n}^{(c)} \ \log(p_n(c|\mathbf{w},\mathbf{I}_t)) + k_c \hat{y}_{t,n}^{(c)}\right] \\ \text{s.t.} \quad & \hat{\mathbf{y}}_{t,n} = \left[\hat{y}_{t,n}^{(1)},...,\hat{y}_{t,n}^{(C)}\right] \in \{\{\mathbf{e}^{(i)}|\mathbf{e}^{(i)} \in \mathbb{R}^C\} \cup \mathbf{0}\}, \forall t,n \\ & k_c > 0, \ \forall c\end{aligned} \tag{2.3}$$

The optimization flow consists in the repetition of several rounds, where each round is composed of 2 steps:

1. The most confident pseudo-labels from the target domain are selected.
   More specifically, the model is used to generate the predictions for the target images, retaining the maximum probability for each pixel and its corresponding class.
   Given each class c, the class c probabilities are selected and sorted in descending order. Denoting as N_c the number of pixels assigned to c, the parameter k_c is set such that $exp(-k_c)$ equals to the amount (p * N_c), where the probability value of p starts from 20% and it is empirically increased of 5% at each round. Intuitively, the $p \times 100\%$ most confident pseudo-labels for each class c are selected at each round.

2. Pseudo-labels are used as ground truth labels for the target images to train the network with a standard Cross-entropy loss.

2.5.2.2 Fourier Domain Adaptation: FDA

In 2020, Yang et Al. published their paper Fourier Domain Adaptation for Semantic Segmentation [43] in CVPR.
In their work, they observe that even if images from different domains share several characteristics at high level, there are low-level source of variability, such us illumination or sensors quality, that can vary substantially without affecting the visual perception. In particular, it is possible to observe that even if pictures are perceptually very close the corresponding low-level spectrum can vary significantly. To make the model able to properly generalize, it is necessary to represent this variability also in the source domain, therefore aligning the low-level statistics of the source and target domain.

Fourier Alignment The proposed solution consists in computing the Fast Fourier Transform (FFT) of each source image and replacing its low-level frequencies with the low-level spectrum of a random sampled target image.

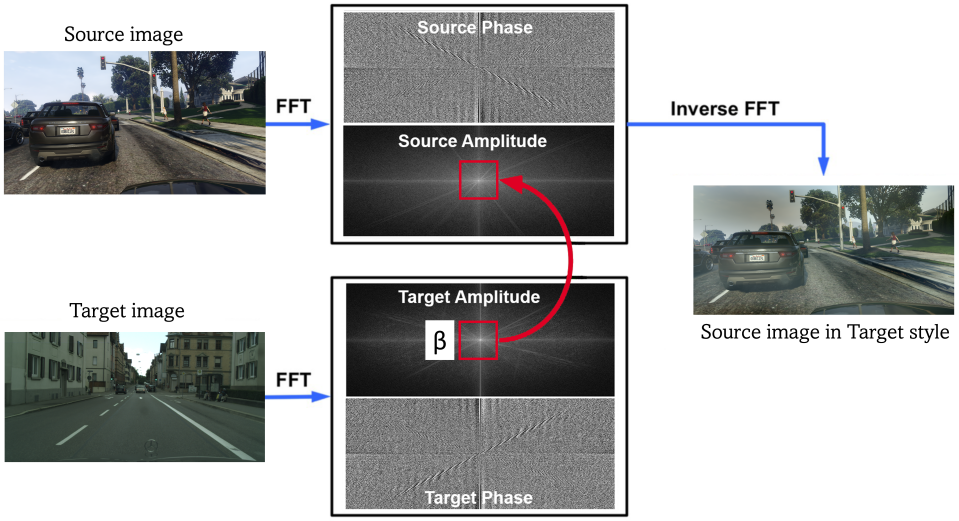


Figure 2.37: Fourier Alignment. Source and target images are mapped into the frequency domain. The low-level spectrum of the target image is copied and pasted into the spectrum of the source sample, which is then converted back into the image space.

In particular, given an input image I_s from the source domain D_s, its Fourier Transform can be computed as:

$$\mathcal{F}(I_s)(m,n) = \sum_{(h,w)} I_s(h,w) e^{-j2\pi(\frac{h}{H}m + \frac{w}{W}n)}, j^2 = -1$$

from which amplitude $\mathcal{F}^A(I_s)$ and phase $\mathcal{F}^P(I_s)$ can be extracted.
Analogously, we can compute the Fourier transform of a randomly picked image I_t from D_t and extract its amplitude $\mathcal{F}^A(I_t)$ and phase $\mathcal{F}^P(I_t)$.
The amplitude component has dimension $H \times W$. Assuming that the image is centered in (0,0), the low-frequencies are contained in the (0,0)-centered rectangle $[\beta H : \beta H, -\beta W : \beta W]$, where β is a parameter in (0,1).
To extract the low-level spectrum of the target image we select a mask of dimension $[-\beta H : \beta H, -\beta W : \beta W]$ centered in (0,0), where all the values inside the rectangle are set equal to 1, while the ones outside are set equal to 0:

$$M_\beta(h,w) = \mathbf{1}_{(h,w) \in [-\beta H : \beta H, -\beta W : \beta W]}$$

This mask is applied to the target image spectrum $\mathcal{F}^A(I_t)$ to retain only its low-frequencies amplitude, which are replaced into $\mathcal{F}^A(I_s)$.
Finally, the inverse Fourier transform $\mathcal{F}^{-1}$ is computed to return into the image space.
Following these steps, the resulting image will have:

- low-frequencies amplitude of I_t
- high frequencies amplitude of I_s
- phase of I_s

This procedure can be formalized as:

$$I_{s\to t} = \mathcal{F}^{-1}([M_\beta \circ \mathcal{F}^A(I_t) + (1 - M_\beta) \circ \mathcal{F}^A(I_s), \mathcal{F}^P(I_s)])$$

The result is an image having the same content of I_s, but with appearance closer to I_t. An example is shown in fig. 2.37.

Fourier Alignment for Semantic Segmentation Given the adapted source dataset $D^{s\to t}$ composed of source-in-target images $x_i^{s\to t}$ and corresponding labels y_i^s, the segmentation network ϕ^w with parameters w is trained by minimizing the cross-entropy loss:

$$\mathcal{L}_{ce}(\phi^w; D^{s\to t}) = -\sum_i \left\langle y_i^s \ , \ \log(\phi^w(x_i^{s\to t})) \right\rangle$$

Fourier Alignment combined with Self Learning Once the segmentation network ϕ^w has been trained, it is possible to perform Self-Supervised training to boost the performances of the network.
The proposed solution consists in averaging the predictions of multiple models to generate the pseudo-labels, regularizing the learning process and making the model robust to variance.
In particular, 3 different models are trained separately using different values of β. Once the models are trained, each target image I_t is passed through ϕ_j, $j = 1,2,3$ to obtain the predictions $\phi_j(I_t)$.
Setting $M = 3$, the mean prediction for I_t can be computed as:

$$\hat{y} = \arg\max_k \frac{1}{M} \sum_m \phi_j(I_t)$$

To create the pseudolabel for I_t only the highly confident predictions are retained. This means that, for each pixel, its predicted semantic class is retained only if its

confidence is within the top 66% or above 0.9; otherwise the pixel is set to void. The entire set of highly confident mean predictions is Y_{psu}.
Finally, the pseudo-labels are used as if they were ground truth labels for the target domain, obtaining the pseudo-labeled dataset $(\hat{D}_t, Y_{psu})$, combined with $(D^{s \to t}, Y_s)$ to train again the 3 segmentation networks. The entire procedure can be repeated for many rounds.

2.5.2.3 Domain Adaptation via Cross-Domain Mixed Sampling: DACS

Existing methods mix images from the unlabeled target domain, along with their pseudo-labels, to generate augmented samples. These new samples are then used in parallel with the source images to train the networks.

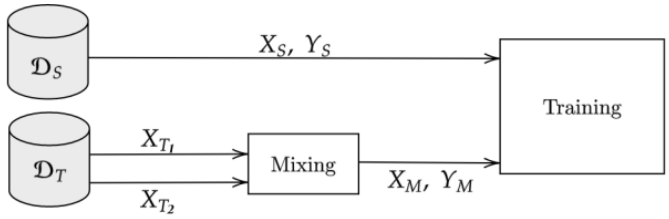


Figure 2.38: Naive Mixing: images X_{T_1} and X_{T_2} from the target domain are mixed to produce the mixed image X_M and corresponding mixed label Y_M. The network is optimized on batches of augmented images and of source images.

However, this approach, referred to as "Naive Mixing", causes the model to conflate some of the classes in the target domain. For example, the rare *rider* class is confused with the more frequent *person*, and the *sidewalk* with the more popular *road*, similarly to the previously described bias toward easy-to-transfer classes. This phenomenon occurs only for the target domain, and does not affect the images sampled from the source domain.
Rather than exclusively consider the target domain for the domain-mixup, Tranheden et Al. [36] proposed to mix images from the source and the target domain, adapting the mixing strategy of ClassMix [35].

Cross-Domain Mixed Sampling Each augmented image is generated by selecting half of the classes from a source image and pasting them into a target image. To construct the corresponding label, first of all the target image is passed through the network to obtain its pseudo-label. Then, the generated pseudo-label is combined pixel-wise with the ground-truth label of the source image in the same way as the images. An example is shown in fig. 2.40.

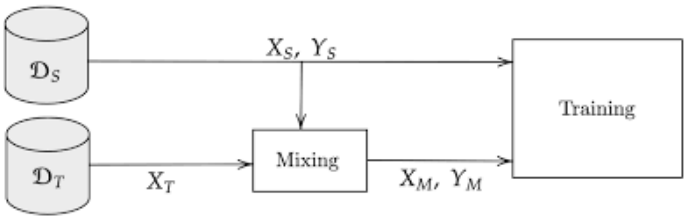


Figure 2.39: DACS mixing: a target image X_T is mixed with a source image X_S to produce the mixed image X_M. The corresponding mixed label Y_M is obtained by mixing the source label with the target pseudo-label. The network is optimized on batches of augmented images and of source images.

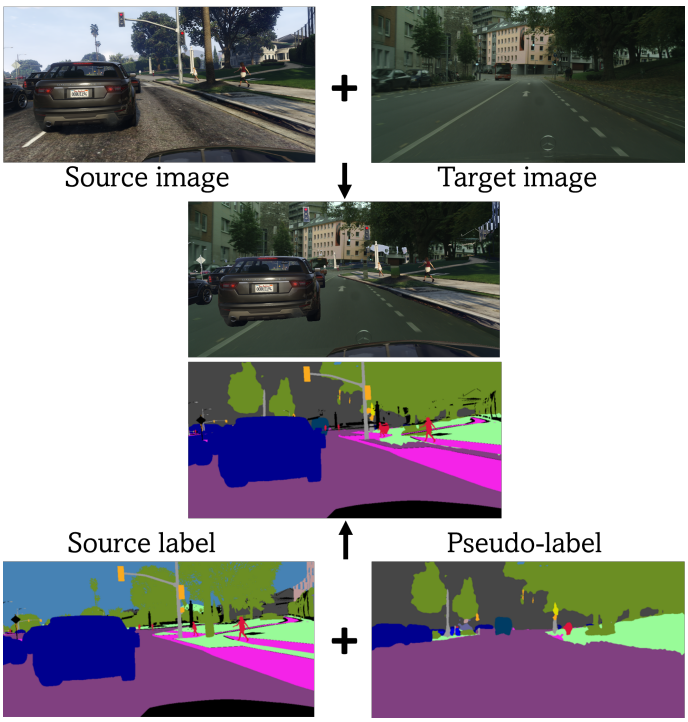


Figure 2.40: Cross-Domain mixing. Half of the classes of a source images are selected and the corresponding pixels are copied and pasted into the target image; the same mask is applied on the source label and the selected pixels are pasted into the pseudo-label generated for the target image.

Cross-Domain Mixed Sampling for Semantic Segmentation Denoting as X_S the source images with corresponding ground truth Y_S, X_M and Y_M the mixed images and pseudo-labels, the network parameters θ are trained by minimizing the

loss:

$$L(\theta) = E\Big[\ H(f_\theta(X_S), Y_S) + \lambda H(f_\theta(X_M), Y_M)\Big]$$

where H is the cross-entropy loss and λ an hyperparameter weighting the unsupervised part of the loss.

2.5.2.4 Improving Network Architectures and Training Strategies for Domain-Adaptive semantic Segmentation: DAFormer

Despite several effective domain adaptation methods have been proposed in recent years, they are mostly based on outdated networks architectures. In addition, recent studies for Image Classification have proven Convolutional Neural Networks to be sensitive to distribution shifts such as adversarial noise [44], image corruptions [45] and domain shifts [46].
Starting from these premises, Hoyer et Al. [40] proposed to combine the mixing strategy of DACS (see 2.5.2.3) with a new architecture, referred to as DAFormer, based on attention-based Transformers [47, 48], which has been proven to be more robust than CNN with respect to these shifts.
In addition, to limit overfitting on the source domain, three strategies are introduced during training: Rare Class Sampling, Thing-Class ImageNet Feature Distance and learning rate warmup.

DAFormer architecture The self-attention mechanism is quite similar to the standard convolutions, since they both perform a weighted sum. However, standard convolutions use weights learned during the training but fixed in inference, while the attention-mechanism involve weights that are dynamically computed based on the affinity between every pair of tokens, making the model more adaptive.

In particular, the architecture of DAFormer is composed of a Transformer encoder, which follows the design of Mix Transformers (MiT) [49], and a multi level context-aware feature fusion decoder.
The encoder treats images as sequences of tokens, where each token is extracted dividing the input images into patches of size 4×4, using an overlapping patch merging process to preserve local continuity. These small patches are then fed as input to the hierarchical Transformer encoder, that performs efficient Self-Attention using sequence reduction [50]. This mechanism produces multi-level multi-scale features F_i with resolution of $\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i$, where $i \in \{1,2,3,4\}$. This hierarchical representation is essential to semantic segmentation, because it permits to retain both high-resolution coarse features and low resolution fine-grained features.
To this purpose, before the feature fusion, each feature map is embedded to the same number of channels C_e by a 1×1 convolution and upsampled to the size of F_1.

