Università degli Studi di Padova

Department of Information Engineering

Master Thesis in Electronic Engineering

Road Scene Understanding using Depth Data from Stereo Vision

Supervisor
Pietro Zanuttigh
Università di Padova

Co-supervisor
Francesco Barbato
Università di Padova

Master Candidate
Leonardo Righetto

Academic Year 2021-2022
September 7th, 2022

Abstract

Multimodal Semantic Segmentation aims at obtaining an accurate pixel level understanding of the scene by jointly exploiting multiple information sources, e.g., standard images and depth maps or 3D data. This thesis tackles the problem via a multi-step process: first we search for the best candidate estimated depths (EDs) through stereo vision and feed them to a pre-trained neural network in order to compute segmentation maps. Then, a neural network is trained from scratch using EDs instead of ground truth depths together with improvements of the results by denoising or filtering the images.

The first step (stereo vision) is performed through well-established methods implemented usign the OpenCV library. The second step (semantic segmentation) is based on a deep learning framework of the encoder-decoder type. A common choice for encoder is ResNet101, while one for decoder is DeepLabv2, still other structures can be considered too, whenever high performance or fast computation is required. The overall work has been evaluated in a virtual environment (synthetic dataset) ensuring the availability of a huge number of images for training, validation, and test. The thesis then analyse Unsupervised Domain Adaptation, with the final purpose to solve the domain shift issue between synthetic data and real-world data.

In conclusion, the combination between visual and 3D information promises optimal results, as shown by the analysis performed during this thesis: the neural network reaches satisfying segmentation ability even in its simplest implementation (e.g., without Domain Adaptation techniques).

Sommario

La Segmentazione Semantica multimodale punta ad ottenere un'accurata comprensione della scena a livello pixel, usando l'unione di rappresentazioni multiple, ad esempio immagini standard con mappe di profondità o dati 3D. La tesi affronta questo problema partendo dalla ricerca delle migliori profondità stimate usando la stereo visione, fornendo queste profondità come input ad una rete neurale pre-allenata che ha il compito di produrre mappe di segmentazione. Dopodiché, una nuova rete viene allenata da zero usando le profondità stimate e vengono applicate tecniche di miglioramento delle immagini, come il filtraggio, con l'obiettivo di rimuovere il rumore.

La prima parte (stereo visione) è sviluppata utilizzando metodi ben consolidati, ottenuti da librerie open-source come OpenCV. La seconda parte (segmentazione semantica) è basata su una struttura di deep learning del tipo codificatore/decodificatore. Una possibile scelta per il codificatore è ResNet101, mentre una per il decodificatore è DeepLabv2, ma altre strutture vengono considerate in base all'obiettivo finale, quindi che si richieda un alto rendimento o dei calcoli veloci. Complessivamente il lavoro è svolto in un ambiente virtuale (dataset sintetici) così da avere facilmente accesso ad un grande numero di immagini per l'allenamento, la validazione e il test della rete. Inoltre, la tesi affronta il problema di risolvere lo spostamento di dominio che avviene tra i dati sintetici e quelli del mondo reale, studiando tecniche di adattamento di dominio non supervisionato.

In conclusione, l'utilizzo combinato di informazione visiva e 3D promette ottimi risultati, come mostrato dalle analisi effettuate in questa tesi: la rete neurale raggiunge capacità di segmentazione soddisfacenti anche nella sua versione più semplice (per esempio, senza utilizzare tecniche di adattamento di dominio).

Contents

Listing of figures

Listing of tables

1 Introduction

One of the main tasks in Computer Vision is scene understanding. The target of the thesis is to tackle this problem using both colour and 3D information. Through the years, many techniques have been used to extract visual information: Image Classification (IC), Object Detection, Semantic Segmentation (SS) and Instance Segmentation. While Image Classification gives just a single description for the whole image, the detail of information that can be obtained grows for each technique. Semantic Segmentation is the most challenging, where a label is assigned to each pixel, thus each single pixel of the image is assigned to a class.

The introduction of Deep Learning (DL) strategies revolutionized this field. Fundamental was the introduction of Deep Convolutional Neural Networks (DCNNs) architectures, first successfully applied to this task in the seminal paper by Long et al. [1]. Here the authors devised a way to adapt high-performance classification networks for Semantic Segmentation. These methods have been then perfectioned and extended, obtaining new highly performing models such as: DeepLab [2] and PSPNet [3]. A common problem to many DL approaches is called "domain shift": even if the model is trained on a very large dataset, it is not able to generalize well to distributions which differ, even slightly, from the training one. The issue is tackled with Domain Adaptation strategies; in particular in the Unsupervised Domain Adaptation (UDA) [4, 5] setting a source dataset contains labelled data, but the target one does not. In this thesis, we consider this setting and adapt a model, trained on synthetic data, to compute segmentation maps eventually on real-world images.

When working with DCNNs for SS the amount of required data is massive. Each image is labelled pixel-by-pixel. The production cost of new datasets would be enormous; for this reason, existing open-source datasets are available for different applications (e.g., Cityscapes [6], KITTI [7]). However, these datasets applied to specific tasks with different settings than the dataset ones would lead to poor results. To tackle this issue, it is possible to use a simulator capable of generating synthetic datasets in a wide range of different settings and conditions. One candidate dataset, for urban scenes as example, is SELMA [8]: a synthetic dataset obtained by a modified version of the open-source simulator CARLA [9] by the LTTM and SIGNET groups at the University of Padova.

Challenges have been proposed during the years in order to find the best ongoing strategies, e.g., the KITTI vision benchmark [10] in an Autonomous Driving context. Nevertheless, SS has still a lot to offer in terms of novel approaches. The combination between visual information and 3D or other sensor-based (e.g., thermal cameras [11]) information is named multimodal fusion [12] and promises optimal results in the near future.

The thesis is organized as follows: Chapter 2 is a summary on formulation and derivation of stereo vision model. This analysis requires overall a large mathematical background; here only fundamental concepts are presented, assuming true others which are less relevant for this specific project. Chapter 3 is a review of semantic segmentation. It includes the most popular deep learning architectures developed in the past years, a list of publicly available datasets thought for the segmentation task, and the analysis of data management techniques for improving performances of neural networks. Chapter 4 is a brief presentation of the domain adaptation problem, specifically in an unsupervised setting. Chapter 5 is a description of the methodology adopted while working on the thesis: code development, numerical strategies, study and adaptation of a neural network. The solution is divided mainly in two parts: estimation of disparity-maps and training of the network. Chapter 6 contains the obtained results: example of disparity-maps with difference parameter configurations, tables of scores for training and test of the network. Chapter 7 makes some considerations about the final results and possible future developments of the topics tackled in this project.

2 Stereo vision

Stereo vision is one of the most studied task in computer vision applications. The basic concept, namely stereopsis (from stereo = "solid" and opsis = "sight"), is the procedure that allows to compute three-dimensional information through binocular vision, that is, using two cameras, each one taking a picture of the same scene from a different perspective. The calculation of 3D data can be split in two sub-tasks: *matching* and *triangulation*.

Matching is about finding the points (or pixels), in the two images, which are projection of the same point in the scene; these two points are called matched. The hypothesis for the matching part is that the content of the two images is similar, but simple algorithms lead to the presence of many wrongly-matched points in the result. It is required to introduce other constraints to increase the robustness of the procedure. The most important of them is the epipolar constraint, which states that, taken one of two matched points in the first image, the other must lie on a line, called epipolar line, of the second image.

If the matches between the points of the two images are available, it is possible to build a representation of the scene, which takes name of depth-map, containing 3D information of the points that are projected on the two images. In order to properly compute a depth-map, it is required to know also the relative position and orientation of the two cameras (extrinsic parameters), as well as to have technical knowledge of each camera (intrinsic parameters). The calculation of intrinsic and extrinsic parameters is handled by a procedure called camera calibration (its analysis is beyond the purpose of this work), while the geometrical computation of the depth-map is the above mentioned *triangulation*.

2.1 Pinhole camera

In order to better understand the topic of this chapter, it is presented the simplest model of a camera and its basic concepts: the pinhole camera. We require the introduction of a reference frame, and a simple structure, to place the points of a scene in a 3D space and their projection on the image. The model includes two planes, the image plane $\mathcal{R}$ and the focal plane $\mathcal{F}$, and a point, the optical center $\mathbf{C}$, placed on the focal plane. The two planes are parallel and the distance between them is the focal distance f. The line orthogonal to the planes and passing through $\mathbf{C}$ is called optical axis, and its intersection with the image plane is the principal point. The model is represented in Figure 2.1.