Then the resulting feature maps are concatenated and passed to the context-aware feature fusion module.
In this module, multiple parallel 3×3 depthwise separable convolutions with different dilation rates are applied, followed by a 1×1 convolution to fuse the stacked multi-level features.
The architecture of DAFormer is depicted in fig 2.41

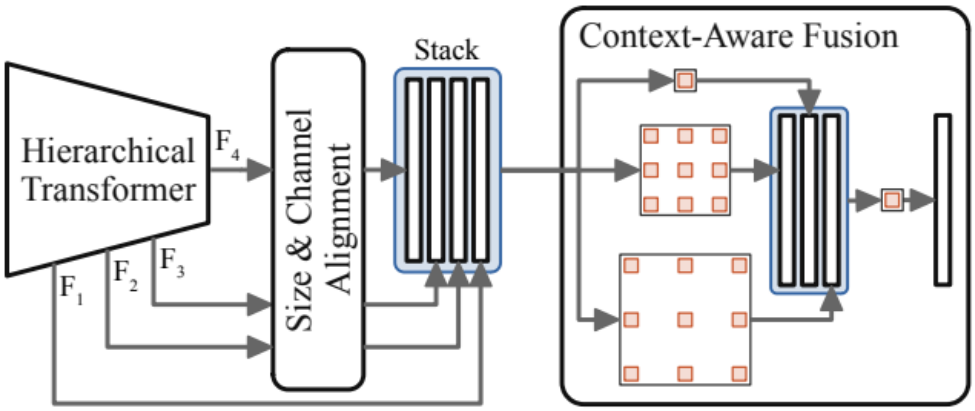


Figure 2.41: DAFormer [40] architecture.

Training strategies: Rare Class Sampling, Thing-Class ImageNet Feature Distance and Learning Rate Warmup In parallel with this powerful architecture, some strategies to limit overfitting on the source data have been proposed.

Starting from the premise that the later a certain class in seen during training, the worse will be the network performance on that class, **Rare Class Sampling** (RCS) attempts to mitigate the bias toward initially well-transferred class by sampling more frequently images containing rare classes.
The frequency of each class f_c is simply the number of pixels of class c in the source dataset:

$$f_c = \frac{\sum_{i=1}^{N_S} \sum_{j=1}^{H \times W} \left[Y_S^{(i,j,c)} \right]}{N_S * H * W}$$

Consequently, it is possible to define the sampling probability for each class c in function of f_c :

$$P(c) = \frac{e^{\frac{(1-f_c)}{T}}}{\sum_{c'=1}^{C} e^{\frac{(1-f_{c'})}{T}}}$$

where T is referred to as *temperature* and controls the smoothness of the distribution.
To select a source image, a class is sampled from the probability distribution $c \sim P$ and the image is sampled from the subset of images containing the selected class c. It is worth noting that sampling more frequently images containing rare classes does not imply ignoring the common classes during training, since they co-occurs with multiple rare classes and therefore are already covered.

Analysing the behaviour of DAFormer in UDA, authors have noticed that the performances of the network over some classes decrease over time. This phenomenon is related to the initialization of the network. In particular, it is common to initialize the segmentation network with the weights from ImageNet classification, because this dataset provides useful information since it contains real-world images.
During training, meaningful features from ImageNet are corrupted while the model overfits to synthetic data. To face this gradual corruption, the proposed solution, referred to as **Thing-Class ImageNet Feature Distance**, consists in regularizing the distance between the bottleneck features of the segmentation network f_θ and the bottleneck features $F_{imageNet}$ of the ImageNet model:

$$d^{(i,j)} = \|F_{ImageNet}(x_s^{(i)})^{(j)} - F_\theta(x_s^{(i)})^{(j)}\|$$

More specifically, the Feature Distance loss is calculated exclusively on *thing-classes*, since the ImageNet model is mostly trained on objects with well-defined shape (such as *cars* or *buildings*), rather than stuff-classes, characterized by amorphous background (such as *road* or *sky*). Denoting as M_{things} the binary mask identifying the thing-classes pixels, the Feature Distance loss is defined as:

$$\mathcal{L}_{FD}^{(i)} = \frac{\sum_{j=1}^{H_F \times W_F} d^{(i,j)} \cdot M_{things}^{(i,j)}}{\sum_j M_{things}^{(i,j)}}$$

In particular, to obtain the mask M_{things}, the label y_s^c is downsampled to the bottleneck feature size, performing average pooling with patch $\frac{H}{H_F} \times \frac{W}{W_F}$ to each class channel. A class is kept when it exceed the ratio r:

$$y_{S,small}^c = \left[AvgPool\left(y_s^c, \frac{H}{H_F}, \frac{W}{W_F}\right) > r \right]$$

In order to consider only feature pixels containing dominant thing-class, the mask is computed as:

$$M_{things}^{(i,j)} = \sum_{c'=1}^{C} y_{S,small}^{i,j,c'} \cdot \left[c' \in C_{things} \right]$$

Lastly, to avoid that a large adaptive learning rate distorts features from ImageNet pretraining, authors proposed **Learning Rate Warmup**, adapting to UDA a successful technique adopted to train both CNN and Transformers. In particular, during the warmup period, referred to as t_{warm}, the learning rate at iteration t is calculated as:

$$\eta_t = \eta_{base} \cdot \frac{t}{t_{warm}}$$

The overall loss $\mathcal{L}$ is:

$$\mathcal{L} = \mathcal{L}_{\mathcal{S}} + \mathcal{L}_{\mathcal{T}} + \lambda_{FD}\mathcal{L}_{\mathcal{FD}}$$

where $\mathcal{L}_{\mathcal{S}}$ and $\mathcal{L}_{\mathcal{T}}$ are the cross-entropy losses computed, respectively, on the labeled source images and the pseudolabeled mixed images.

2.6 Datasets

To validate and compare the methods that we are going to describe, we use a popular synthetic-2-real Unsupervised Domain Adaptation setting, where the synthetic source labeled data comes from either GTA5 [5] or SYNTHIA [51], and the unlabeled target data from Cityscapes [2].

2.6.1 GTA5

GTA5 [5] dataset consists of 24.966 images, each having resolution 1914 x 1052. Each image is paired with the corresponding label, provided with pixel-level semantic annotation of 33 classes.
To make the dataset compatible with other Semantic Segmentation datasets for outdoor scenes, only a subset of 19 classes is considered.
All the images are from car perspective and the represented streets are from American-style virtual cities.

2.6.2 SYNTHIA

SYNTHIA (*a SYNTHetic collection of Imagery and Annotations of urban scenarios*) [51] is a collection of photo-realistic frames, subdivided in 2 complementary sets of images: SYNTHIA-Rand (the folder adopted in this work) and SYNTHIA-Seqs.
SYNTHIA-Rand consists of 13.400 images, each of resolution 960 x 720 pixels, paired with the corresponding label, provided with precise pixel-level semantic annotation of 13 classes.

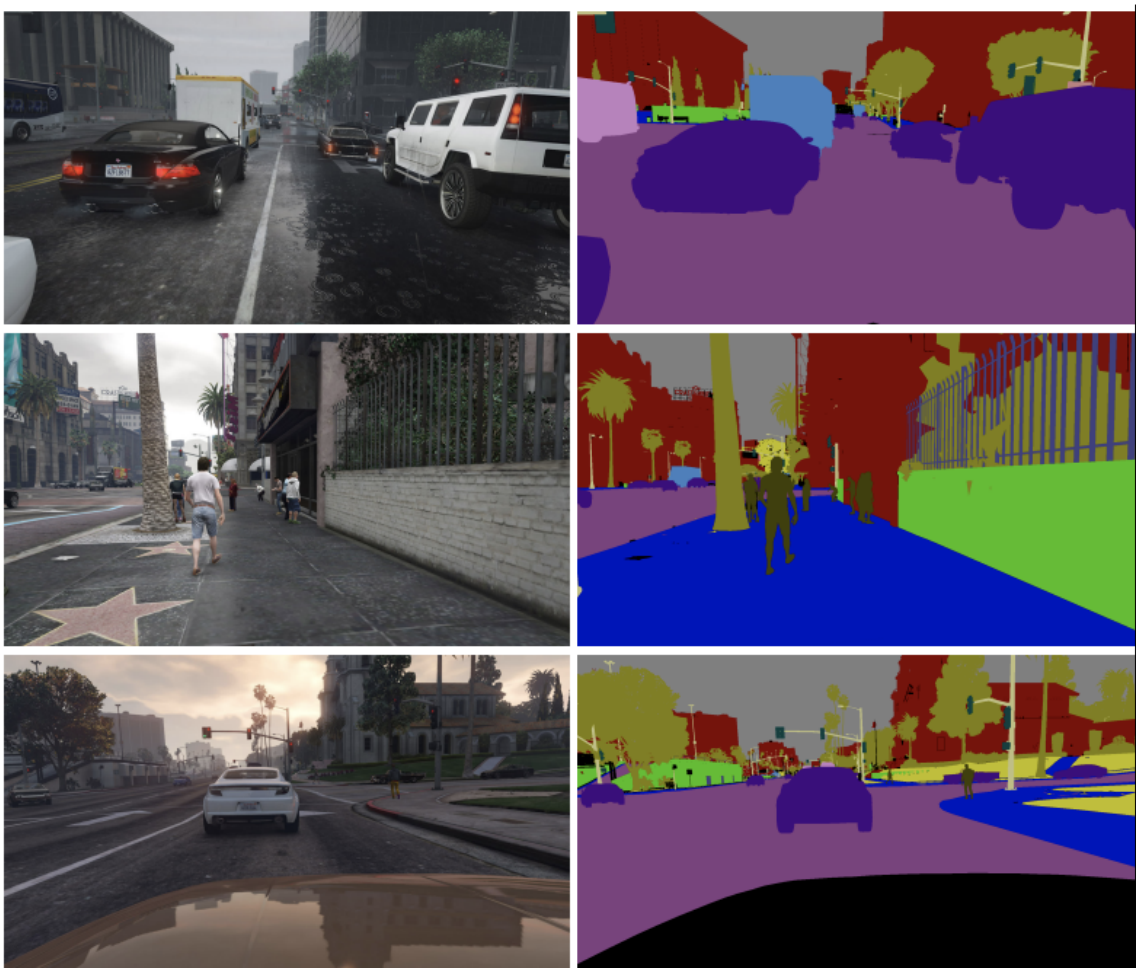


Figure 2.42: Examples of GTA5 images and ground truth labels.

Images are acquired from multiple-points, by letting several cameras move randomly in the city, at at a height from the ground ranging between [1.5m, 2m]. In addition, to increase the visual variability, each camera is forced to stay at least 10 meters far from all the other cameras.

2.6.3 Cityscapes

Cityscapes [2] is a large-scale benchmark dataset, designed to capture the high variability of outdoor street scenes.
Images have been collected letting a vehicle move in 50 cities, mainly in Germany. The process lasted few months, making it possible to collect image during 3 different seasons: spring, summer and fall.
For what concern the annotation, 5000 images from 27 cities have been selected for

dense pixel wise annotations. It is worth notice that, on average, annotation and quality control required 90 minutes, for each single image.
From the remaining 23 cities, other 20.000 frames were extracted. To reach a trade-off between quality and speed, these images were provided of coarse pixel-wise annotations, which required a span of maximum 7 minutes.

The 5000 densely annotated images have been subdivided in 3 sets, to support training, validation and test. In particular, to make each split reflect the variability of each street scene, the dataset has been subdivided to equally share frames from:

- Large, medium and small cities
- Geographic west, center and east
- Geographic north, center and south
- Beginning, middle and end of the year

At the end of this split process, we end up having 2975 images for training, 500 for validation and the remaining 1525 images for test.

Chapter 3

Benchmark

The methods described in section 2.5 attempt to bridge the gap between synthetic and real domains, by making use of the most frequently used Domain Adaptation techniques that can be subdivided in Adversarial-based or Self-Learning-based (SSL). In these works, and more in general in DA literature, it is common to adopt Domain Adaptation techniques on top of large scale Deep Learning models, since their large number of learnable parameters allows to reach remarkable results in terms of mean Intersection over Union (mIoU) on the target domain.
One of the most adopted networks is DeepLabV2 (see 2.4.1.1). This network is characterized by a huge number of parameters and high latency, but guarantees high quality segmentation outputs on the target domain.
Usually, the segmentation model is trained on the labeled source domain, with an Adversarial or SSL optimization on the unlabelled target domain.

At this point, we'd like to make some considerations.

- Despite being characterized by high performance in terms of mIoU, the complexity of the segmentation network has a significant impact on the inference time. Taking as reference the DeepLabv2 network, the total number of parameters of the model is around 44 million and the average time to process a single image on a NVIDIA-TITAN RTX is 13.75 ms. Even if this latency is undoubtedly very limited, it is not negligible in the real world self-driving scenario.
  In addition, the huge number of parameters of the network has an impact on the hardware requirements needed to run the model inside the vehicle, increasing the design and production costs from manufacturers point of view.
  To face this issues, in the last years researchers have proposed lightweight

models characterized by lower number of learnable parameters, and, consequently, by lower latency. Taking as reference MobileNetV2 with ASPPv2 as head, the total number of parameters is around 3 million, while the average inference time drops down to 6.06 ms .
On the other hand, due to the lower number of parameters of the lightweight model, the capacity of the network is reduced, at the expense of performances.

- The second consideration is about the training of the lightweight network in the Domain Adaptation setting. A straightforward solution could be to simply adopt the aforementioned DA techniques to train the MobileNetV2 segmentation network.
  However, we imagine a different scenario, especially from companies point of view, in which in phase of deployment we expect to integrate the lightweight models on the vehicles, but, in phase of training, we actually have access to the powerful but slower DeepLabV2 network. This scenario pushes us to research a way to take advantage of the fine grained knowledge of DeepLabV2 when training the lightweight MobileNetV2.

- The third point is about the availability of the datasets. Starting from the scenario just described, we imagine a situation in which a company has access to the pre-trained DeepLabV2 network and aims to exploit its fine knowledge to train the lightweight MobileNetV2 on a real target domain, without having access to source dataset originally used to train the DeepLabv2.