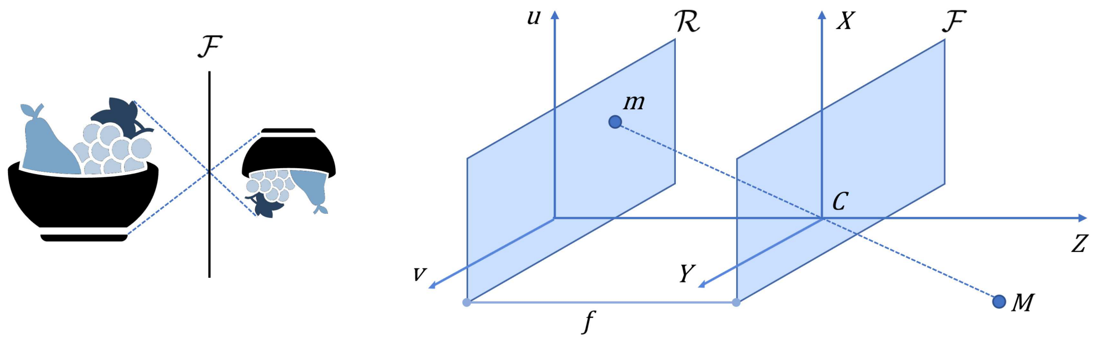

Figure 2.1: Pinhole camera model containing focal and image planes, the two reference systems, focal distance, optical center and an example of projection.

Two reference frames are created: (X, Y, Z) is the frame for the focal plane and is built placing the origin in the same position as the optical center, and pointing Z in the same direction of the optical axis; (u, v) is the frame for the image plane and is built placing the origin on the principal point, and pointing u and v in the same direction of X and Y, respectively.

A general point in the 3D space is called $\mathbf{M}$ and its coordinate are $\widetilde{\mathbf{M}} = [\mathbf{x}, \mathbf{y}, \mathbf{z}]^{\mathbf{T}}$; the projection of $\mathbf{M}$ on the image is $\mathbf{m}$ and its coordinate are $\widetilde{\mathbf{m}} = [\mathbf{u}, \mathbf{v}]^{\mathbf{T}}$. The projection $\mathbf{m}$ is the intersection between the image plane and the line that pass through $\mathbf{M}$ and $\mathbf{C}$. From Figure 2.2, we can derive the following equations:

$$\frac{f}{z} = \frac{-u}{x} = \frac{-v}{y}$$
$$\begin{cases} u = \dfrac{-f}{z}x \\ v = \dfrac{-f}{z}y \end{cases} \tag{2.1}$$

where the minus sign is used to keep track of the changing of sign of the coordinates.

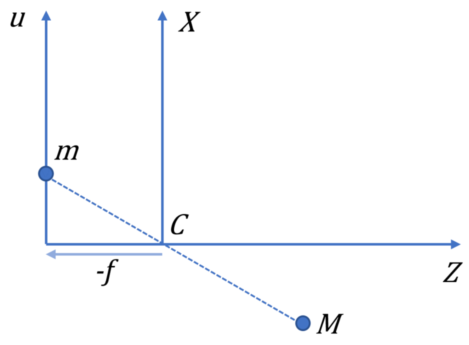

Figure 2.2: Simplified two-dimensional view of the pinhole camera model. Minus sign on f is used to keep track of the changing of sign of the coordinates.

The problem with the coordinates defined above is that the operation becomes non-linear because of the division by z. Instead, using homogeneous coordinates the operation is linear: $\mathbf{M} = [\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{1}]^{\mathbf{T}}$ and $\mathbf{m} = [\mathbf{u}, \mathbf{v}, \mathbf{1}]^{\mathbf{T}}$. We can write the final equation as:

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -fx \\ -fy \\ z \end{bmatrix} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.2}$$

where the z multiplied at the beginning can be seen as a scale factor. In fact, all the points M_1, M_2, and so on, that lie on the same line passing through the point $\mathbf{M}$ and $\mathbf{C}$, have the same projection on the image as point $\mathbf{M}$, i.e., point $\mathbf{m}$. The scale factor z is the distance of these points from the camera. The matrix which is pre-multiplied in the final term is the projection matrix P. In matrix form, the equation becomes:

$$\mathbf{m} \propto z\mathbf{m} = P\mathbf{M} \tag{2.3}$$

where the symbol $\propto$ means the left-handed term is a scaled version of the right-handed one.

Up to now, we considered an ideal case. In order to generalize this model, we must include the knowledge about the position of the camera (extrinsic parameters) and its internal non-idealities (intrinsic parameters).

2.1.1 Intrinsic parameters

It must be considered that the internal representation of the image is made of pixels, thus it is a discrete quantity. The origin of the frame (u, v) can be shifted w.r.t. its ideal position, and each axis can present a different scaling factor:

$$\begin{cases} u = k_u \dfrac{-f}{z} x + u_0 \\ v = k_v \dfrac{-f}{z} y + v_0 \end{cases} \tag{2.4}$$

where (u_0, v_0) are the real coordinates of the principal point, $k_{u,v} = \delta_{u,v}^{-1}$ and $\delta_{u,v}$ is the dimension of a pixel along the direction u, v. The new form for the projection matrix is:

$$P = \begin{bmatrix} -fk_u & 0 & u_0 & 0 \\ 0 & -fk_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.5}$$

It can also be possible that the two axes u and v are not perfectly orthogonal, but they present a generic angle θ between them. This way, the equation of the projection matrix is updated as follows:

$$P = \begin{bmatrix} -fk_u & fk_u \cot\theta & u_0 & 0 \\ 0 & -fk_v / \sin\theta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.6}$$

2.1.2 Extrinsic parameters

Usually, there is a fundamental reference frame, called world frame, which is different from the camera reference frame. It is necessary to define the affine transform that binds these two frames. What is required are a rotation matrix R and a translation vector $\mathbf{t}$. Afterward, we can call $\mathbf{M_c}$ the homogeneous coordinates of a point expressed in the camera frame, and $\mathbf{M}$ the same coordinates expressed in the world frame. The relation is:

$$\mathbf{M_c} = T_c \mathbf{M} \tag{2.7}$$

where T_c is the transformation matrix from world to camera frame, and its expression is:

$$T_c = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

From Equation 2.6 we can extract the sub-matrix K so that:

$$K = \begin{bmatrix} -fk_u & fk_u \cot\theta & u_0 \\ 0 & -fk_v/\sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \implies P = K\,[I|\mathbf{0}] \tag{2.9}$$

where I is the identity matrix. Then, from Equation 2.3 we derive:

$$\mathbf{m} \propto K\,[I|\mathbf{0}]\,\mathbf{M_c} = K\,[I|\mathbf{0}]\,T_c\mathbf{M} \implies P = K\,[I|\mathbf{0}]\,T_c \tag{2.10}$$

this is the most general form of the projection matrix P; T_c contains information about the extrinsic parameters, K about the intrinsic ones.

2.2 Triangulation

Triangulation is way simplified if the two cameras are aligned so to have coincident planes (focal and image planes). In this situation, the distance (in pixel units) between two matched points, which is called disparity, is purely horizontal. Using as reference frame a coordinate system placed on the left camera, Figure 2.3 and following equations hold true:

$$\begin{cases} \dfrac{f}{z} = \dfrac{-u}{x} \\ \dfrac{f}{z} = \dfrac{u'}{x-b} \end{cases} \tag{2.11}$$

from which can be derived:

$$z = \frac{f \cdot b}{u' - u} \tag{2.12}$$

From Equation 2.12 it is possible to extract the information about depth (z) by just knowing a simplified version of the camera system (f is the focal length, b is the baseline) and the disparity ($u' - u$). If b or f are unknown, it is possible to reconstruct a scaled version of the

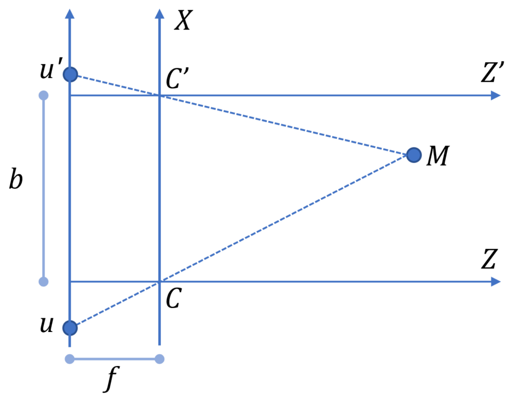

Figure 2.3: Triangulation geometry in the simplified case.

depth-map.

In general, the two coordinates (u, v) can be computed as follows, starting from equation 2.3:

$$\mathbf{m} = P\mathbf{M} \implies \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} \mathbf{M} \implies \begin{cases} u = \dfrac{\mathbf{p}_1^T\mathbf{M}}{\mathbf{p}_3^T\mathbf{M}} \\ v = \dfrac{\mathbf{p}_2^T\mathbf{M}}{\mathbf{p}_3^T\mathbf{M}} \end{cases} \tag{2.13}$$

where $\mathbf{p}_1^T$, $\mathbf{p}_2^T$ and $\mathbf{p}_3^T$ are the rows of matrix P. Note that the projection matrix has 11 degrees of freedom: 5 from intrinsic parameters, 3 from rotation matrix (camera orientation) and 3 from translation vector (camera position); moreover, remember it is defined up to a scale factor z. Developing Equation 2.10 it's possible to express the third dimension, namely depth, in the image space, which is the distance from the focal plane:

$$z = \begin{bmatrix} r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \mathbf{M} = \mathbf{r}_3^T \widetilde{\mathbf{M}} + t_3 \tag{2.14}$$

where $\mathbf{r}_3^T$ is the third row of rotation matrix R. Also in this case a second projection $\mathbf{m}'$ and its projection matrix P' are required to find a single solution.

2.3 Epipolar geometry

Epipolar geometry [13] studies how to define two matched points and their characteristics. Let's first introduce two elements that are required for the next analysis:

- **Optical center C**: it is defined as the origin of the (X, Y, Z) frame, or the origin of

(u, v) frame projection over the focal plane (see Figure 2.1). Algebraically, it is the kernel of P:

$$P\mathbf{C} = 0 \implies \left.\begin{matrix} \mathbf{p}_1^T\mathbf{C} = 0 \\ \mathbf{p}_2^T\mathbf{C} = 0 \\ \mathbf{p}_3^T\mathbf{C} = 0 \end{matrix}\right\} \tag{2.15}$$

Dividing the projection matrix P in its rotation R and translation $\mathbf{t}$ parts, and substituting in Equation 2.15, we obtain:

$$\begin{cases} P = [R|\mathbf{t}] \\ \mathbf{C} = \begin{bmatrix} \widetilde{\mathbf{C}} \\ 1 \end{bmatrix} \end{cases} \implies \mathbf{C} = R\widetilde{\mathbf{C}} + \mathbf{t} = 0 \implies \widetilde{\mathbf{C}} = -R^{-1}\mathbf{t} \tag{2.16}$$

- **Optical ray** $\mathbf{M}_\lambda$: given a point $\mathbf{m}$, it is defined as the locus of world points of which $\mathbf{m}$ is projection. Two relevant points belong to the optical ray: the optical center and the point at infinity, defined as:

$$\begin{bmatrix} R^{-1}\mathbf{m} \\ 0 \end{bmatrix} \implies P\begin{bmatrix} R^{-1}\mathbf{m} \\ 0 \end{bmatrix} = RR^{-1}\mathbf{m} = \mathbf{m} \tag{2.17}$$

  We can write its parametric equation as follows, where $\lambda \in \mathbb{R} \cup \{\infty\}$:

$$\mathbf{M}_\lambda = \mathbf{C} + \lambda \begin{bmatrix} R^{-1}\mathbf{m} \\ 0 \end{bmatrix} \tag{2.18}$$

Looking at Figure 2.4 we can define the following elements:

- **Epipolar line**: given a point $\mathbf{m}$, it is the line where its matched $\mathbf{m}'$ must lie on the second image.
- **Epipolar plane**: given a point $\mathbf{m}$ and its matched $\mathbf{m}'$, it is the plane where their two epipolar lines lie on.
- **Epipole** $\mathbf{e}$: it is the intersection point, on the image plane, of all the epipolar lines of one image.
- **Baseline**: it is the line passing through $\mathbf{C}$ and $\mathbf{C}'$, which defines a sheaf of planes containing all possible epipolar planes. When speaking of baseline length, it refers to the length of segment $\overline{\mathbf{C}\mathbf{C}'}$.

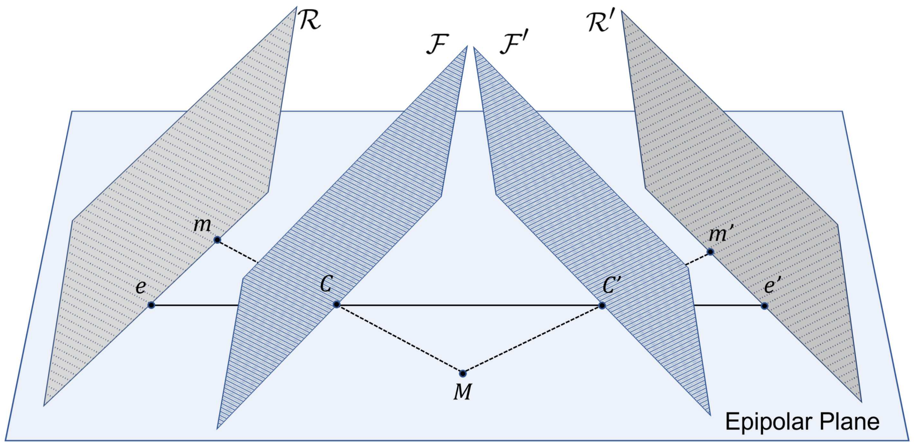

Figure 2.4: Representation of epipolar geometry between two non-parallel planes.

The epipolar line of $\mathbf{m}$ is the projection of $\mathbf{M}_\lambda$ over the second camera, thus projected through P'. Its equation is:

$$\mathbf{m}'_\lambda \simeq \lambda R' R^{-1}\mathbf{m} + \mathbf{e}' \qquad (2.19)$$

Managing last equation, we can extract the fundamental matrix F which contains all the required information for a correct evaluation of epipolar geometry:

$$F = E' R' R^{-1} \qquad (2.20)$$

where E' is an asymmetric matrix so that $E' \times \mathbf{e}' = \mathbf{0}$.

2.4 Matching

The core problem in stereo vision is finding the value of disparity of two matched points. Disparity can be seen as the difference (in vector space) of the two points when the images are superimposed (coincident reference frame). The output of disparity computation is called ”disparity-map” which corresponds to an image of same resolution as the input ones, but where each element contains the value of disparity of the associated pixel in the reference image (one between left and right is chosen). The search for disparity is further simplified if the two images are rectified and aligned, making the problem one-dimensional. This way, the distance between two matched points is just an horizontal shift in pixel units. Camera rectification is not addressed in this work, we will assume the two images to be already rectified. An example of disparity-map is shown in Figure 2.5.

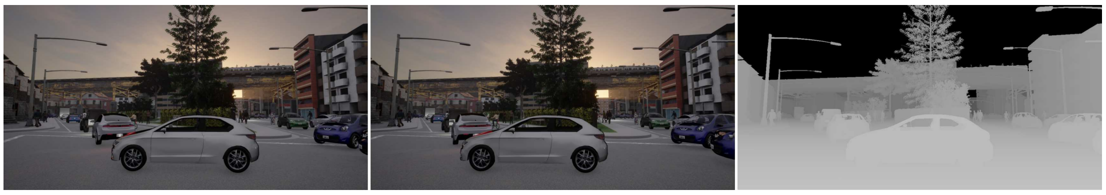

Figure 2.5: Left and right pictures of the scene (on the left and central). Ground truth representation of a disparity-map (on the right).

The matching operation of two images has to overcome a few complicated problems and to observe some constraints. In principle, in order to have similar appearances, the images have to contain similar content. Many strategies have been developed to address this task, which can be classified in local methods and global methods.

Problems in disparity calculation come from various phenomena, and one of them is the principle of similarity which is also the basis of the procedure: false matches – because the two images must be similar, one point could be matched with many other of the other image; occlusions – due to surfaces discontinuity or shape of objects, some regions of the scene appear only in one image, implying some points are automatically not matchable; radiometric distortion – due to physical characteristics of surfaces, radiance coming from them is seen differently by the two cameras; perspective distortion – due to perspective projection, objects have different shapes in the two images; baseline trade-off – the baseline in stereo vision can be seen as the distance between the two cameras. If the baseline is big, all the previous problems are worsen, but if it is small then the disparity is less significant (intuitively, if the baseline is zero it's like to use one single camera).

Some constraints can be exploited to partially solve these issues: similarity – one particular of the scene is similar in the two images; epipolar geometry – the matched of one point lies on the epipolar line of the second; smoothness – far from edges, a surface is homogeneous, limiting the disparity's gradient; uniqueness – a point of the left image can be matched with one and only one point of the right image, unless transparent surfaces appear in the scene; monotone order – if a point is placed on the right of another point in the first image, then the two points must be placed in the same position in the second image too. This is not true if the points are in the forbidden region of a point (Figure 2.6).

Studied methods to solve the matching task are divided in local and global. Both types try to apply the above constraints. The first ones work on a small number of pixels in a neighborhood of the central pixel, while the second work on the horizontal line that passes through the pixel of interest, named "scan-line", or on the whole image.

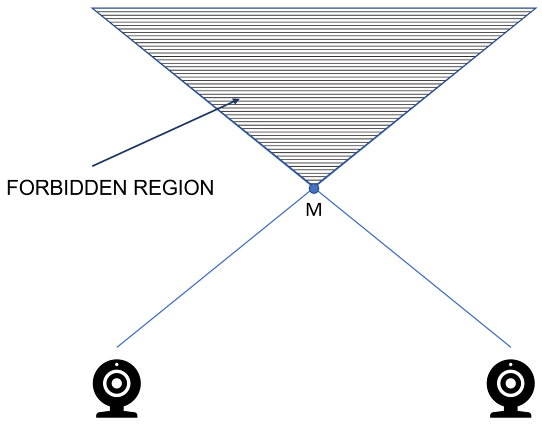

Figure 2.6: Forbidden region w.r.t. point M.