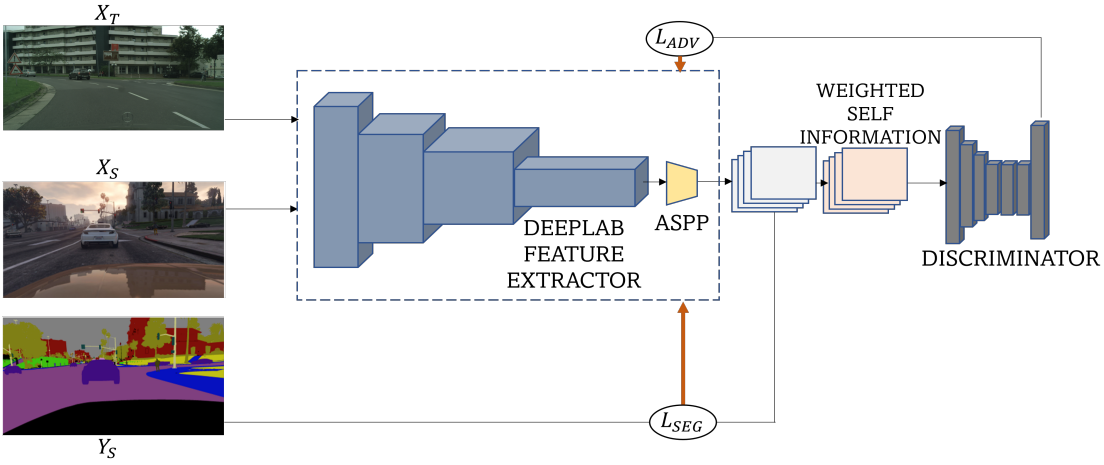
3.1 Protocols

Putting together all these aspects, we can finally introduce our work. We propose to extend three traditional DA settings (Adversarial, Self-Learning, Adversarial+Self-Learning) to exploit the fine grained knowledge of DeepLabV2 to train the lightweight MobileNetV2. Since we expect to move the knowledge from a complex model to the lighter one, we refer to DeepLabV2 as *source model* and to MobileNetV2 as *target model*.
Rather than adopting a pre-trained version of DeepLabV2, in this work we decided to train jointly the source and the target model, exploiting the gradual knowledge of DeepLabV2 to train the lightweight network step after step. To this purpose, we relax the constraint about the not-availability of the source dataset (essential for training DeepLabV2), but we try to reduce as much as possible the dependency of the lightweight model from the images coming from the source domain.
In particular, we begin by extending the conventional Adversarial Domain Adaptation setting to align the features of the target images fed to MobileNetV2 to those extracted by DeepLabV2 on the source images, making it possible to use

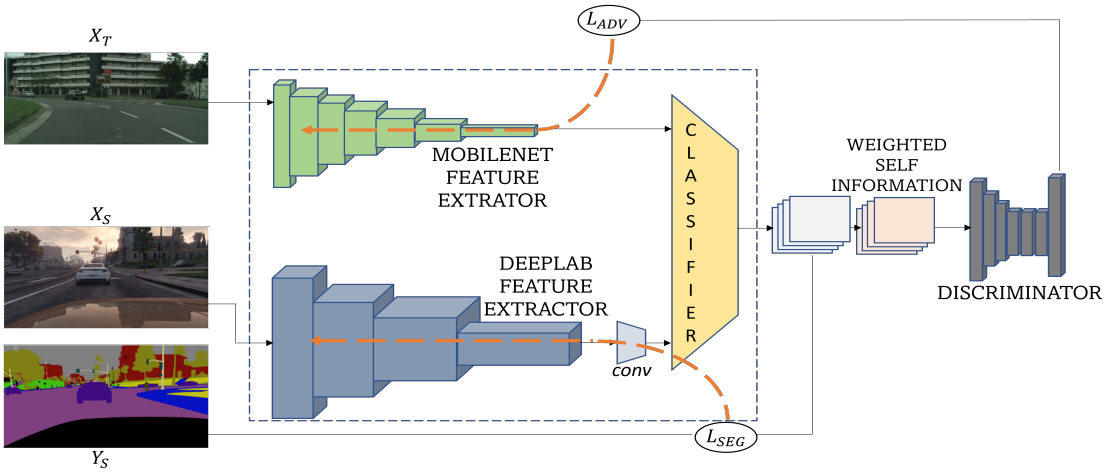
the classifier trained on the source domain to segment the target images fed to the lightweight network. As a result, the classifier (the ASPPv2 module) in this configuration is shared by the DeepLabV2 and MobileNetV2 feature extractors.
We next switch to the Self Learning DA configuration where we suggest optimizing the lightweight network using the pseudolabels produced by the source model. We design a simple SSL framework in which the generation of pseudolabels is based on a threshold on the confidence. In order to make this method comparable with the Adversarial-based one, we maintain the same architecture with the shared classifier. We then extend the existing SSL Domain Adaptation techniques described in section 2.5.2: CBST, FDA, DACS and DAFormer. For this frameworks, we treat the source and the target models as separate networks.
Then, using the architecture with the shared classifier, we train MobileNetV2 by combining the Adversarial and SSL Domain Adaptation approaches. While MobileNetV2's features are aligned using an adversarial loss, DeepLabV2's pseudolabels are used to train the lightweight model on the target domain.

3.1.1 Deep2Light Adversarial

We begin with the Adversarial-based Domain Adaptation scenario, using the ADVENT-proposed architecture described in section 2.5.1.1 as starting point for our **Deep2Light** adversarial framework.
In the original configuration, described in fig. 3.1a, the segmentation network is optimized with two losses. The first one is the segmentation loss, computed on the synthetic images coming from the source domain and the corresponding ground truth labels. The second is the adversarial loss, used to align the weighted self-information distributions of source and target images, by training a discriminator to classify whether an image is coming from source or from target and, at the same time, training the segmentation network to fool the discriminator.



(a) Traditional Adversarial Domain adaptation Setting, with reference to ADVENT.

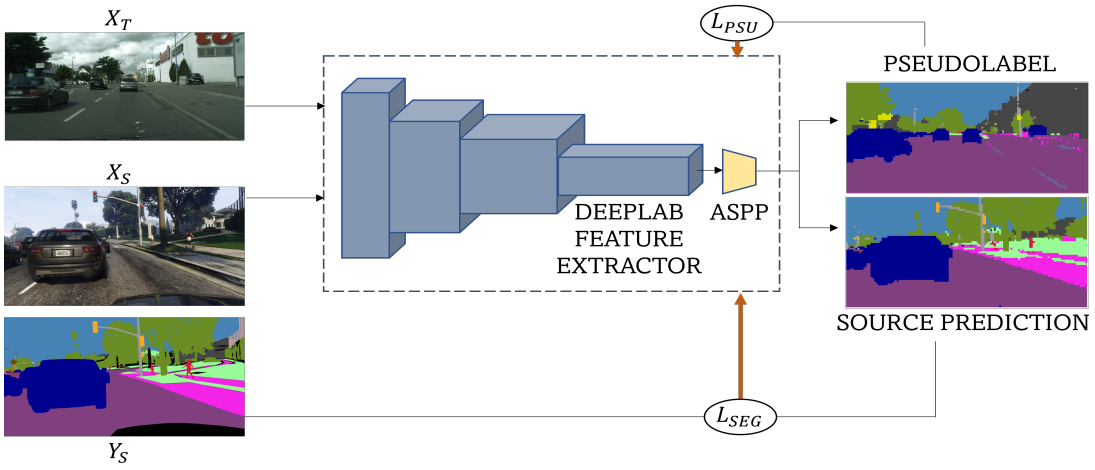


(b) Deep2Light Adversarial setting.

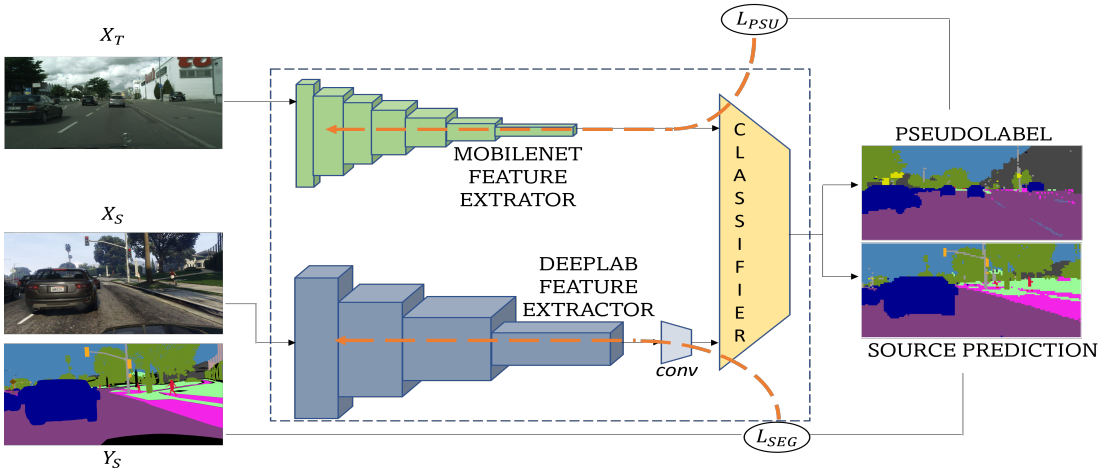
Figure 3.1

Taking inspiration from ADDA (see section 2.5.1), we imagine the possibility of learning, on the source model, a discriminative representation for the source domain along with a unique classifier and, jointly, mapping the target data into the same space of source images through a domain-adversarial loss on the target model.
In particular, we denote the sets of labeled and unlabeled samples respectively as S and T. At each iteration, an image X_S from the source domain S is fed as input to the DeepLabV2 feature extractor M_S, which produces the feature representation $M_S(X_S)$ for the source image. $M_S(X_S)$ is then fed as input to the classifier C, which produces the segmentation output $C(M_S(X_S))$. M_S and C are trained using a standard cross-entropy loss.

In the same iteration, an image X_T is sampled from the target domain T and fed as input to MobileNetV2 feature extractor M_T, which produces the target mapping $M_T(X_T)$. $M_T(X_T)$ is then passed to the classifier C, that generates the corresponding segmentation map $C(M_T(X_T))$
A domain discriminator D receives the weighted self-information computed from $C(M_S(X_S))$ and $C(M_T(X_T))$ and is trained to classify whether the input is drawn from the source or from the target domain, with a cross entropy domain classification loss.
At the same time, with the parameters of the classifier freezed, the feature extractor of MobileNetV2 is trained with an adversarial loss to fool the discriminator. In particular, it feeds the discriminator with the segmented output of the target image but labeled as source. This loss will encourage the feature extractor M_T to align the features of the target images to the ones produced by DeepLabV2 for the source images, making it possible to use the classifier optimized on the source images to segment the target samples.
In this architecture, the head of DeepLabV2 (i.e. the ASPPv2 module) behaves as shared classifier for the two networks. However, since the number of channels of the source and target output features does not match, we insert a 1×1 convolution block on the output features of DeepLabV2 to reduce the number of channels from 2048 to 1280, making it compatible with the output features of MobileNetV2.
The overall architecture is illustrated in fig. 3.1b.



(a) Traditional Self-Learning Domain adaptation Setting.



(b) Deep2Light Self-Learning setting.

Figure 3.2

3.1.2 Deep2Light Self-Learning

Traditionally, in the SSL setting, the segmentation network is trained with a segmentation loss, computed on the annotated images coming from the source domain. In addition, the segmentation network in asked, at the same time or in a following iteration, to generate the pseudo-labels for the target images. These pseudo-labels are used as ground truth labels for the target images to further optimize the segmentation network with a second loss, which we refer to as segmentation pseudo loss.
The traditional SSL framework is illustrated in fig. 3.2a.

Deep2Light SSL-with-Threshold

We modify the framework so that DeepLabV2 is trained exclusively on the source domain, but is still used to generate the pseudo-labels for the target images. In contrast with the traditional approach, now we propose to use the target images with the pseudo-labels generated by the source model to train the lightweight network. Both the feature extractors and the classifier are trained using a standard cross-entropy loss.
In this setting, theoretically, we could separate the source and the target model, but we choose to stick with the previously described architecture, complete with a shared classifier and a convolution block on the output features of DeepLabV2, in order to make the results comparable to those obtained in the adversarial scenario. The overall architecture is illustrated in fig. 3.2b.
For the pseudo-labels generation strategy, we set a threshold on the confidence of the prediction. In particular, for each semantic class we only accept predictions having confidence > 0.968.

Deep2Light CBST

The original work of CBST has been illustrated in section 2.5.2.1.
Synthetically, the whole pipeline can be described as the repetition of several rounds, where each round can be summarized as:

1. Target images are fed into the network, which outputs the predictions with the corresponding confidence map.
2. The confidence maps of the whole set of target images are used to determine the thresholds K_c for each class, which drive the pseudolabel generation for the target set of images.
3. The network is finetuned with the pseudolabeled target images, plus a small portion of source images, for 2 epochs.

In the original work, a pre-trained-in-source version of DeepLabV2 is adopted, reason why, during the CBST pipeline, the model is fed with only a negligible portion of source images.

We propose to use the pseudo-labels generated by DeepLabV2 not only to finetune the weights of the source network, but also to train MobileNetV2 on the target domain. Images from source domain, instead, are forwarded exclusively to DeepLabV2, since we want to reduce as much as possible the use of source images in the training of MobileNetV2.
The overall framework is described in fig. 3.3.

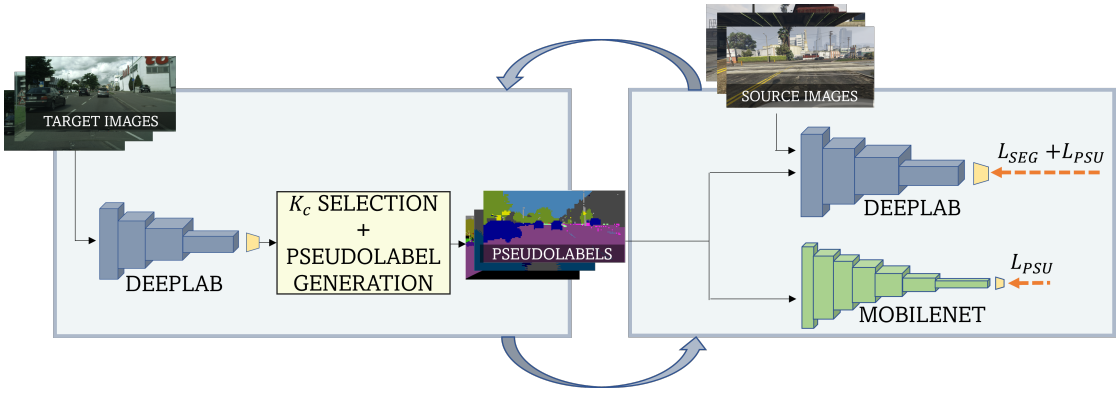


Figure 3.3: Deep2Light CBST. The pipeline can be summarized as the repetition of several rounds, each composed of 2 steps. In the first step, on the left, target images are fed to DeepLabV2 to generate pseudo-labels. In the second step, on the right, DeepLabV2 is finetuned on the labeled source domain and on the pseudo-labeled target images. In addition, the target images, paired with the same pseudolabels generated by DeepLabV2, are used to train MobileNetV2.

Deep2Light FDA

The original approach of FDA has been presented in section 2.5.2.2.
The work starts the observation that even if images are perceptually very close, their low-level spectrum can vary significantly. To make the network able to generalize on the target domain, the proposed strategy consists in training the network with source-in-target images, obtained by replacing the low-level spectrum of source images with the ones extracted from the target samples (see section 2.5.2.2 for details about the generation of source-in-target images).
In this process, to amount of low-level spectrum moved from target to source can be controlled by a parameter, named β, that permits to create multiple datasets, each having a different degree of translation in target.
The overall framework can be subdivided in the following stages:

1. Source-in-target dataset creation. Multiple datasets are created using different values of β.

2. Round 0. In parallel, multiple networks are optimized with a standard Cross Entropy loss, each one on a different adapted dataset composed of source-in-target images.

3. Pseudolabels generation. Target images are fed to the networks and their predictions are averaged to obtain the final pseudo-labels for the set of target images.

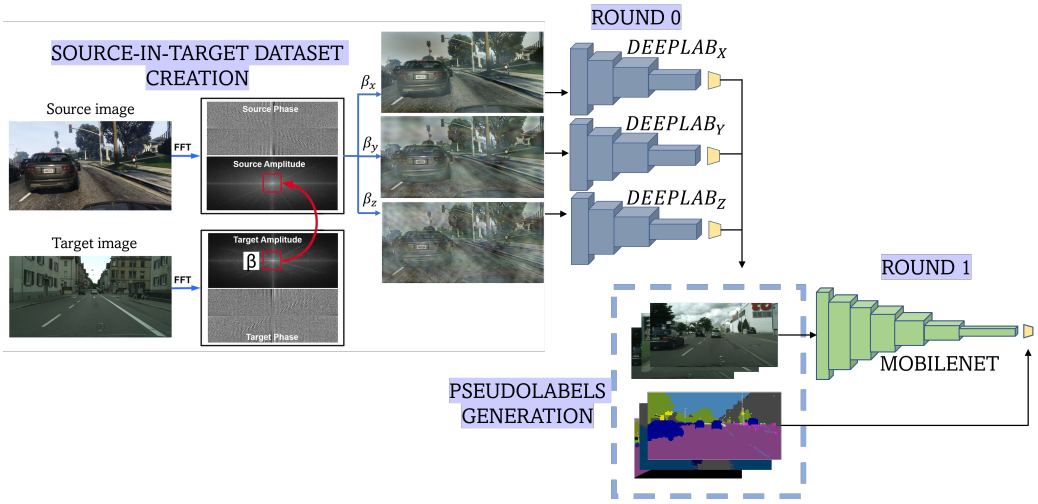


Figure 3.4: Deep2Light FDA. Multiple source-in-target datasets are created, each one with a different value of β, and used to optimize multiple DeepLabV2 models. Once trained, the networks are asked to generate pseudo-labels for the target images, that are later used to train MobileNetV2.

4. Round 1. The networks are trained again from scratch, each one using a different source-in-target dataset plus the target images paired with the generated pseudo-labels.

We propose to extend this framework to use the target images and the pseudo-labels generated by DeepLabV2 during the point 3) to train MobileNetV2 with a standard Cross Entropy loss. This means that the whole process remains unchanged, apart from the round 1. In this round, instead of training DeepLabV2 with the source-in-target dataset plus the pseudo-labeled target images, we use exclusively the target images, paired with the pseudo-labels generated by DeepLabV2, to optimize the lightweight network.
The workflow is described in fig. 3.4.

Deep2Light DACS

The original work of DACS has been presented in section 2.5.2.3.
In this paper, authors affirm that training networks with images mixed from the unlabeled target dataset cause the model to conflate some of the classes in the target domain. Since this problem does not occur when images are sampled from the source domain, they propose Cross-Domain Mixing, adapting the mixing strategy of ClassMix to mix together images from source and target domain (see section

2.5.2.3 for details about the generation of mixed images).
In the previous frameworks, we trained MobileNetV2 exclusively on the target images and the pseudo-labels generated by DeepLabV2, without using not even a source image during the optimization of the lightweight model.
Now, we relax the restriction on MobileNetV2's complete independence from the source dataset, giving access to the mixed images to benefit from DeepLabV2's developing capacity to segment target images.
In particular, we modify the original pipeline of DACS to exploit the mixed images in order to train MobileNetV2.

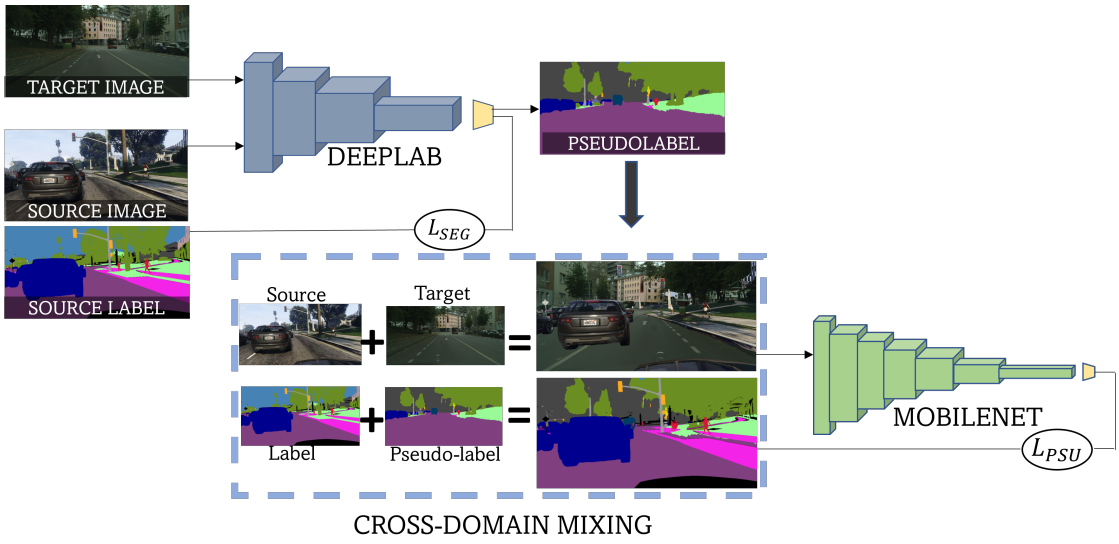


Figure 3.5: Deep2Light DACS. At each iteration, a labeled image is sampled from the source domain and used to train DeepLabV2. An unlabeled image from the target domain is sampled and fed to DeepLabV2, which generates the corresponding pseudo-label. Images and labels are mixed together and used to optimize MobileNetV2.

At each iteration, the pipeline is then modified as follows:

1. Train with source. DeepLabV2 is optimized on the source domain.
2. Mixing. Half of the classes of a source image are selected and the corresponding pixels are pasted into a selected target image to obtain the mixed image. DeepLabV2 receives the target image and is asked to generate a pseudo-label, that is combined with the source label with the same mask to obtain the mixed label.
3. Train with mixed images. MobileNetV2 is optimized with the pseudolabeled mixed images with a standard Cross Entropy loss.

The architecture is described in fig. 3.5.

Deep2Light DAFORMER

DAFormer has been detailed in section 2.5.2.4.
The primary idea of this work is the adoption of a new architecture, called DAFormer, that is built on attention-based Transformers (see section 2.5.2.4 for details about DAFormer architecture). In the paper, authors propose three strategies to limit overfitting on the source domain: Rare Class Sampling (RCS), Thing-Class ImageNet Feature Distance (FD), Learning Rate Warmup. During training, the Cross-Domain Mixing strategy of DACS (see 2.5.2.3) is adopted to augment images of the target domain.
We extend the original framework to integrate MobileNetV2 in the training process. In specifically, using the extension of DACS in section 3.5 as a guide, we train MobileNetV2 using the mixed images and related mixed labels, that were originally utilized to optimize DAFormer.

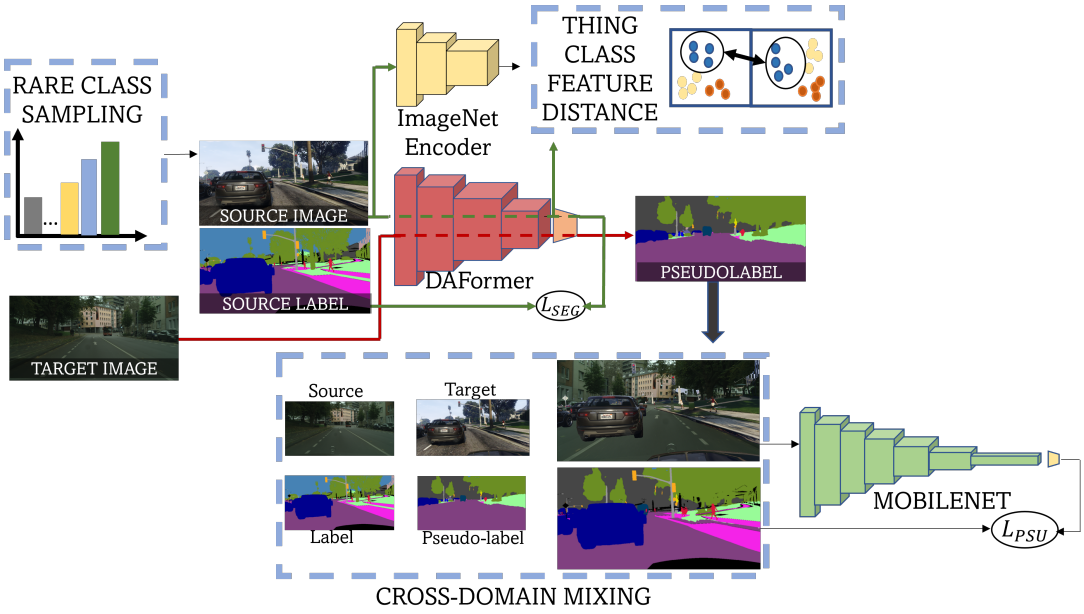


Figure 3.6: Deep2Light DAFormer.

The whole workflow, described in fig. 3.6, can be summarized as the repetition of the following steps:

1. **Rare Class Sampling**. To each class c is associated the frequency f_c, corresponding to the number of pixels of class c in the source dataset. Computing

the sampling probability of each class c as:

$$P(c) = \frac{e^{\frac{(1-f_c)}{T}}}{\sum_{c'=1}^{C} e^{\frac{(1-f_{c'})}{T}}}$$

it is possible to sample, at each iteration, a class c from the probability distribution $P(c)$. A source image is sampled from the subset of images containing the selected class c.

2. **Train DAFormer in source**. DAFormer is trained with the source image and its label with a standard Cross Entropy loss.

3. **Thing-Class ImageNet Feature Distance (FD)**. The distance $d^{(i,j)}$ between the features of DAFormer and the features of the model pretrained on ImageNet is used to compute the Feature Distance loss $\mathcal{L}_{FD}$ on the *thing-classes* identified by the mask M_{thing}:

$$\mathcal{L}_{FD}^{(i)} = \frac{\sum_{j=1}^{H_F \times W_F} d^{(i,j)} \cdot M_{things}^{(i,j)}}{\sum_j M_{things}^{(i,j)}}$$

$\mathcal{L}_{FD}$ is backpropagated in DAFormer.

4. **Cross-Domain Mixing**. To create the mixed image, half of the classes of the source image are chosen, and the associated pixels are pasted into a sampled image from the target domain. When given the target image, DAFormer is asked to create a pseudo-label, that is combined with the source label using the same mask in order to create the mixed label.

5. **Train MobileNetV2 with mixed images**. MobileNetV2 is optimized on the pseudolabeled mixed image with a standard Cross-Entropy loss.

The whole pipeline is described in fig. 3.6.

3.1.3 Deep2Light Adversarial+SSL

In the last experiment, we combine the Adversarial and Self-Learning settings into a unique framework, using the shared-classifier-architecture adopted in the Adversarial and SSL-with-threshold approaches. While DeepLabV2 is trained on the source domain, the lightweight model is optimized with two losses. The first one is the segmentation pseudo loss, computed on the target images and the pseudo-labels generated by the source model. This loss optimizes MobileNetV2's feature extractor and the shared classifier. With the parameters of the classifier freezed, the adversarial loss optimizes MobileNetV2, encouraging it to produce,

for the target images, features more and more similar to the ones generated by DeepLabV2 for the source domain.
The architecture is described in fig. 3.7.

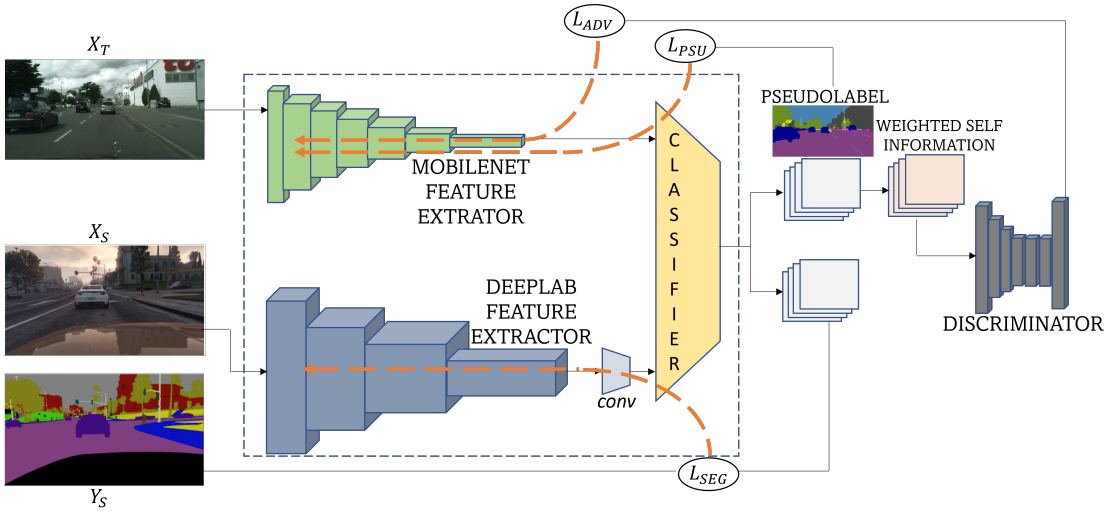


Figure 3.7: Deep2Light Adversarial+Self-Learning.

3.2 Implementation details

This section details the experimental setup for our experiments. All of the experiments are carried out by applying the two conventional synthetic-to-real protocols: GTA5 to Cityscapes and SYNTHIA to Cityscapes.