For what concerns local methods, the main strategies analyse a small region (window) in the first image, and compare it with regions of equal dimension in the other image, looking for the most similar one. The comparison is done using the gray level or a function of colour intensity. The procedure is applied to each single pixel of the first image, searching for its matched in the second one. The window on the second image is shifted along the epipolar line, which in the rectified case is just horizontal. The value of disparity is equal to the shift in pixel unit. In addition to the window-based methods, there are other types based on gradient, segmentation, and features.

Global methods instead try to bypass local problems such as uniformity of regions and occlusions. The optimization of these strategies implies a greater computational cost. Global methods exploit the use of Disparity Space Image (DSI), which is a 3D image $\mathcal{V}$, where $\mathcal{V}(u, v, d)$ is the value of the matching metric between $p = (u, v)$ and $p' = (u + d, v)$. The disparity-map is a surface inside the DSI, which is the best w.r.t. a cost function and the constraints presented in the previous section.

3
Semantic segmentation

Semantic segmentation is an ongoing task in recent computer vision applications, which consists in classifying an image on a pixel level, thus each pixel is assigned to a specific class (see Figure 3.1). It is a very dense and precise way to describe the scene contained in an image, and many research fields need a so well detailed description: autonomous driving, indoor navigation, visual servoing, virtual or augmented reality and many others. Semantic segmentation is a high-level task and one of the final steps to reach complete scene understanding [14]. The task was originally tackled using traditional computer vision and machine learning solutions, but the turning point was reached with the introduction of deep learning frameworks. Deep learning, in particular Convolutional Neural Networks (CNNs) [1], revolutionized the computer vision field, surpassing older methods and defining a new edge in terms of accuracy. Just to mention, stereo vision and depth estimation have been also studied through deep learning, but this is beyond the scope of this work. Despite deep learning has been studied for many years and outstanding results have been achieved, it has still a lot to offer and to be improved, especially when it comes to computation demand, real-time operations and the use of multimodal data. In order to understand which are the results obtained by deep learning in this field, in the following sections are presented the most relevant structures and the evaluation methods developed in the past years.

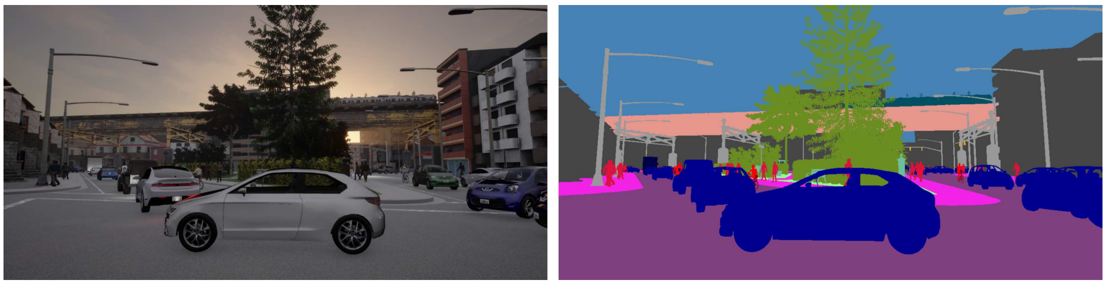

Figure 3.1: Image rgb from SELMA (on the left) and relative ground truth semantic segmentation-map (on the right), where each colour identifies a different class (e.g., pedestrians are red, cars are blue).

3.1 Evaluation metrics

One of the fundamental aspects of the research environment is the possibility to have a common reference, in order to make a reasonable comparison between new solutions and already existing ones. For this reason, during the years challenges have been proposed with the purpose of finding the best ongoing model. Challenges differ for task to be tackled and metric used for assigning a score. For image classification the most popular task has been ILSVRC (ImageNet Large Scale Visual Recognition Challenge) which refers to the ImageNet [15, 16, 17] dataset. For semantic segmentation, and in general for more recent architectures, there are several competitions related to different datasets, e.g., PASCAL VOC [18], MS-COCO [19], LVIS [20], KITTI [7, 10]. As said before, there must be a common way to compare various proposed solutions, so to choose the one that reaches the highest score as winner of the challenge. The three most common metrics for semantic segmentation are:

- **Precision**: it intuitively measures the precision of the model, and can be seen as a quality metric, but it leads to meaningless results when there is not enough data. Moreover, even if there is a sufficient amount of samples, if the dataset is class-unbalanced the model is likely to learn simple classes, leading to predictions containing only them. Its equation is $P = \frac{TP}{TP+FP}$.
- **Recall**: it is used to strengthen the meaning of the precision measure by giving a measure of how many true labels have been found, it can be seen as a quantity metric. Its equation is $R = \frac{TP}{TP+FN}$.
- **Intersection over Union**: it is the more relevant among the three metrics. Taking A as the ground truth region and B as the predicted region, the metric is evaluated as $IoU = \frac{(A \cap B)}{(A \cup B)} = \frac{(A \cap B)}{|A|+|B|-(A \cap B)}$. Ideally, $A = B$ which means the two regions are superimposed.

Each metric is applied separately over each class. In order to have a single score summarizing all the classes, the average is taken obtaining, for example, the mIoU (mean Intersection over Union). Using the same letters as in Section 3.3:

$$mIoU = \frac{\sum_i^M IoU_{c_i}}{|\mathcal{C}|} \tag{3.1}$$

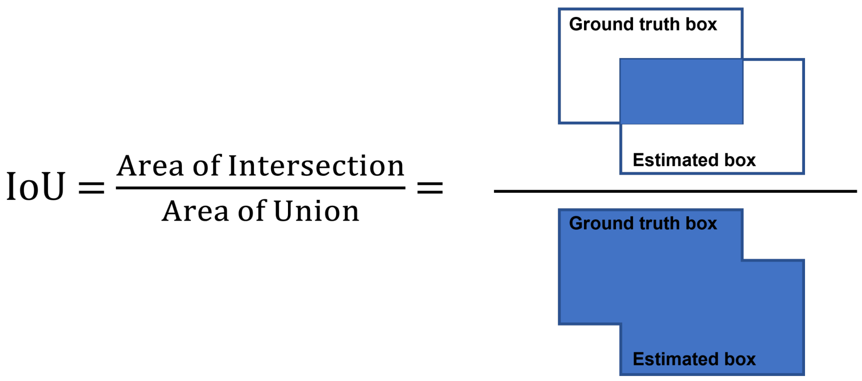

Figure 3.2: Graphical description of IoU. The more the two boxes are superimposed, the better the score is (visually, intersection is more or less 25% of union, which is not good).

3.2 Datasets

In this section are presented some of the most common datasets (some of them already mentioned above) with a brief description for each one. In the following list mainly datasets containing 2D images are described, although many others exist for 2.5D and 3D data representation [14].

3.2.1 PASCAL VOC (Visual Object Classes)

The dataset and the associated challenge address five possible tasks: classification, detection, segmentation, action classification and person layout. The basic version of the dataset allows the description of 21 classes: aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, TV/monitor, bird, cat, cow, dog, horse, sheep, person and background. The dataset has almost 3000 available images (1464 for training, 1449 for validation), plus a test set kept private for the challenge. As understandable from the list of classes, this dataset is thought for general purpose, it's not good when the application is very specific. However, this is downright the most popular dataset for semantic segmentation, so every remarkable architecture has been tested on it.

The basic version of PASCAL VOC has then been extended in the PASCAL Context [21] dataset. This second version contains the original 21 classes, plus another 519, for a total of 540 classes, which are divided in three macro-categories: objects, stuff, and hybrids. Actually, only 59 of the classes are really relevant for almost every cases.

Another version developed starting from the basic dataset is the PASCAL Part [22]. This time, only the original classes are kept, but they are divided in their sub-parts, creating a lot of new sub-classes. For example, bicycle is divided in back wheel, chain wheel, front wheel, handlebar, headlight, and saddle.

3.2.2 MS-COCO (Microsoft Common Objects in Context)

The dataset and the associated challenge address different tasks, focused on detection and segmentation, for which it includes 80 classes, 82783 samples for training, 40504 for validation and more than 80000 for test. An interesting feature offered by this dataset is that the test set is split in four sub-sets, used separately for additional validation, standard test, test performed during the challenge and an additional set called “reserve”. The last one is used to further inspect some particular cases. The dataset gained reputation and importance thanks to its large scale, starting to be annually used in official competitions together with ImageNet.