Deep2Light Adversarial

We train the shared-classifier-architecture for 90.000 iteration with SYNTHIA and for 120.000 with GTA5. Images from GTA are resized to 1280×720, those from SYNTHIA to 1280×760 and the target images from Cityscapes to 1024×512. We use *batch size* = 1 for both the source and target samples. The segmentation networks are optimized via Stochastic Gradient Descent, with learning rate=2.5e-4, momentum=0.9 and weight decay=0.0005, while the discriminator is trained using Adam as optimizer, with learning rate=1e-4.
The adversarial loss used to optimize the lightweight model is weighted by the parameter λ_{adv}. We perform two experiments, setting $\lambda_{adv} = 0.001$ and $\lambda_{adv} = 1$.

Deep2Light Self-Learning with Threshold

We optimize the shared-classifier-architecture architecture with SYNTHIA and GTA5, respectively for 90.000 and 120.000 iterations, using SGD with learning rate=2.5e-4, momentum=0.9 and weight decay=0.0005. Images from GTA are resized to 1280×720, those from SYNTHIA to 1280×760 and the target images from Cityscapes to 1024×512. We use *batch size* = 1 for both the images coming from source and target domain.
The threshold on the confidence of the predictions for the pseudolabel generation is set to 0.968.

Deep2Light CBST

Following the original work of CBST, we use a pretrained-in-source version of DeepLabV2.
In particular, we train the network with standard Cross Entropy loss on GTA5 (analogously on SYNTHIA) for 120000 iterations, batch size = 4, SGD as optimizer with learning rate=2.5e-4, momentum=0.9 and weight decay=0.0005.
The backbone of MobileNetV2 is pretrained on ImageNet.
Images coming from source domain, either GTA5 or SYNTHIA, are resized, randomly scaled, cropped and randomly flipped.
505 images from the train set of Cityscapes are selected and used in the training process as target images. During this process, images are neither resized, cropped or scaled.
At each round, the $p \times 100\%$ most confident pseudo-labels for each class are selected (see section 2.5.2.1 for details). The parameter p starts from an initial value and is gradually increased at each round.
Once the images are pseudolabeled by DeepLabV2, the 505 target images are loaded again, with their pseudolabels. In particular, after being randomly resized, a hard sample mining strategy is applied to give priority, during cropping, to the portion of images containing rare classes, where the rare classes are the worst 3 classes extracted from the target predictions.
The portion of source images used to finetune DeepLabV2, although very limited, gradually increases round after round.
We perform hyperparameter tuning in order to find the most suitable number of epochs per round for the optimization of MobileNetV2, which we discover to be 24. Table 3.1 lists the parameters utilized for the training.

Deep2Light FDA

Both the backbones of DeepLabV2 and MobileNetV2 are pretrained on ImageNet.
During the first round, named *round 0*, we perform 3 experiments, each training

	GTA5	**SYNTHIA**
RESIZE SOURCE IMAGES	1914 × 1052	1280 × 760
RESIZE TARGET IMAGES	2048 × 1024	2048 × 1024
RANDOM SCALE SOURCE	0.5-1.5	0.8-1.2
RANDOM SCALE TARGET	0.5-1.5	0.6-1.5
CROP SIZE OF SOURCE IMAGES	1024 × 512	900 × 600
OPTIMIZER	Adam	Adam
LEARNING RATE	5e-5	5e-5
MOMENTUM	0.9	0.9
WEIGHT DECAY	0.0005	0.0005
BATCH SIZE	2	2
NUMBER OF ROUNDS	4	3
EPOCHS PER ROUND - DEEPLAB	2	2
EPOCHS PER ROUND - MOBILENET	24	24
INITIAL VALUE OF p	0.2	0.2
STEP OF p	0.05	0.05
MAXIMUM VALUE OF p	0.5	0.5
INITIAL PORTION OF SOURCE	3%	2%
PORTION STEP OF SOURCE	0.25%	0.25%
MAX PORTION OF SOURCE	6%	6%

Table 3.1: Parameters of Deep2Light CBST.

DeepLabV2 with a different β, which determines the degree of translation in target. Source images are cropped and resized, whereas target images are just resized.
Once the models are trained, they are used to generate the pseudolabels for the train set of Cityscapes. The predictions of the models are averaged and, for each pixel, the semantic class is retained only if its confidence is within the top 66% or above 0.9.
During the second round, named *round 1*, target images and pseudolabels are loaded, resized and used to train MobileNetV2 with a Cross Entropy loss.
Parameters are reported in tab. 3.2.

Deep2Light DACS

Both the backbones of DeepLabV2 and MobileNetV2 are pretrained on ImageNet. Source images, coming from GTA5 or SYNTHIA, and target images, coming from the train set of Cityscapes, are resized and cropped.
DeeplabV2 is trained on the labeled source images with Cross Entropy loss.

	GTA5 / SYNTHIA
RESIZE SOURCE IMAGES	1024 × 512 / 1280 × 760
RESIZE TARGET IMAGES	1280 × 720
β	0.01, 0.05, 0.09
NUMBER OF STEPS	100000
OPTIMIZER	SGD
LEARNING RATE	2.5e-4
MOMENTUM	0.9
WEIGHT DECAY	0.0005
BATCH SIZE	1

Table 3.2: Parameters of Deep2Light FDA.

For the pseudo-labels generation strategy, we set a threshold on the confidence of the prediction. In particular, for each semantic class we only accept predictions having confidence > 0.968.
Images are mixed with ClassMix, and then augmented by Color Jitter and Gaussian Blur.
MobileNetV2 is trained on the pseudolabeled mixed images with a Cross Entropy loss.
Parameters are reported in tab. 3.3.

	GTA5 / SYNTHIA
RESIZE SOURCE IMAGES	1280 × 720
RESIZE TARGET IMAGES	1024 × 512
CROP SIZE	512 × 512
COLOR JITTER	TRUE
GAUSSIAN BLUR	TRUE
BATCH SIZE	2
NUMBER OF STEPS	250000
OPTIMIZER	SGD
LEARNING RATE	2.5e-4
MOMENTUM	0.9
WEIGHT DECAY	0.0005

Table 3.3: Parameters of Deep2Light DACS.

Deep2Light DAFormer

The encoder of DAFormer is MiT-B5 [49], pretrained on ImageNet-1k. The backbone of MobileNetV2 is also pretrained on ImageNet.
Source and target images are resized, cropped and randomly flipped.
Labeled source images are used to train DAFormer with a Cross-Entropy loss, while the bottleneck features are used to compute the Feature Distance loss $\mathcal{L}_{\mathcal{FD}}$, that is weighted by the factor λ_{FD} in the DAFormer optimization process.
For the pseudo-labels generation strategy, we set a threshold on the confidence of the prediction. In particular, for each semantic class we only accept predictions having confidence > 0.968.
In the Cross-Domain Mixing, images are mixed with ClassMix and then augmented with Color Jitter and Gaussian Blur.
MobileNetV2 is trained on the pseudolabeled mixed images with Cross Entropy loss.
Parameters are reported in tab. 3.4.

		GTA5 / SYNTHIA
	RESIZE SOURCE IMAGES	1280 × 720 / 1280 × 760
	RESIZE TARGET IMAGES	1024 × 512
	CROP SIZE	512 × 512
	RANDOM FLIP	TRUE
	ITERATIONS	40000
	BATCH SIZE	2
CROSS-DOMAIN MIXING	**COLOR JITTER**	TRUE
	GAUSSIAN BLUR	TRUE
RARE CLASS SAMPLING	**T**	0.01
FEATURE DISTANCE	**r**	0.75
	λ_{FD}	0.005
LR WARMUP	**OPTIMIZER**	AdamW
	LEARNING RATE	6e-5
	WARMUP PERIOD	1500
	WEIGHT DECAY	0.01

Table 3.4: Parameters of Deep2Light DAFormer.

Deep2Light Adversarial+Self-Learning

We train the shared-classifier-architecture for 90.000 iteration with SYNTHIA and for 120.000 with GTA5. Images from GTA are resized to 1280×720, those from SYNTHIA to 1280×760 and the target images from Cityscapes to 1024×512. We use *batch size* = 1 for both the source and target samples. The segmentation networks are optimized via Stochastic Gradient Descent, with learning rate=2.5e-4, momentum=0.9 and weight decay=0.0005, while the discriminator is trained using Adam as optimizer, with learning rate=1e-4. The adversarial loss used to optimize the lightweight model is weighted by the parameter λ_{adv}. We perform 2 experiments, setting $\lambda_{adv} = 0.001$ and $\lambda_{adv} = 1$.
The threshold on the confidence of the predictions for the pseudolabel generation is set to 0.968.

3.3 Results

We established few baselines before reporting the outcomes of our experiments. To define a lower bound for our framework, we first train MobileNet on the source domain (GTA and SYNTHIA) and then test the model on the target domain.
As upper bound, we directly train MobileNet on the training set of the Cityscapes dataset.
Results are reported in table 3.5.
In table 3.6 we report the outcomes of the first Deep2Light frameworks: Adversarial and SSL-with-threshold.
The results unequivocally show a significant imbalance in MobileNetV2's performance when trained using the two alternative paradigms.
We originally choose $\lambda_{adv} = 0.001$ in the adversarial scenario, sticking to the basic strategy of ADVENT[29], which served as the basis for our research. In contrast to ADVENT, where the reference network is optimized using both the adversarial and segmentation losses, in our framework the target network (i.e. MobileNetV2) is trained purely using adversarial loss. Starting from this observation, we increased the weight of the adversarial loss, by setting $\lambda_{adv} = 1$. Despite obtaining lower performance in terms of mIoU, in this second experiment the amount of correctly assigned pixels is better distributed over classes, even if several classes are still completely ignored. Since MobileNetV2 does not even reach the lower bound set for GTA5 and SYNTHIA, neither with $\lambda_{adv} = 0.001$ and $\lambda_{adv} = 1$, we can affirm that the adversarial framework fails in this task.
On the other hand, by adopting the Self-Learning framework, we achieve promising results, outperforming the lower bounds established for GTA5 and SYNTHIA in nearly every class as well as in the final mIoU.
Despite the unsatisfactory outcomes of the adversarial framework, we merge it with

	LOWER BOUNDS		UPPER BOUND
	GTA5	**SYNTHIA**	**Cityscapes**
Road	8.04	33.77	95.28
Sidewalk	4.86	16.37	68.12
Building	52.51	48.52	83.35
Wall	1.02	4.35	34.55
Fence	9.82	0.14	39.55
Pole	5.37	7.71	14.94
Light	7.24	0.00	27.44
Sign	1.14	0.03	35.34
Vegetation	61.16	50.98	83.00
Terrain	3.14	-	49.80
Sky	61.01	63.96	83.99
Person	22.20	28.43	50.10
Rider	0.45	0.59	28.42
Car	16.08	25.21	83.94
Truck	5.54	-	43.59
Bus	2.08	4.91	53.25
Train	0.00	-	35.72
Motorbike	2.15	0.00	29.84
Bicicle	0.00	3.49	51.58
mIoU-19	13.88%	-	52.20%
mIoU-16	-	18.03%	53.92%

Table 3.5: We report the lower bounds established for MobileNetV2 in the two conventional settings GTA5→Cityscapes and SYNTHIA→Cityscapes, as well as the upper bound. Results are shown as per-class IoU and mIoU for 16 and 19 classes.

SSL in the final experiment. Results are reported in table 3.7.

Performances substantially decrease with $\lambda_{adv} = 1$, reaching those observed with the adversarial-only configuration. We try to lessen this effect by lowering the weight of the adversarial loss, setting $\lambda_{adv} = 0.001$. By doing this, we get results that are similar to those of the SSL-only configuration.

In light of this, we disregard the adversarial framework, which doesn't seem to add anything to our framework, and solely concentrate on Self-Learning approaches.

In table 3.8 we report the results of the Deep2Light Self-Learning techniques:

	ADVERSARIAL				**SELF-LEARNING**	
	GTA2CS		SYNTHIA2CS		GTA2CS	SYNTHIA2CS
	$\lambda = 0.001$	$\lambda = 1.0$	$\lambda = 0.001$	$\lambda = 1.0$		
Road	70.96	61.51	64.55	10.60	42.86	56.48
Sidewalk	0.00	0.04	11.93	3.00	13.95	25.51
Building	43.36	32.36	36.83	29.38	75.4	59.13
Wall	0.00	0.03	0.00	0.95	14.37	0.03
Fence	0.00	0.02	0.00	0.03	14.33	0.00
Pole	0.00	0.18	0.00	1.46	5.35	9.71
Light	0.00	0.00	0.00	0.00	6.09	0.00
Sign	0.00	0.00	0.00	0.17	5.03	4.09
Vegetation	0.00	3.55	0.39	0.58	78.04	74.00
Terrain	0.00	0.43	-	-	35.92	-
Sky	0.00	4.17	0.00	0.07	68.32	74.34
Person	0.00	0.01	0.00	0.29	40.03	37.87
Rider	0.00	0.00	0.00	0.00	0.00	4.03
Car	0.00	3.67	0.00	2.91	69.11	70.92
Truck	0.00	0.19	-	-	16.1	-
Bus	0.00	0.00	0.00	0.21	22.17	13.12
Train	0.00	0.00	-	-	0.00	-
Motorbike	0.00	0.00	0.00	0.17	0.00	0.11
Bicicle	0.00	0.00	0.00	0.04	0.00	0.76
mIoU	6.02%	5.59%	7.11%	3.12%	26.69%	26.88%

Table 3.6: Results of our Deep2Light Adversarial and Deep2Light Self-Learning frameworks in the two conventional settings GTA→Cityscapes (GTA2CS in the table) and SYNTHIA→Cityscapes (SYNTHIA2CS in the table).

CBST, FDA, DACS and DAFormer. Our results are quite low when using CBST, even worse than when using SSL-with-threshold, yet they are still greater than the lower bounds established for GTA and SYNTHIA.
We obtain higher performances with FDA, especially in the GTA→Cityscapes setting, where the results are comparable with the ones obtained with DACS, which instead guarantees good quality results also on SYNTHIA→Cityscapes.
However, the technique that unequivocally provides the best results in both the *synthetic2real* settings is DAFormer. This framework outperforms the other SSL

	ADV+SSL			
	GTA→Cityscapes		SYNTHIA→Cityscapes	
	$\lambda_{adv} = 0.001$	$\lambda_{adv} = 1.0$	$\lambda_{adv} = 0.001$	$\lambda_{adv} = 1.0$
Road	62.37	15.21	59.98	43.38
Sidewalk	18.26	4.61	26.43	13.13
Building	73.72	31.57	60.48	41.50
Wall	9.18	0.00	0.00	0.00
Fence	8.27	0.00	0.00	0.00
Pole	4.00	0.00	7.78	0.00
Light	5.23	0.00	0.00	0.00
Sign	6.20	0.00	3.36	0.00
Vegetation	77.95	19.87	73.59	24.07
Terrain	32.89	1.19	-	-
Sky	65.73	5.79	73.37	47.21
Person	39.52	0.21	40.98	0.37
Rider	0.00	0.00	4.73	0.00
Car	68.40	16.96	73.31	4.03
Truck	16.08	0.00	-	-
Bus	9.66	0.00	17.37	0.00
Train	0.00	0.00	-	-
Motorbike	0.00	0.00	0.00	0.00
Bicicle	0.00	0.00	0.37	0.00
mIoU	26.18%	5.02%	27.61%	10.86%

Table 3.7: Results of our Deep2Light Adversarial+Self-Learning framework.

techniques in nearly every class as well as in the final mIoU. In particular, if we compare the results with the upper bounds, we can observe a gap in the mIoU of around 10% in GTA→Cityscapes setting and 15% in the SYNTHIA→Cityscapes one. In light of this, the results obtained with the Deep2Light DAFormer framework are quite encouraging, as we are able to successfully exploit a network trained exclusively on the source domain to approach the upper bound, which, we recall, is established by taking advantage of the target annotations.

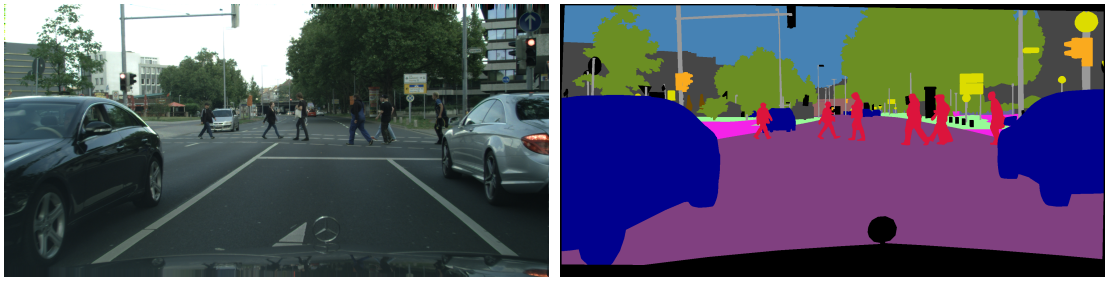
In fig. 3.8 we provide some graphical results of our Self-Learning based Deep2Light techniques in the GTA→Cityscapes setting, which qualitatively confirm

	Deep2Light GTA→CITYSCAPES				Deep2Light SYNTHIA→CITYSCAPES			
	CBST	**FDA**	**DACS**	**DAF**	**CBST**	**FDA**	**DACS**	**DAF**
Road	20.56	89.21	81.31	83.79	16.76	49.80	61.36	77.65
Sidewalk	0.40	42.89	28.82	26.16	7.94	18.19	24.62	28.78
Building	65.09	75.10	79.02	80.03	42.03	64.63	71.55	77.14
Wall	1.24	16.89	22.31	31.33	0.04	0.23	10.25	16.82
Fence	8.20	18.42	24.46	22.45	0.01	0.04	0.02	1.32
Pole	14.90	13.29	8.85	13.19	2.97	11.56	10.21	14.05
Light	0.17	10.44	10.46	23.55	6.16	0.04	0.15	21.68
Sign	1.18	12.49	7.62	17.82	11.08	9.49	6.70	7.39
Vegetation	69.88	74.79	80.20	81.34	68.41	58.05	78.10	78.39
Terrain	28.92	32.68	36.31	36.90	-	-	-	-
Sky	58.97	68.88	79.64	81.77	66.66	70.66	77.00	78.48
Person	36.86	41.08	43.40	48.81	31.24	36.30	45.24	42.45
Rider	13.66	13.58	0.82	25.39	12.80	6.43	8.00	17.71
Car	46.94	71.25	78.35	80.39	59.12	67.32	77.66	79.32
Truck	1.08	18.29	30.55	31.79	-	-	-	-
Bus	0.83	29.67	37.09	45.06	5.88	9.62	19.54	40.65
Train	0.00	0.00	1.40	23.11	-	-	-	-
Motorbike	5.90	13.80	13.80	27.88	1.80	0.24	2.31	11.62
Bicicle	35.24	36.85	18.79	32.35	13.25	27.27	36.39	31.96
mIoU (%)	21.58	35.77	35.96	42.80	21.63	26.87	33.07	39.09

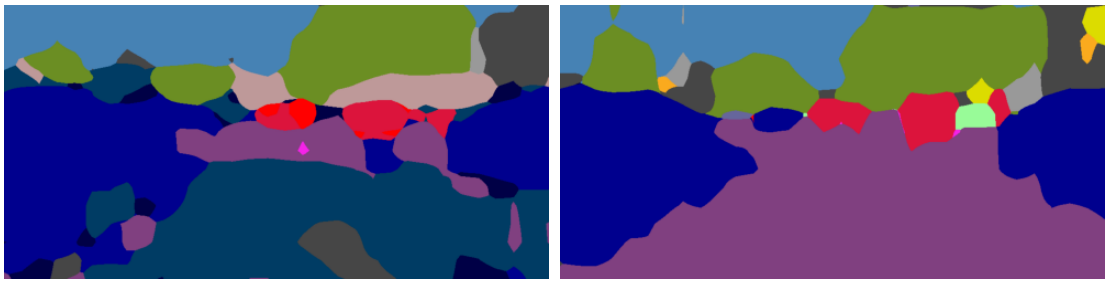
Table 3.8: Results of our Deep2Light i) CBST, ii) FDA, iii) DACS, iv) DAFormer (DAF in the table).

the results reported in table 3.8.

To further reduce the complexity of the lightweight model, and consequently reduce the inference time as well as the hardware requirements needed to run the model, we perform an additional experiment in which we simplify the backbone of MobileNetV2.
In particular, since the second last feature map of MobileNet is composed of only 320 channels instead of 1280, we build the ASSPV2 head on top of this second last feature map, rather than on the original one. We repeat the aforementioned Deep2Light SSL-based experiments (CBST, FDA, DACS and DAFormer) replacing the target network with the simplified MobileNetV2 just described.



(a) Target image **(b)** Ground Truth Label



(c) Deep2Light CBST **(d)** Deep2Light FDA

(e) Deep2Light DACS **(f)** Deep2Light DAFormer

Figure 3.8: Qualitative results of a validation image from Cityscapes, when training models on the GTA5 dataset with the Deep2Light frameworks c) CBST, d) FDA, e) DACS and f) DAFormer.

We report the results in table tab. 3.9.
As we can see from the table, with the lighter MobileNetV2 we obtain similar performances in all the tested Deep2Light SSL-techniques. Therefore, with this simplification of the network, we are able to reduce the latency as well as the number of learnable parameters, which decreases from 3 million to 2 million, with a negligible loss of accuracy.

	Deep2Light GTA→CITYSCAPES				Deep2Light SYNTHIA→CITYSCAPES			
	CBST	**FDA**	**DACS**	**DAF**	**CBST**	**FDA**	**DACS**	**DAF**
Road	18.60	89.44	80.76	79.93	12.87	42.57	59.14	80.34
Sidewalk	1.23	42.85	29.39	28.88	5.42	17.14	23.25	29.14
Building	63.65	75.68	78.32	76.22	42.53	62.41	71.89	77.47
Wall	4.47	15.49	21.34	28.02	0.09	1.20	1.90	11.25
Fence	11.77	20.73	22.82	22.05	0.53	0.27	0.11	1.14
Pole	16.98	13.48	6.47	11.53	3.64	12.39	9.26	15.45
Light	5.77	13.64	10.00	25.59	4.43	0.68	2.76	19.00
Sign	8.13	12.79	7.55	23.24	8.36	10.38	5.98	11.55
Vegetation	70.42	76.46	80.10	81.03	65.18	58.10	77.61	77.66
Terrain	27.24	36.60	37.29	36.47	-	-	-	-
Sky	44.74	67.70	79.61	81.32	67.23	68.24	79.48	78.57
Person	32.42	39.73	42.58	49.37	29.33	37.94	44.04	44.23
Rider	15.53	12.93	3.25	20.20	11.99	9.17	8.18	16.72
Car	29.65	70.65	78.61	80.94	57.65	65.57	77.17	77.93
Truck	1.55	17.31	17.58	31.97	-	-	-	-
Bus	0.66	21.65	31.07	44.43	12.46	8.91	15.95	37.19
Train	0.00	0.34	0.00	16.37	-	-	-	-
Motorbike	1.13	9.03	14.34	27.04	0.22	1.15	2.68	10.71
Bicicle	34.24	33.78	22.35	38.53	14.48	26.96	24.46	32.60
mIoU (%)	20.43	35.28	34.92	42.27	21.03	26.45	31.49	38.81

Table 3.9: Results of the simplified MobileNet with our Deep2Light i) CBST, ii) FDA, iii) DACS, iv) DAFormer (DAF in the table).

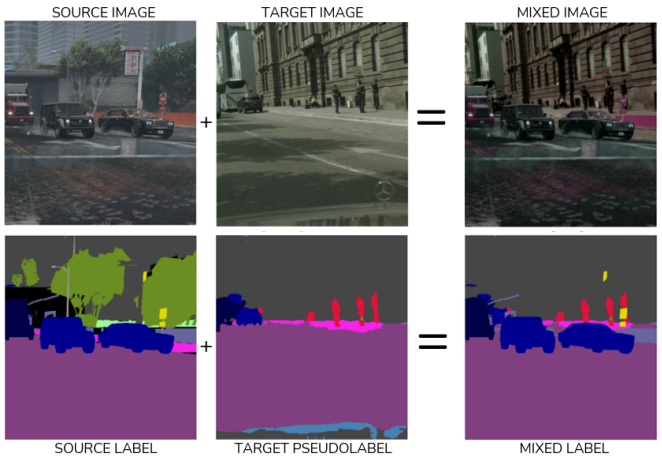
Chapter 4

Extension

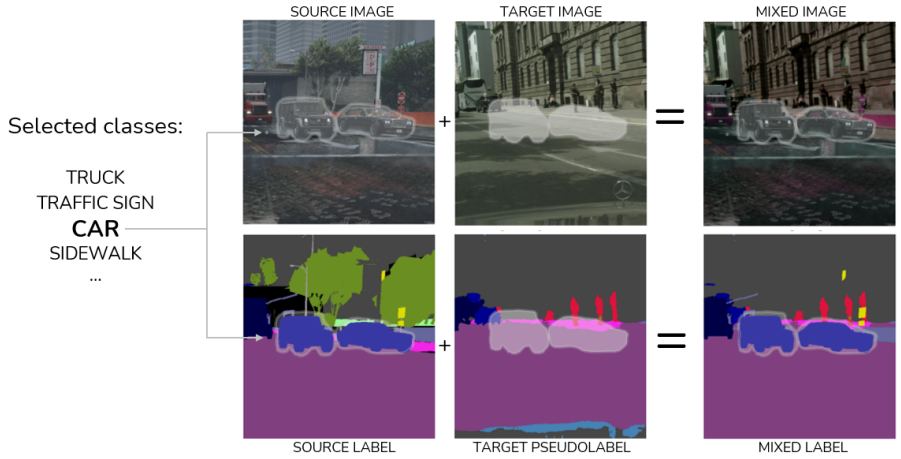
Starting from the promising results obtained with the Deep2Light DAFormer framework, we propose a new method to further improve the performances of the lightweight network.
In particular, we propose a novel mixing strategy for self-learning which takes advantage of the several instances for each class and re-weights the loss according to the network confidence.
In the first part of this section, we are going to revise the mixing strategy adopted in DAFormer as well as the standard Cross-Entropy loss used in the optimization process of MobileNetV2, and we will describe how to extend them them to improve the perfomances of the target model. The second and the third part, respectively, detail the experimental setup for our experiments and the results.

4.1 Method

Following the original approach of DAFormer [40], in the benchmark we mixed together source and target images via Cross-Domain Mixed Sampling [36].
Synthetically, this technique consists in selecting half of the classes from an image sampled from the source domain and pasting the corresponding pixels into an image coming from the target domain. The mixed label is constructed in the same manner, combining the source label with the pseudolabel generated for the target sample (see section 2.5.2.3 for details about Cross-Domain Mixed Sampling). A graphical representation is depicted in fig. 4.1.



(a) Cross-Domain Mixing proposed in DACS.



(b) For the mixing, half of the classes are selected from the source image. For example, one of the selected classes is *car*. Pixels corresponding to *cars* in the source image are copied and pasted into the target image. In the same manner, pixels corresponding to *cars* in the source label are pasted into the target pseudolabel.

Figure 4.1

We propose a novel mixing strategy, which is conditioned on the *instances* identified in the source image rather than on the *classes*.
Before detailing our strategy, we should delineate the difference between the concept of *class* and *instance*. When we refer to *class* we treat multiple objects within a single category as one entity. On the contrary, when we move to the concept of *instance*, we identify the individual objects within the same category as separate entities. Fig. 4.2 shows this difference through an example.
In this work, we propose to move from the original *class selection* of DACS to a more sophisticated *instance selection*.

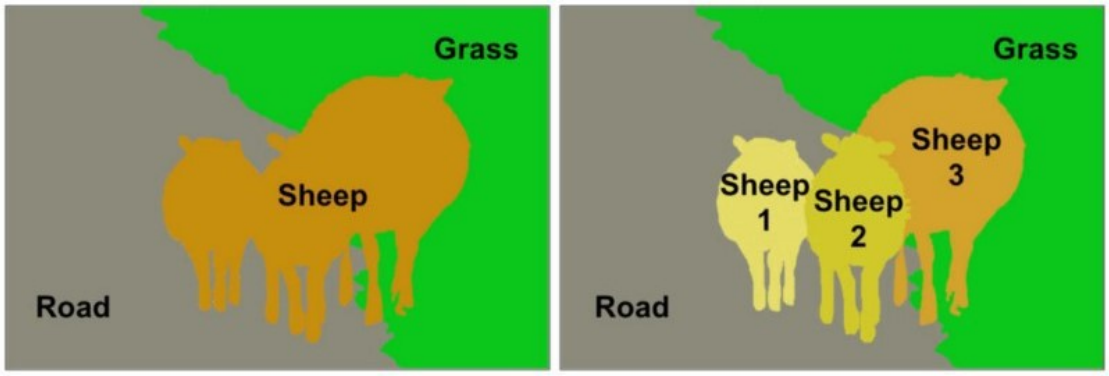


Figure 4.2: On the left, three different classes are identified in the image: *road*, *sheep*, *grass*. On the right, the class *sheep* is segmented into individual entities, each representing a different instance.