3.2.3 KITTI

The dataset, and its KITTI vision benchmark suite, is very popular in autonomous driving applications. It is made capturing real traffic scenarios using various methods, such as RGB images at high resolution, grayscale stereo cameras and 3D laser scanners. The problem with KITTI is that it does not contain ground truth samples for semantic segmentation. For this reason, researchers have manually labelled some data necessary for their situation [23, 24, 25, 26].

The following datasets better fit the topic of this thesis, being they thought for urban scene understanding and including synthetically generated datasets. In particular, SELMA is the one available for this project.

3.2.4 SYNTHIA (SYNTHetic collection of Imagery and Annotations)

SYNTHIA [27] is a synthetic dataset, generated in a virtual environment, and contains a set of photorealistic renderings (13407 samples for training), specifically labelled for semantic segmentation, in a driving and urban context. It includes 11 classes (void, sky, building, road, sidewalk, fence, vegetation, pole, car, sign, pedestrian, and cyclist) and allows diversity of scenes (towns, cities, highways, dynamic objects, seasons, and weather).

3.2.5 Cityscapes

Similarly to SYNTHIA, Cityscapes [6] aims to urban scene understanding through semantic segmentation, but it's not a synthetic dataset. In fact, it contains almost 5000 well annotated samples and 20000 coarse annotated ones. It was created capturing images in 50 different cities in different periods, weather, and daytime.

3.2.6 SELMA

SELMA [8] is the dataset used for developing this whole project. It is a synthetic dataset obtained by a modified version of the open-source simulator CARLA [9]. The authors explain how many synthetic datasets for road scene understanding lack of multi-sensor data and variety of scenes (different daytime, weather, etc.), and how they tried to fix this issue. SELMA potentially includes more than 20M samples, acquired combining more than 30000 waypoints, 24 different sensors, e.g., RGB cameras, depth cameras, semantic cameras and LiDAR sensors [28], and 27 different weather and daytime conditions.

3.3 Popular deep learning architectures

The evolution of image understanding is identifiable in 3 macro-steps. The original task was image classification, where it was sufficient to describe the whole image depending on which objects were contained in it. The next step was object localization, drawing some geometrical regions (bounding boxes) where the classified objects are placed in the image. The final step would be semantic segmentation (and, possibly, instance segmentation as its extension), a comprehensive knowledge of the whole image, where each pixel belongs to a specific class defined a priori. In this case, the addressed problem is semantic segmentation, which can be formulated in the following way: $\mathcal{C} = \{c_1, ..., c_M\}$ is the set of classes, $\mathcal{P} = \{p_1, ..., p_N\}$ is the set of pixels. The classes are extended to M+1 because the "void" class is added, to describe the background/uncertain pixels. This additional class is not interested while training a model, thus, after a prediction is compared with its ground truth reference, no back-propagation is operated on pixels belonging to the void class. $\mathcal{P}$ is a 2D image of resolution $W \times H = N$, but in general it can have any dimension (such as 3D data).

3.3.1 AlexNet

AlexNet [15] was the first neural network that won the ILSVRC in 2012 with a test accuracy of 84.6%, beating its strongest competitor, which used traditional methods, achieving an accuracy of 73.8%. Compared to most recent network, the architecture of AlexNet was relatively simple. It has five convolutional, max-pooling, ReLU, three fully-connected and dropout layers.

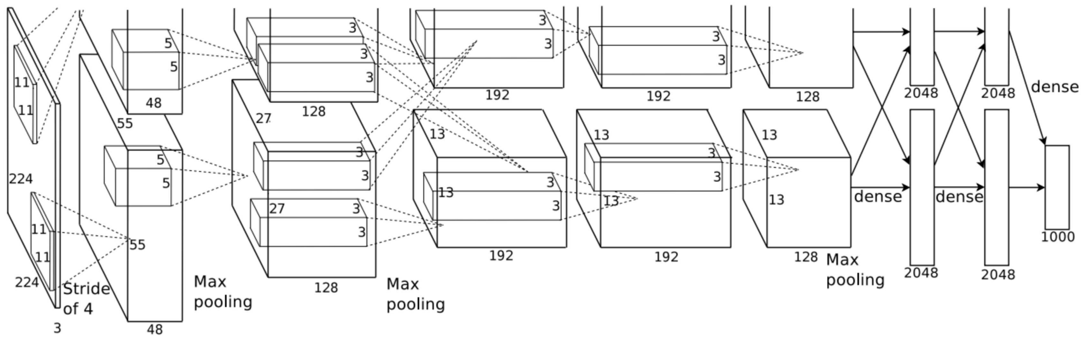

Figure 3.3: AlexNet architecture (image taken from [15]).

3.3.2 VGG

VGG (Visual Geometry Group) [29] is a CNN that won the ILSVRC in 2013 with an accuracy of 92.7%. The model used in the challenge is specifically the VGG-16 which uses 16 weight layers. The main difference introduced by VGG is the use of many convolution layers with small receptive field. This way, the amount of model's parameters is reduced while more non-linearities are introduced, obtaining a more discriminative decision function and reducing the effort required for training.

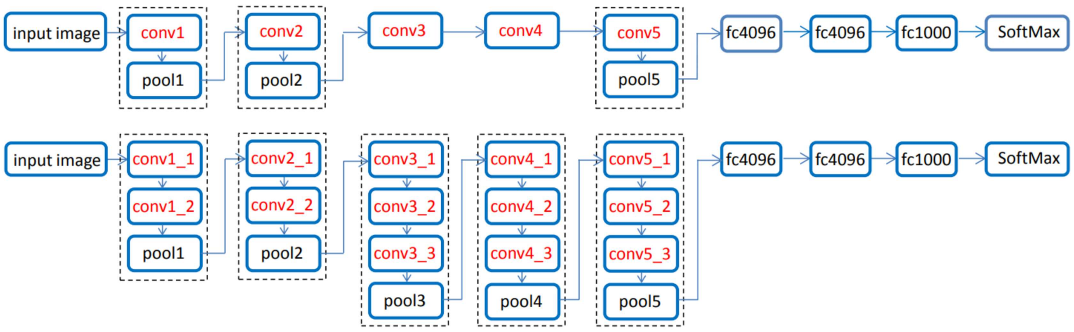

Figure 3.4: Comparison between AlexNet (top) and VGG (bottom) architectures (image taken from [30]).

3.3.3 GoogLeNet

GoogLeNet [31] won the ILSVRC in 2014 with an accuracy of 93.3%. With respect to its predecessors, the complexity of the model is increased, including 22 layers and the introduction of a new building block named "inception" module. The standard way to build a network is to put one layer after another, in a sequence. This new block changed this idea, creating a NiN (Network in Network), where pooling and convolutional layers are computed in parallel and are followed by a 1x1 convolution operation. This way, model's parameters are reduced, thus decreasing memory usage and computational cost.

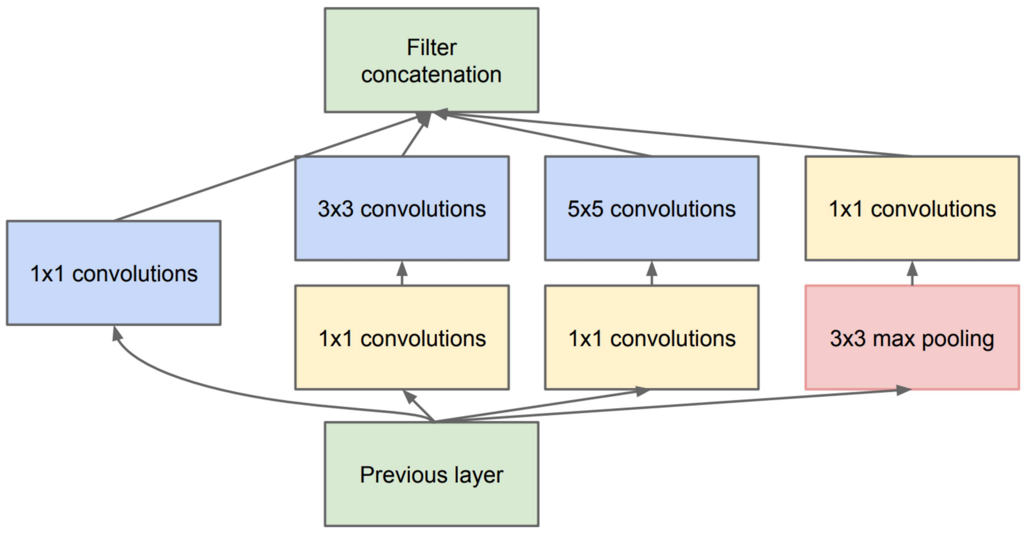

Figure 3.5: GoogLeNet fundamental block called: Inception module (image taken from [31]).

3.3.4 ResNet

ResNet [32] is a CNN developed by Microsoft which won the ILSVRC in 2016 with an accuracy of 96.4%. There are various versions of the network, which differ for the overall number of layers (e.g., 18, 34, 50, 101, 152). In addition, it introduces a new building block named "residual". The residual blocks help in training very deep network by coping the input of a layer to the next one without modifying the data. This operation is called "identity skip connection". This strategy ensures that the next layer obtains more and different knowledge, but also it helps overcoming the vanishing gradient problem.