In particular, given an image sampled from the source domain, we identify the set of instances within each category. Then, we select half of the instances from the source image and we paste the corresponding pixels into a sampled target image. The mixed label is obtained applying the same mask on the source label and pasting the corresponding pixels into the target pseudo-label generated by the source model.
A graphical representation is depicted in fig. 4.3.
In addition, we propose to re-weight the pseudo segmentation loss in the optimization process of the lightweight network.
In all the Deep2Light Self-Learning-based techniques explored in the benchmark, we trained MobileNetV2 with a standard Cross Entropy loss:

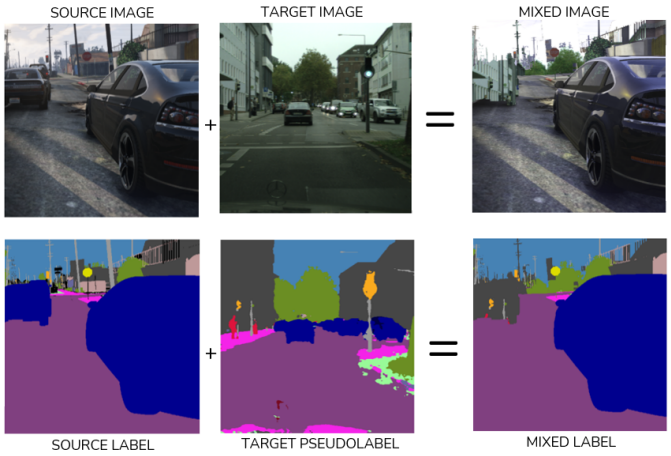
$$\mathcal{L}_{\mathcal{PSU}}(\mathbf{X_T}, \hat{\mathbf{Y}}_\mathbf{T}) = -\sum_{h=1}^{H}\sum_{w=1}^{W}\sum_{c=1}^{C} \hat{\mathbf{Y}}_\mathbf{T}^{(h,w,c)} \cdot \log \mathbf{P_{X_T}}^{(h,w,c)}$$

where $\mathbf{X_T}$ and $\hat{\mathbf{Y}}_\mathbf{T}$ are respectively the target image and its pseudolabel and $\mathbf{P_{X_T}}$ the lightweight network prediction for $\mathbf{X_T}$.
We propose to re-scale the weight given to each class on the basis of the confidence of MobileNetV2's predictions on the mixed images:

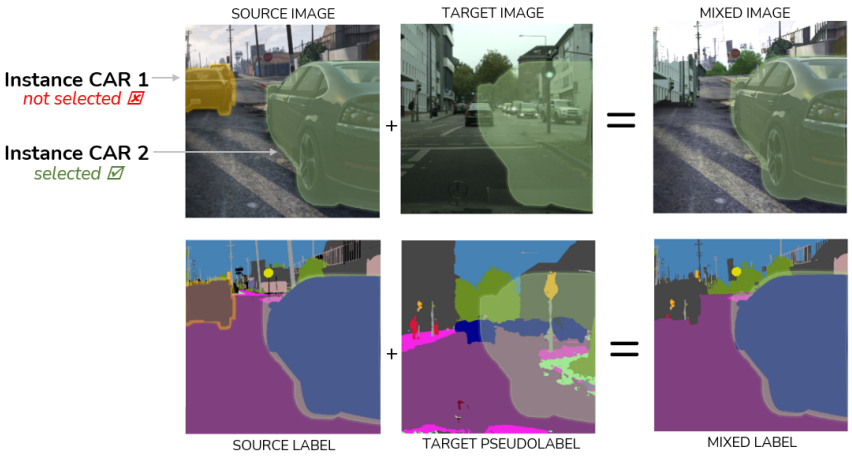
$$\mathcal{L}_{\mathcal{PSU}}(\mathbf{X_T}, \hat{\mathbf{Y}}_\mathbf{T}) = -\sum_{h=1}^{H}\sum_{w=1}^{W}\sum_{c=1}^{C} w_c \cdot \hat{\mathbf{Y}}_\mathbf{T}^{(h,w,c)} \cdot \log \mathbf{P_{X_T}}^{(h,w,c)}$$

where w_c is the confidence associated to the class c.
In particular, the *confidence* is a measure of the ability of the network to represent each pixel. The more the network is able to correctly classify the pixels of class c, the lower will be the contribution of the loss on this class. On the opposite, if the

network has an high uncertainty on c, the loss on that class will be weighted more. In this sense, the confidence measures the *certainty* of the predictions.



(a) Our Cross-Domain Instance-based Mixing.



(b) The individual instances within each class are identified in the source image. For example, in the source image, the whole class *car* can be decomposed into two instances. For the mixing, half of the instances are selected and the corresponding pixels are pasted into the target sample. In this example, only one of the two cars has been chosen in the instance selection.

Figure 4.3

4.2 Implementation Details

We recognize instances in the source image through an algorithm which labels connected regions of integers arrays. In particular, two pixels are *connected* if they are neighbors and have the same value. The concept of neighbors is not

absolute, but depends on an hyperparameter which controls the maximum number of orthogonal hops that we accept between two pixels having the same value. In our experiments, we set this hyperparameter equal to 1. All the background pixels are ignored in this algorithm and labeled as 0.
To re-weight the loss, at each iteration we extract the probability $p_i^c(x)$ for each predicted class c at pixel i. Denoting as N_c the number of pixels predicted as c, the confidence w_c for each class c is computed as:

$$w_c = 1 - \frac{1}{N_c} \sum_{i=1}^{N_c} p_i^c(x)$$

The other parameters used during training are the ones reported in tab. 3.4.

4.3 Results

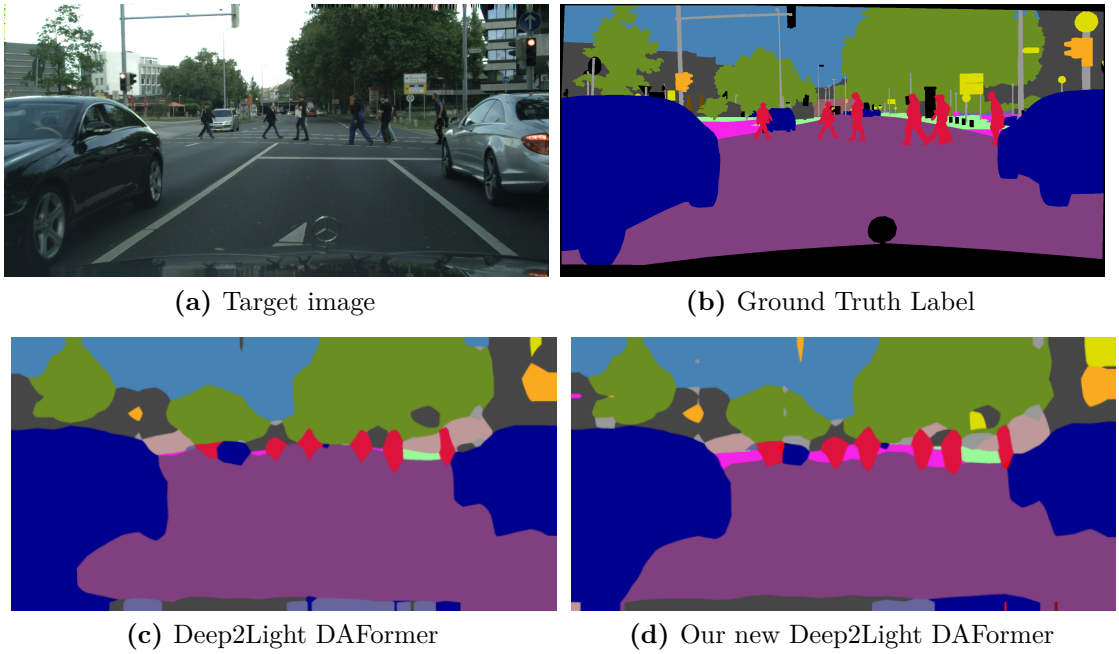
In tab. 4.1 we report the results obtained with our new Deep2Light DAFormer framework, as well as the original results of our Deep2Light DAFormer (see section 3.2) and the upper bound established for MobileNetV2.
We can see that this framework outperforms the standard Deep2Light DAFormer in nearly every class as well as in the final mIoU, in both the settings.
In particular, in GTA→Cityscapes, we obtain an improvement of 1.68% with respect to the standard Deep2Light DAFormer setting. Compared with the upper bound, we are able to reduce the gap from 9.4% to 7.7%.
In SYNTHIA→Cityscapes, results show an improvement in the mIoU of 2.5% and, with respect to the upper bound, a consequent gap reduction from 15% to 12.5%.
These results are qualitatively confirmed in Fig. 4.4, where our new Deep2Light DAFormer provide a better adaptation for small and rare classes, such as *traffic sign* or *traffic light*.
We also test, separately, Deep2Light DAFormer with Instance-based Cross-Domain Mixing Strategy and Deep2Light DAFormer with confidence-based re-weighted loss, and compare it with our proposed Deep2Light DAFormer where the two techniques are combined.
We report these intermediate results for GTA→Cityscapes in table 4.2 and for SYNTHIA→Cityscapes in table 4.3.

For what concerns the first protocol, GTA→Cityscapes, we can see that, with respect to the original Deep2Light DAFormer, the two techniques (i.e. instance-based mixing and confidence-based weghted loss) enable a marginal improvement, that does not exceed the 1%. By combining the two, instead, we can observe a boost in the final mIoU of around 1.68%, and a relative improvement in almost all the classes.

	GTA→Cityscapes		**SYNTHIA→Cityscapes**		
	Standard Deep2Light DAFormer	**Our Deep2Light DAFormer**	Standard Deep2Light DAFormer	**Our Deep2Light DAFormer**	Upper Bound
Road	83.79	83.60	77.65	79.68	95.28
Sidewalk	26.16	28.21	28.78	31.12	68.12
Building	80.03	81.04	77.14	79.19	83.35
Wall	31.33	31.94	16.82	20.59	34.55
Fence	22.45	24.81	1.32	3.33	39.55
Pole	13.19	17.82	14.05	18.63	14.94
Light	23.55	29.19	21.68	26.26	27.44
Sign	17.82	23.01	7.39	12.49	35.34
Vegetation	81.34	81.39	78.39	79.11	83.00
Terrain	36.90	35.86	-	-	49.80
Sky	81.77	81.12	78.48	79.63	83.99
Person	48.81	50.26	42.45	43.44	50.10
Rider	25.39	27.25	17.71	20.48	28.42
Car	80.39	79.26	79.32	74.98	83.94
Truck	31.79	28.71	-	-	43.59
Bus	45.06	48.48	40.65	39.44	53.25
Train	23.11	26.41	-	-	35.72
Motorbike	27.88	28.50	11.62	16.13	29.84
Bicicle	32.25	38.21	31.96	40.84	51.58
mIoU-19	42.80%	44.48%	-	-	52.20%
mIoU-16	-	-	39.09%	41.58%	53.92%

Table 4.1: Results of our new Deep2Light DAFormer. We report also the results of the standard Deep2Light DAFormer and the upper bound established for MobileNetV2.

In SYNTHIA→Cityscapes, instead, we can observe a marginal increase when applying the Instance-based Cross-Domain mixing but a significant boost with the confidence weighted loss (+2.57%). With this second technique, performances, in terms of final mIoU, are higher than those obtained with our proposed technique. However, if we observe the per-class IoU, we can see that with our framework we obtain higher results in almost all the classes, especially in the difficult classes such



(a) Target image **(b)** Ground Truth Label

(c) Deep2Light DAFormer **(d)** Our new Deep2Light DAFormer

Figure 4.4: Qualitative results of a validation image from Cityscapes, when training models on the GTA5 dataset.

as *wall*, *fence* and *pole*.

	GTA→Cityscapes				
	Standard Deep2Light DAFormer	Confidence-based weighted loss Deep2Light DAFormer	Instance-based Cross-Domain Mixing Deep2Light DAFormer	Our Deep2Light DAFormer	Upper Bound
Road	83.79	83.06	**84.44**	83.60	95.28
Sidewalk	26.16	**30.31**	20.58	28.21	68.12
Building	80.03	80.61	79.96	**81.04**	83.35
Wall	31.33	29.77	**32.61**	31.94	34.55
Fence	22.45	21.79	23.92	**24.81**	39.55
Pole	13.19	17.71	12.63	**17.82**	14.94
Light	23.55	27.19	26.15	**29.19**	27.44
Sign	17.82	20.65	18.68	**23.01**	35.34
Vegetation	81.34	80.58	**82.04**	81.39	83.00
Terrain	**36.90**	35.34	34.75	35.86	49.80
Sky	**81.77**	81.62	80.41	81.12	83.99
Person	48.81	49.59	49.56	**50.26**	50.10
Rider	25.39	25.11	**27.62**	27.25	28.42
Car	**80.39**	80.13	79.67	79.26	83.94
Truck	31.79	**32.48**	30.11	28.71	43.59
Bus	45.06	44.23	46.78	**48.48**	53.25
Train	23.11	19.86	**26.68**	26.41	35.72
Motorbike	27.88	29.10	**32.85**	28.50	29.84
Bicicle	32.25	**38.74**	30.47	38.21	51.58
mIoU-19	42.80%	43.57%	43.15%	**44.48**%	52.20%

Table 4.2: Results of GTA→Cityscapes Deep2Light: i) Standard DAFormer, ii) DAFormer with Confidence-based Weighted loss, iii) DAFormer with Instance-based Cross-Domain Mixing strategy, iv) Our proposed DAFormer, v) Upper bound.

	SYNTHIA→Cityscapes				Upper Bound
	Standard Deep2Light DAFormer	Confidence-based weighted loss Deep2Light DAFormer	Instance-based Cross-Domain Mixing Deep2Light DAFormer	Our Deep2Light DAFormer	
Road	77.65	**82.86**	76.66	79.68	95.28
Sidewalk	28.78	**33.30**	29.21	31.12	68.12
Building	77.14	78.94	77.82	**79.19**	83.35
Wall	16.82	19.15	18.68	**20.59**	34.55
Fence	1.32	2.14	1.69	**3.33**	39.55
Pole	14.05	18.39	15.16	**18.63**	14.94
Light	21.68	**26.51**	23.08	26.26	27.44
Sign	7.39	12.35	7.21	**12.49**	35.34
Vegetation	78.39	78.95	78.71	**79.11**	83.00
Sky	78.48	78.76	78.41	**79.63**	83.99
Person	42.45	42.40	42.97	**43.44**	50.10
Rider	17.71	19.66	19.02	**20.48**	28.42
Car	**79.32**	78.42	78.19	74.98	83.94
Bus	40.65	39.22	**40.86**	39.44	53.25
Motorbike	11.62	**16.77**	14.31	16.13	29.84
Bicicle	31.96	38.75	34.08	**40.84**	51.58
mIoU-16	39.09%	**41.66**%	39.75%	41.58%	53.92%

Table 4.3: Results of SYNTHIA→Cityscapes Deep2Light: i) Standard DAFormer, ii) DAFormer with Confidence-based Weighted loss, iii) DAFormer with Instance-based Cross-Domain Mixing strategy, iv) Our proposed DAFormer, v) Upper bound.

Chapter 5

Conclusions

Summary In this work we attempt to address some of the challenges arising in the Self-Driving scenario related to the adoption of Deep Learning models to perform Unsupervised Domain Adaptation in Semantic Segmentation. In particular, if on one hand deep and complex models guarantee highly accurate predictions, on the other hand they are often overparametrized, impacting on the inference time to process images and on the hardware requirements needed to run the models.

Given this premise, we shift our attention to lighter models, characterized by a significant lower number of parameters, which are clearly more suitable for real-world applications. However, rather than directly training a lightweight model with existing Domain Adaptation techniques, we imagine the possibility, especially from companies point of view, of having access to a deep and complex model at training time, which provides highly accurate predictions. We suggest to exploit the fine-grained understanding of images of the deep model to train our lightweight network, moving knowledge from a complex model to a lighter one.
In particular, we propose a benchmark to test the capability of existing Domain Adaptation algorithms to transfer knowledge from a Deep to a Lightweight model, exploring three conventional DA settings: Adversarial, Self-Learning and Adversarial+Self-Learning.

Results show that the Adversarial-based Deep2Light frameworks fail in the task, while the SSL-based Deep2Light configuration allows to achieve promising results in both GTA→Cityscapes and SYNTHIA→Cityscapes.
Starting from this encouraging result, we further extend the benchmark by adapting several existing SSL-based DA techniques to integrate the lightweight network in the optimization process.
The technique that allows to achieve the higher performances is Deep2Light DAFormer, through which we are able to move knowledge from a powerful

Transformers-based deep architecture to our lightweight network, by using mixed images obtained via Cross-Domain Mixed Sampling.
Starting from Deep2Light DAFormer, we further improve the performances of our lightweight model by proposing a novel mixing strategy for Self-Learning which takes advantage of the several instances for each class and re-weights the loss according to the lightweight network confidence.

Future works This work represents a starting point towards most complex solutions, which include:

- **Weighted Cross-Domain Instance Mixing**. Rather than randomly selecting the instances from the source images, we suppose that driving the instance selection on the basis of the lightweight network per-class confidence could encourage the representation of hard classes.
- **Thing Cross-Domain Instance Mixing**. The presence of several instances deriving from amorphous classes as *sky* or *road* causes, often, the generation of highly unrealistic mixed images. We suppose that considering only the instances of the "thing" classes (as *person*, *car* or *traffic sign*) could have a positive impact on the quality of the generated images.
- **Pretrained Deep Network**. The next step consists in taking advantage of a pretrained-in-source version of the deep and complex network to train the lightweight network, by adapting the Deep2Light techniques to this new scenario.
- **Exploit the knowledge of the lightweight network on target**. Since the deep network is trained exclusively on the source domain, the knowledge on the target that it can offer is limited. On the contrary, the developing knowledge of the lightweight network on the target domain could be exploited to further optimize the network with an appropriate SSL-based technique.

Bibliography

[1] Shijie Hao, Yuan Zhou, and Yanrong Guo. «A Brief Survey on Semantic Segmentation with Deep Learning». In: *Neurocomputing* 406 (Apr. 2020). DOI: 10.1016/j.neucom.2019.11.118 (cit. on p. 2).

[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*. 2016. DOI: 10.48550/ARXIV.1604.01685. URL: https://arxiv.org/abs/1604.01685 (cit. on pp. 3, 31, 51, 52).

[3] Sicheng Zhao et al. *A Review of Single-Source Deep Unsupervised Visual Domain Adaptation*. 2020. DOI: 10.48550/ARXIV.2009.00155. URL: https://arxiv.org/abs/2009.00155 (cit. on pp. 3, 31, 32).

[4] Jun Xie, Martin Kiefel, Ming-Ting Sun, and Andreas Geiger. *Semantic Instance Annotation of Street Scenes by 3D to 2D Label Transfer*. 2015. DOI: 10.48550/ARXIV.1511.03240. URL: https://arxiv.org/abs/1511.03240 (cit. on p. 3).

[5] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. *Playing for Data: Ground Truth from Computer Games*. 2016. DOI: 10.48550/ARXIV.1608.02192. URL: https://arxiv.org/abs/1608.02192 (cit. on pp. 3, 51).

[6] Alan M. Turing. «Computing Machinery and Intelligence A.M. Turing». In: 2007 (cit. on p. 6).

[7] Rikiya Yamashita, Mizuho Nishio, Richard Do, and Kaori Togashi. «Convolutional neural networks: an overview and application in radiology». In: *Insights into Imaging* 9 (June 2018). DOI: 10.1007/s13244-018-0639-9 (cit. on p. 19).

[8] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556 (cit. on pp. 22, 27).

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 22).

[10] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2014. DOI: 10.48550/ARXIV.1411.4038. URL: https://arxiv.org/abs/1411.4038 (cit. on pp. 22, 23).

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597 (cit. on pp. 23, 24).

[12] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. *Learning Deconvolution Network for Semantic Segmentation*. 2015. DOI: 10.48550/ARXIV.1505.04366. URL: https://arxiv.org/abs/1505.04366 (cit. on p. 23).

[13] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1511.00561. URL: https://arxiv.org/abs/1511.00561 (cit. on pp. 23, 24).

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2016. DOI: 10.48550/ARXIV.1612.03144. URL: https://arxiv.org/abs/1612.03144 (cit. on pp. 24, 25).

[15] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. *Pyramid Scene Parsing Network*. 2016. DOI: 10.48550/ARXIV.1612.01105. URL: https://arxiv.org/abs/1612.01105 (cit. on pp. 24, 25).

[16] 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*. 2016. DOI: 10.48550/ARXIV.1606.00915. URL: https://arxiv.org/abs/1606.00915 (cit. on pp. 25–27).

[17] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. «DenseASPP for Semantic Segmentation in Street Scenes». In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3684–3692. DOI: 10.1109/CVPR.2018.00388 (cit. on p. 25).

[18] Fisher Yu and Vladlen Koltun. *Multi-Scale Context Aggregation by Dilated Convolutions*. 2015. DOI: 10.48550/ARXIV.1511.07122. URL: https://arxiv.org/abs/1511.07122 (cit. on p. 25).

[19] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation*. 2016. DOI: 10.48550/ARXIV.1606.02147. URL: https://arxiv.org/abs/1606.02147 (cit. on p. 25).

[20] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. *Rethinking Atrous Convolution for Semantic Image Segmentation*. 2017. DOI: 10.48550/ARXIV.1706.05587. URL: https://arxiv.org/abs/1706.05587 (cit. on p. 26).

[21] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. DOI: 10.48550/ARXIV.1802.02611. URL: https://arxiv.org/abs/1802.02611 (cit. on p. 26).

[22] Pengcheng Xu, Zhongyuan Guo, Lei Liang, and Xiaohang Xu. «MSF-Net: Multi-Scale Feature Learning Network for Classification of Surface Defects of Multifarious Sizes». In: *Sensors* 21 (July 2021), p. 5125. DOI: 10.3390/s21155125 (cit. on p. 26).

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385 (cit. on p. 27).

[24] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. «MobileNetV2: Inverted Residuals and Linear Bottlenecks». In: (2018). DOI: 10.48550/ARXIV.1801.04381. URL: https://arxiv.org/abs/1801.04381 (cit. on pp. 28, 30).

[25] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661 (cit. on pp. 32, 33).

[26] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. «Domain-Adversarial Training of Neural Networks». In: (2015). DOI: 10.48550/ARXIV.1505.07818. URL: https://arxiv.org/abs/1505.07818 (cit. on pp. 33, 34).

[27] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. *Adversarial Discriminative Domain Adaptation*. 2017. DOI: 10.48550/ARXIV.1702.05464. URL: https://arxiv.org/abs/1702.05464 (cit. on pp. 34, 35).

[28] 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*. 2018. DOI: 10.48550/ARXIV.1802.10349. URL: https://arxiv.org/abs/1802.10349 (cit. on p. 35).

[29] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Pérez. *ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation*. 2018. DOI: 10.48550/ARXIV.1811.12833. URL: https://arxiv.org/abs/1811.12833 (cit. on pp. 36, 37, 71).

[30] Massih-Reza Amini, Vasilii Feofanov, Loic Pauletto, Emilie Devijver, and Yury Maximov. *Self-Training: A Survey*. 2022. DOI: 10.48550/ARXIV.2202.12040. URL: https://arxiv.org/abs/2202.12040 (cit. on p. 38).

[31] Pan Zhang, Bo Zhang, Ting Zhang, Dong Chen, Yong Wang, and Fang Wen. *Prototypical Pseudo Label Denoising and Target Structure Learning for Domain Adaptive Semantic Segmentation*. 2021. DOI: 10.48550/ARXIV.2101.10979. URL: https://arxiv.org/abs/2101.10979 (cit. on p. 39).

[32] Nikita Araslanov and Stefan Roth. *Self-supervised Augmentation Consistency for Adapting Semantic Segmentation*. 2021. DOI: 10.48550/ARXIV.2105.00097. URL: https://arxiv.org/abs/2105.00097 (cit. on p. 39).

[33] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. «FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence». In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 596–608. URL: https://proceedings.neurips.cc/paper/2020/file/06964dce9addb1c5cb5d6e3d9838f733-Paper.pdf (cit. on p. 39).

[34] Geoff French, Samuli Laine, Timo Aila, Michal Mackiewicz, and Graham Finlayson. *Semi-supervised semantic segmentation needs strong, varied perturbations*. 2019. DOI: 10.48550/ARXIV.1906.01916. URL: https://arxiv.org/abs/1906.01916 (cit. on p. 39).

[35] Viktor Olsson, Wilhelm Tranheden, Juliano Pinto, and Lennart Svensson. *ClassMix: Segmentation-Based Data Augmentation for Semi-Supervised Learning*. 2020. DOI: 10.48550/ARXIV.2007.07936. URL: https://arxiv.org/abs/2007.07936 (cit. on pp. 40, 46).

[36] Wilhelm Tranheden, Viktor Olsson, Juliano Pinto, and Lennart Svensson. *DACS: Domain Adaptation via Cross-domain Mixed Sampling*. 2020. DOI: 10.48550/ARXIV.2007.08702. URL: https://arxiv.org/abs/2007.08702 (cit. on pp. 40, 46, 78).

[37] Qianyu Zhou, Zhengyang Feng, Qiqi Gu, Jiangmiao Pang, Guangliang Cheng, Xuequan Lu, Jianping Shi, and Lizhuang Ma. *Context-Aware Mixup for Domain Adaptive Semantic Segmentation*. 2021. DOI: 10.48550/ARXIV.2108.03557. URL: https://arxiv.org/abs/2108.03557 (cit. on p. 40).

[38] Qing Lian, Fengmao Lv, Lixin Duan, and Boqing Gong. *Constructing Self-motivated Pyramid Curriculums for Cross-Domain Semantic Segmentation: A Non-Adversarial Approach*. 2019. DOI: 10.48550/ARXIV.1908.09547. URL: https://arxiv.org/abs/1908.09547 (cit. on p. 41).

[39] Ke Mei, Chuang Zhu, Jiaqi Zou, and Shanghang Zhang. *Instance Adaptive Self-Training for Unsupervised Domain Adaptation*. 2020. DOI: 10.48550/ARXIV.2008.12197. URL: https://arxiv.org/abs/2008.12197 (cit. on p. 41).

[40] Lukas Hoyer, Dengxin Dai, and Luc Van Gool. *DAFormer: Improving Network Architectures and Training Strategies for Domain-Adaptive Semantic Segmentation*. 2021. DOI: 10.48550/ARXIV.2111.14887. URL: https://arxiv.org/abs/2111.14887 (cit. on pp. 41, 48, 49, 78).

[41] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander Hauptmann. «Self-paced Curriculum Learning». In: Jan. 2015 (cit. on p. 42).

[42] Yang Zou, Zhiding Yu, B. V. K. Vijaya Kumar, and Jinsong Wang. *Domain Adaptation for Semantic Segmentation via Class-Balanced Self-Training*. 2018. DOI: 10.48550/ARXIV.1810.07911. URL: https://arxiv.org/abs/1810.07911 (cit. on p. 42).

[43] Yanchao Yang and Stefano Soatto. «FDA: Fourier Domain Adaptation for Semantic Segmentation». In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4084–4094. DOI: 10.1109/CVPR42600.2020.00414 (cit. on p. 43).

[44] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. *Intriguing properties of neural networks*. 2013. DOI: 10.48550/ARXIV.1312.6199. URL: https://arxiv.org/abs/1312.6199 (cit. on p. 48).

[45] Dan Hendrycks and Thomas Dietterich. *Benchmarking Neural Network Robustness to Common Corruptions and Perturbations*. 2019. DOI: 10.48550/ARXIV.1903.12261. URL: https://arxiv.org/abs/1903.12261 (cit. on p. 48).

[46] Dan Hendrycks et al. *The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization*. 2020. DOI: 10.48550/ARXIV.2006.16241. URL: https://arxiv.org/abs/2006.16241 (cit. on p. 48).

[47] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: 10.48550/ARXIV.2010.11929. URL: https://arxiv.org/abs/2010.11929 (cit. on p. 48).

[48] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. *Training data-efficient image transformers amp; distillation through attention*. 2020. DOI: 10.48550/ARXIV.2012.12877. URL: https://arxiv.org/abs/2012.12877 (cit. on p. 48).

[49] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*. 2021. DOI: 10.48550/ARXIV.2105.15203. URL: https://arxiv.org/abs/2105.15203 (cit. on pp. 48, 70).

[50] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. *Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions*. 2021. DOI: 10.48550/ARXIV.2102.12122. URL: https://arxiv.org/abs/2102.12122 (cit. on p. 48).

[51] 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: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3234–3243. DOI: 10.1109/CVPR.2016.352 (cit. on p. 51).