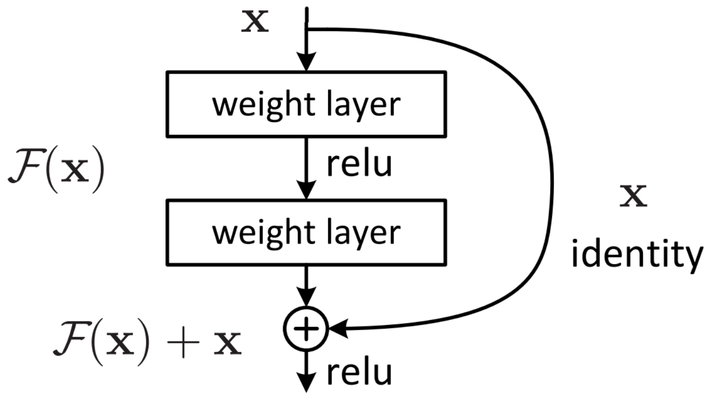

Figure 3.6: ResNet fundamental block called: Residual learning (image taken from [32]).

3.3.5 DeepLab

DeepLab [2] is a CNN developed aiming to obtain high performances in the semantic segmentation and dense prediction tasks. It includes three major features:

- **Atrous convolution** [33], which is used to control the resolution of features inside the network and to increase the field of view filters, enabling a larger comprehension of the context without increasing the number of parameters, thus keeping the same computation effort.

- **Atrous spatial pyramid pooling (ASPP)**, that allows efficient segmentation of objects at multiple scales. The ASPP action concerns giving convoluted features as input to many filters at different sampling rates and field of views, achieving good knowledge of both objects and the whole image at multiple scales.

- **Conditional Random Field (CRF)** [34], which is an already existing structure exploited by the first versions of DeepLab but removed in the latters. It is a fully connected layer placed after the final layer of the DCNN and is used to improve the localization of objects. The CRF is then used in combination with max-pooling and down-sampling properties of classical DCNNs, which reach good invariance but have problems in localization.

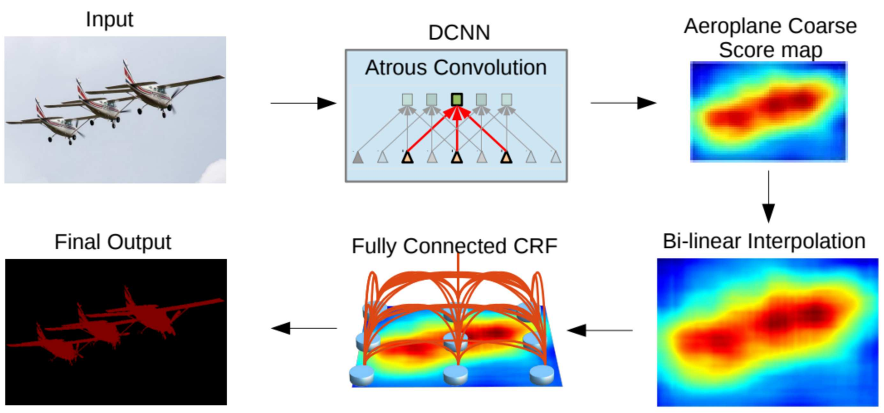

Figure 3.7: DeepLab model procedure. First, a DCNN is implemented in a fully convolutional way. Then, using atrous convolution is possible to reduce the required downsampling (from 32x down 8x). A bilinear interpolation stage enlarges the feature maps to the original image resolution. A fully connected CRF is then applied to refine the segmentation result and better capture the object boundaries (image taken from [2]).

3.4 Transfer learning and data augmentation

Usually, the training of a deep neural network from scratch is a hard task because of the huge amount of data which is required, especially for semantic segmentation where each pixel must be labelled (intuitively, millions of pixels for thousands of images). Furthermore, the convergence of the model can take a very long time, and because of pixel-wise prediction, which implies contrasting gradients in neighboring pixels, convergence can never happen. In general, it is a good procedure to start the new training with already trained weights, instead of randomized ones. This scenario takes name of transfer learning. It has been demonstrated that transferring features between two different tasks can be better than starting from random initialization, but it must be considered that the transferability is more efficient when the two tasks are similar each other [35]. Transfer learning is not straightforwardly applicable because there are constraints imposed by the network's architecture. On the other hand, it is very common to use already existing architectures instead of creating a new one, thus easing the use of transfer learning. Moreover, the training of a network is different when it is computed from a pre-trained network: only some specific layers have to be fine-tuned and a new policy for the learning rate has to be chosen, usually making it smaller considering that the pre-trained weights are expected to be relatively good. Due to the overwhelming amount of labelled data used by semantic segmentation, only small datasets are available, and the problem get worse when the data becomes three-dimensional. For this reason, transfer learning and the use of pre-trained network has become very common in semantic neural networks.

Data augmentation is a strategy used to improve machine learning models, especially deep learning frameworks. It can increase the model's efficiency, decreasing the convergence time, improving robustness against overfitting and making it more capable of generalization[36]. It consists in performing some transformation, usually in the data space, sometimes in the feature space too. The target is to generate new data from the already existing ones, applying transformations such as: translation, rotation, scaling, warping, crops, shifts in the colour space. The increase in the number of samples, in particular for training, is useful to avoid some issues like overfitting. Data augmentation is helpful specifically for small datasets, like the ones for semantic segmentation. As shown in [37], the use of augmentation allowed to increase the dataset's dimension from 1500 to 19000 samples, obtaining a IoU score of 94.2%, instead of the 73.09% obtained with the non-augmented dataset.

4 Unsupervised Domain Adaptation

Domain Adaptation (DA) is a group of strategies studied to handle the so called "domain shift issue". Usually, machine learning algorithms are built with the idea that training and test datasets are homogeneous, thus their data are similar. In practical cases, especially in computer vision applications where pictures of various environments can be involved, this is not true. The way the training dataset is created and its content can differ from the test one, however keeping some similarities. The dissimilarity is managed at the domain level, where data are defined, calling source domain the one for training samples, and target domain the one over which good performances are expected. The role of domain adaptation is trying to fill the gap between the two domains, helping a model, trained on source data, to work in an environment it has not entirely learned. Many research areas, other than computer vision, widely started to apply this strategy [38, 39].

Domain adaptation is linked to transfer learning (explained in Section 3.4), which uses labelled data from a source domain to tackle new tasks in the target domain. The greater the domain shift is, the more the performance of a model are degraded, thus domain adaptation is an easier task when the two domains are more correlated. For this reason, the ability of discovering suitable source data, to extract useful clues from, is of fundamental importance.

A particular case is when no labelling information is available from the target domain. In this situation, one idea could be to train the model only on the source domain (which is labelled), hoping to get a functional machine also in different domains. This leads in general to poor results, even if the domain shift is not so visually noticeable. A better idea is to use all the labelled data from source domain and combine them with unlabelled data from target domain. This takes name of Unsupervised Domain Adaptation (UDA) [4] and is almost mandatory in applications like semantic segmentation where the acquisition of new data is feasible (especially when using synthetic datasets), while the annotation of samples for making new ground truth data is a much more demanding task.

4.1 Adaptation spaces

As mentioned in the introduction of this chapter, when the distribution of source domain (where a model is trained) and target domain (where a model is tested/used) is not the same, finding a way to eliminate the presence of domain shifts is of essential importance to decrease the performance drop of the model. In a network (focusing the analysis on semantic neural networks), the adaptation step can be performed at different levels:

- **Adaptation at the input level** $\rightarrow$ the first possible strategy is to manage input samples (images) so to achieve uniformity with target samples. Although the two domains can share high-level features, differences at lower levels, even if unrelated to semantic, imply a degradation of the model prediction efficiency. In common cases, a function has to be found that can map input images to a new domain, or space, preserving the semantic content, where they gain similarities w.r.t. target images; the model's training is made on this new set of mapped images. One strength of this strategy is that it is independent from the task (training is performed after the mapping step). On the other hand, in its simplest version, it is not very discriminative; it needs some further regularization scheme. In practice, it could be possible to realize a mapping function able to add many domain invariances to images, all lacking semantic meaning, eventually making the network still unable to perform its task. During the years, many techniques have been studied to make input image adaptation consistent also in a semantic sense, for example exploiting image reconstruction constraints, uniformity of segmentation regions or ad-hoc manipulation of low-level statistics.

- **Adaptation at the feature level** $\rightarrow$ another idea is to act inside the network, when the training process is already started, forcing the feature extractor (encoder) to adjust the distribution of source and target domains in the latent space. This way, the network classifier (decoder) should learn how to segment both source and target domain images, exploiting the similarities defined at the latent level. Notice that supervision is applied only from source data. This technique turned out optimal for the image classification task but presents problems with segmentation. In this second case, the feature space reaches much greater complexity and dimensionality due to the presence of global and local (class-wise) visual cues.

- **Adaptation at the output level** → adaptation can be performed on the network's output semantic map (or on the layer before the output one). This way, a lot of semantic cues are kept, and the dimensionality of output space is smaller w.r.t. the one of latent space simplifying the adaptation step. In addition, the operation is done on semantic maps, thus allowing to infer labelled data to unlabelled ones of the target domain, as in a weak supervised environment.

- **Adaptation at specific network levels** → in general, adaptation can be performed at any level of the network, trying to align the relative space. The designated level depends on the structure, and the reason in doing so is to achieve better high-level patterns for segmentation and to reasonably match the source and target domains.

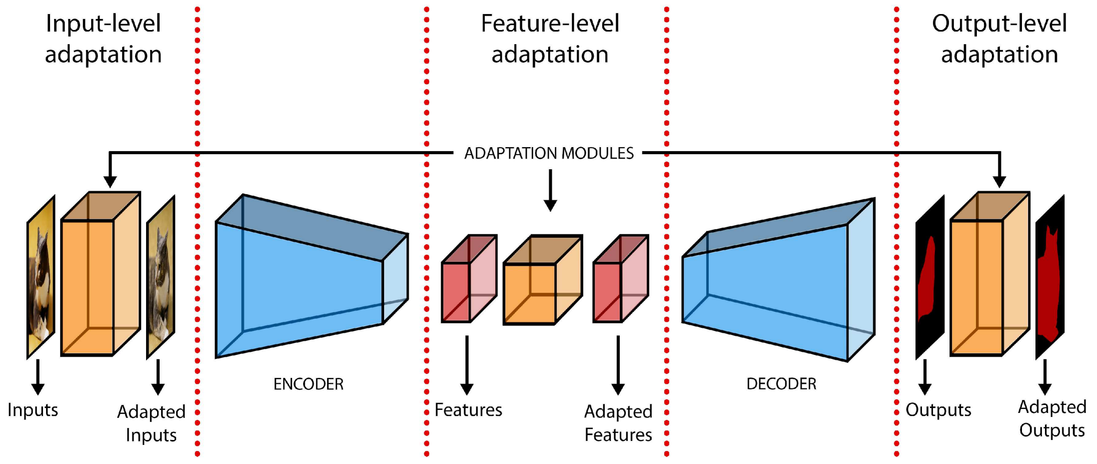

Figure 4.1: Schematic representation of the autoencoder deep learning architecture, with highlighted spots for adaptation spaces (image take from [4]).

The scheme in Figure 4.1 shows various levels at which adaptation can be performed. In addition, it is also a representation of the *autoencoder* architecture, composed by the two main blocks encoder and decoder (also called backbone and classifier). The encoder has the role of extracting features from input data, while the decoder makes predictions and resizes features to their original size. In practical cases, pre-trained encoder are used to fasten the training of new deep learning models, generally improving the obtained results. This structure helped a lot in tasks like semantic segmentation, where a common encoder can be used while changing only the decoder depending on the specific requirements.

4.2 Mathematical formulation

Computer vision tasks such as classification and segmentation can be seen as the problem of finding the best function h (the one that minimizes a specific cost function) among a set of functions $\mathcal{H}$ (hypothesis class, defined using prior knowledge of the problem), so that $h : \mathcal{X} \to \mathcal{Y}$ is defined from an input space $\mathcal{X}$ (e.g., images) to an output space $\mathcal{Y}$ (e.g., semantic maps). Mathematically, it is assumed that all real-world couples of data $(x, y) \in \mathcal{X} \times \mathcal{Y}$ have a distribution $\mathcal{D}$ drawn according to an unknown probability.

In a Domain Adaptation environment, there are two different but related distributions over the space $\mathcal{X} \times \mathcal{Y}$, one for source domain $\mathcal{D}_{\mathcal{S}}$ and one for target domain $\mathcal{D}_{\mathcal{T}}$, and two training sets $\mathcal{S}$ and $\mathcal{T}$, sampled over $\mathcal{X}$, relatively for the same two domains. The purpose of Domain Adaptation is to use labelled samples from $\mathcal{S}$ and labelled, unlabelled or both from $\mathcal{T}$ in order to find the best h that works on $\mathcal{T}$. If only unlabelled data are used from $\mathcal{T}$ the adaptation is said unsupervised (UDA).

Afterward, in Domain Adaptation three sub-categories can be found for source ($\mathcal{C}_{\mathcal{S}}$), target ($\mathcal{C}_{\mathcal{T}}$) and learning process ($\mathcal{C}_{\mathcal{L}}$), thus defining few sub-types of adaptation, represented in Figure 4.2:

- **Closed Set DA**: $\mathcal{C}_{\mathcal{S}} = \mathcal{C}_{\mathcal{T}} \implies$ all categories are in both source and target domains.
- **Partial DA**: $\mathcal{C}_{\mathcal{S}} \supset \mathcal{C}_{\mathcal{T}} \implies$ all categories in source domain, only few categories in target domain.
- **Open Set DA**: $\mathcal{C}_{\mathcal{S}} \subset \mathcal{C}_{\mathcal{T}} \implies$ all categories in target domain, only few categories in source domain.
- **Open-Partial DA**: $\mathcal{C}_{\mathcal{S}} \neq \mathcal{C}_{\mathcal{T}} \land \mathcal{C}_{\mathcal{S}} \cap \mathcal{C}_{\mathcal{T}} \neq \varnothing \implies$ some categories in source domain, some in target domain and some in both domains.
- **Boundless DA**: $\mathcal{C}_{\mathcal{S}} \subset \mathcal{C}_{\mathcal{T}} \land \mathcal{C}_{\mathcal{L}} = \mathcal{C}_{\mathcal{S}} \cup \mathcal{C}_{\mathcal{T}} \implies$ equal to Open Set DA but all categories in target domain are individually learned.

4.3 UDA in road scene scenarios

UDA gains relevant importance in some particular cases. Its primary feature is to transfer knowledge between datasets representing different contexts. This comes particularly in hand when working with a source domain obtained by synthetic data, thus made of computer generated images. Having such dataset allows a drastic reduction of costs when dealing with

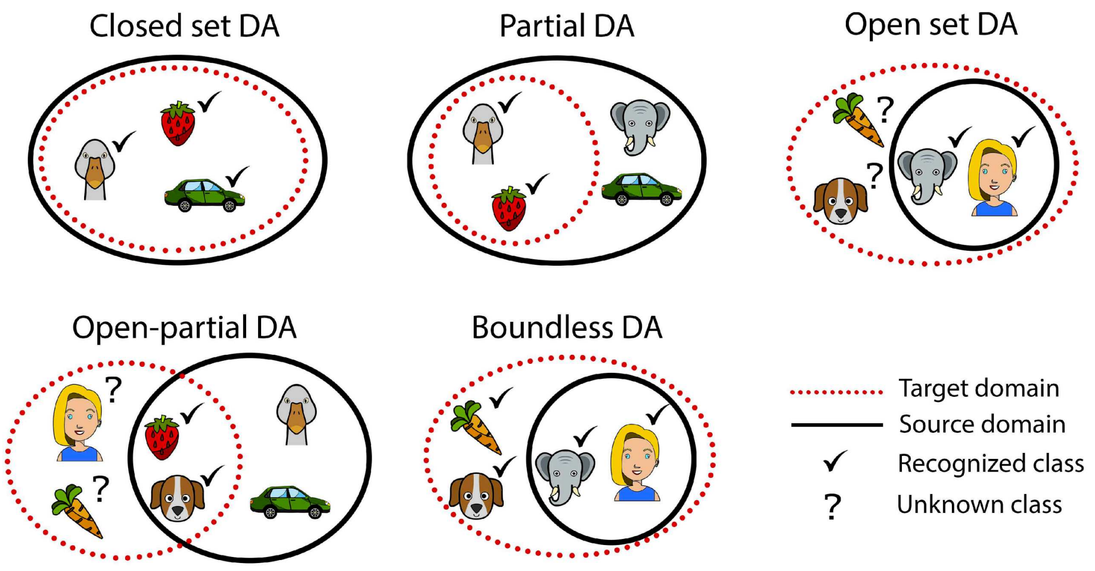

Figure 4.2: Five categories for Domain Adaptation. In Open Set DA, classes that belong only to the target domain are labelled as unknown by the model, instead in Boundless DA they are learned separately (image taken from [4]).

generation of labelled data, because if before it was an operation done manually by human operators, now all required information is automatically accessible through software. The role of UDA is to adapt this synthetic and labelled data to real-world images.

Having said that, a big effort was put to apply this strategy to urban and road scenes (images containing vehicles, buildings, objects inside cities), and this was mainly due to the increasing interest in Autonomous Driving research area. Works on Autonomous Driving began to be the reference scenario regarding UDA for semantic segmentation, also because a fully autonomous vehicle must have a precise and reliable understanding of the surrounding environment. This later extended to other scenarios, like autonomous robotics (visual servoing).

Moreover, there are plenty of datasets (both synthetic and real-world) that have been made publicly available for this task. Some of them are cited in Section 3.2, but many other exist:

- **GTA5** [40] $\rightarrow$ it is a synthetic dataset for urban driving scenario, and one of the first of its genre. It is made of almost 25000 labelled images, from car perspective, for semantic segmentation. Renderings have been computed using the video game Grand Theft Auto V (GTA5) and have outstanding visual quality thanks to the high budget invested in the video game engine (created by a popular brand). There are 19 object classes, which after some label-managements are compatible with real-world dataset such as Cityscapes.

- **Mapillary** [41] $\rightarrow$ it is a real-world dataset and one of the most complete for this task. It contains 25000 high resolution images taken around the globe from different devices, thus achieving extreme variability in its content (many classes, different appearances, diversity of settings and external environments). It includes 152 object classes, which are often adapted to the ones of Cityscapes.

- **Oxford RobotCar Dataset** [42] $\rightarrow$ it is a real-world dataset and an optimal one from the scene diversity point. It has been made traversing the same route (10 km) for almost a year, generating 20M samples which are heterogeneous speaking of daytime, weather, light condition, and sensor acquisition (6 cameras plus LiDAR, GPS and INS sensors).

- **Google Street View** $\rightarrow$ Google Street View has been used to generate a dataset in the paper by Chen et al.[43]. Four cities have been chosen trying to reach maximum visual variations: Rome, Tokyo, Rio, Taipei. Location captured in the cities have been randomly chosen.

5 Proposed solution

The first task of this thesis project consists of the generation of estimated depth-maps. Shortly summarizing Chapter 2, the depth-map of an image is another image of same resolution as the first one, where each pixel contains information about the distance (depth) between the camera and the object seen in the pixel position. In order to obtain a depth-map, many ways have been proposed, which differ in hardware complexity, production cost, performance of the results. In this work, we analyse the easiest to implement, economic advantageous, though low performance solution: stereo vision. This technique requires a pair of cameras (left and right), which take a picture of the same scene from two different perspectives. With the purpose of simplify further calculation, the two images must be aligned and placed in the same 2D plane, and a reference frame has to be defined, e.g., with the origin on the left camera. This way, it is possible to compute a first map, where each element contains the disparity between a left pixel and its corresponding right pixels, thus the value of distance (in pixel unit) between two pixels that represent the same object in the scene. If the images are correctly aligned, the disparity is the difference between the position of the two pixels on the horizontal axis, i.e., it is the horizontal shift of one pixel to the other. Afterward, the depth-map can be computed by taking the inverse of the disparity map as follows:

$$depth = \frac{f \cdot b}{disparity} \tag{5.1}$$

where f is the focal distance (distance between focal plane and image plane) and b is the baseline (distance between the two cameras). The poor results offered by this strategy come from the fact that, due to many artifact phenomena, such as obstacles or surface reflectivity, it is not possible to evaluate the disparity for each pixel, resulting in high noise level in the map.

5.1 Depth-map generation

In order to get the best results, the code has been implemented in Python 3 [44] using OpenCV [45], a well-known library for image processing. The work was carried out on the SELMA dataset. Disparities have been computed using a method based on the Semi-Global Matching (SGM) algorithm [46], which implements pixel-wise cost calculation, smoothness constraint, sub-pixel accuracy for disparity computation and occlusion detection. The method is named *stereoSGBM()* and it needs two main parameters: *numDisparities* and *blockSize*. Many other secondary parameters are available, and they have been used in later versions of the code. The first parameter indicates the maximum horizontal shift explored by the algorithm to find the two matching pixels. The greater it is, the smaller the measurable depth is, but the overall noise increases, too. The second parameter indicates the size of the square window used by the algorithm to compare the left pixel (and its neighbourhood) with the candidate right pixels (and their neighbourhood). A small window gives precise though noisy measurements, while a large window gives smoother but less precise measurements. The increase of one or both parameters also imply an increase of the computational time.

Before addressing the effective disparity calculation, a small application was developed to have a better visual understanding. The code allows to compute a disparity-map and open it on a window, together with sliders to manually change the parameters. This has been an initial analysis useful to gain familiarity with the concept of disparity and to visualize the effects of different parameter configurations. Also, it was helpful to make some considerations about the variable type (in Python) of the matrix obtained as output from the disparity computation.

The first approach to find valid estimated depth maps (ED maps) was to iteratively compute the disparity map for many configurations of interest, saving some relevant information in order to make a comparison among them. Chosen configurations were the possible combinations between the two parameters available in the *stereoSGBM()* method. With a certain range, only reasonable values have been checked. For *numDisparities*, after some initial tries, a lower bound was found necessary to keep track of object very near the cameras (this will be explained later), and an upper bound was arbitrarily chosen, knowing that the larger this parameter is, the higher is the noise level. Furthermore, using this method, it can be only a multiple of 16. For *blockSize*, which must be an odd number, the bounds were decided considering that a high value can eliminate some noise, and a low value maintains the shape of objects, thus it ensures precision. The most used parameters are summarized in Table 5.1.

numDisparities	96, 112, 128, 144, 160, 176, 192
blockSize	7, 9, 11, 13, 15, 17, 19

Table 5.1: Most analysed parameters during the iterative search.

While working on this, most time was spent by the algorithm to run, because images were initially processed on a CPU. In this condition, computation of thousands of disparity-maps can take many hours, and the situation is worsened when multiple computations for the same pair of images have to be done, for example, in order to compare different configurations of the estimation method (different values of *numDisparities* and *blockSize*). For this reason, the research was always done on a subset of the training set of SELMA (not of the test set in order to respect learning paradigms), for example, taking 100, 500 or even 5000 random images from it. This way, optimization was not performed on the whole dataset, or even the whole training set, but it was sufficient to include a large number of samples so to have an average idea of the result.

Initially, a random subset of the train dataset was created. For each image, it was taken the difference between its ground truth (GT) map and various EDs (which translates into a difference between matrices). Afterward, their mean absolute error (MAE) and mean square error (MSE) have been calculated:

$$\begin{aligned} MAE &= mean(|GT - ED|) \\ MSE &= mean((GT - ED)^2) \end{aligned} \tag{5.2}$$

where GT and ED are matrices containing respectively ground truth depth information and estimated depth information. The $mean(\cdot)$ operator computes average among all elements of one matrix, while the absolute ($|\cdot|$) and square ($(\cdot)^2$) operators act pixel-by-pixel. Ideally, the ED map which is the most similar to the GT one would be the best candidate, which means the one that gives the smaller MAE, MSE or both, depending on the cost function that is used.

This strategy was still very coarse, because in the difference matrix ($GT - ED$) there were many elements containing invalid values. The first refinement step was to exclude these values by masking the GT map, which is required to include only useful information, and computing the difference only for the remaining elements. These were:

- invalid disparity values which are set to -1. As said above, the calculation of disparities is far from a perfect procedure, thus many pixels of the left image are not matched with the correspondent pixels of the right image
- GT depth values greater than the maximum measurable distance, which has been identified by Equation 5.1 when the disparity is equal to 1, thus $depth = f \cdot b$.

Once the masking action was done, a further idea was to weight each score of MAE and MSE by the number of useful elements. This way, if the error of one configuration is low but also the number of pixels used to calculate that error is small, that configuration could be worse than another with greater error but also greater number of useful pixels. The weighting function was:

$$MAE = mean(|GT - ED|) \cdot N_{pixel} \tag{5.3}$$

and similarly is for MSE. The coefficient N_{pixel} is the number of useful pixels for current ED map. At code level, it was obtained exploiting the fact that, in Python, by masking a matrix, a vector with just desired values can be outputted. Actually, the real coefficient used in the source code is a scaled version of N_{pixel}, which corresponds to:

$$\frac{N_{pixel}}{TOT_{pixel}} = \frac{N_{pixel}}{H \cdot W} \tag{5.4}$$

where $H = 640$ and $W = 1280$ are respectively height and width of the map, i.e., the resolution of each image.

The final step was to compute the weighted errors on a new subset containing 5000 images of the train dataset. After some analysis, the final range used for *numDisparities* was [96,112,128,144,160,176] and the one for *blockSize* was [9,11,13]. In order to have a clear view of the results, boxplots were created for both MAE and MSE. Once the great part of outliers is removed, the boxplots show how the errors mean and distribution increase with increasing of both parameters: Figure 5.1. Additionally, as already mentioned, great values of $numDisparities$ lead to more overall noise, and great values of $blockSize$ degrade the shape of objects. Nevertheless, as shown in Figure 5.1, the lowest parameter configuration (96,9) does not imply the lowest average error, which is given from configuration (112,9).

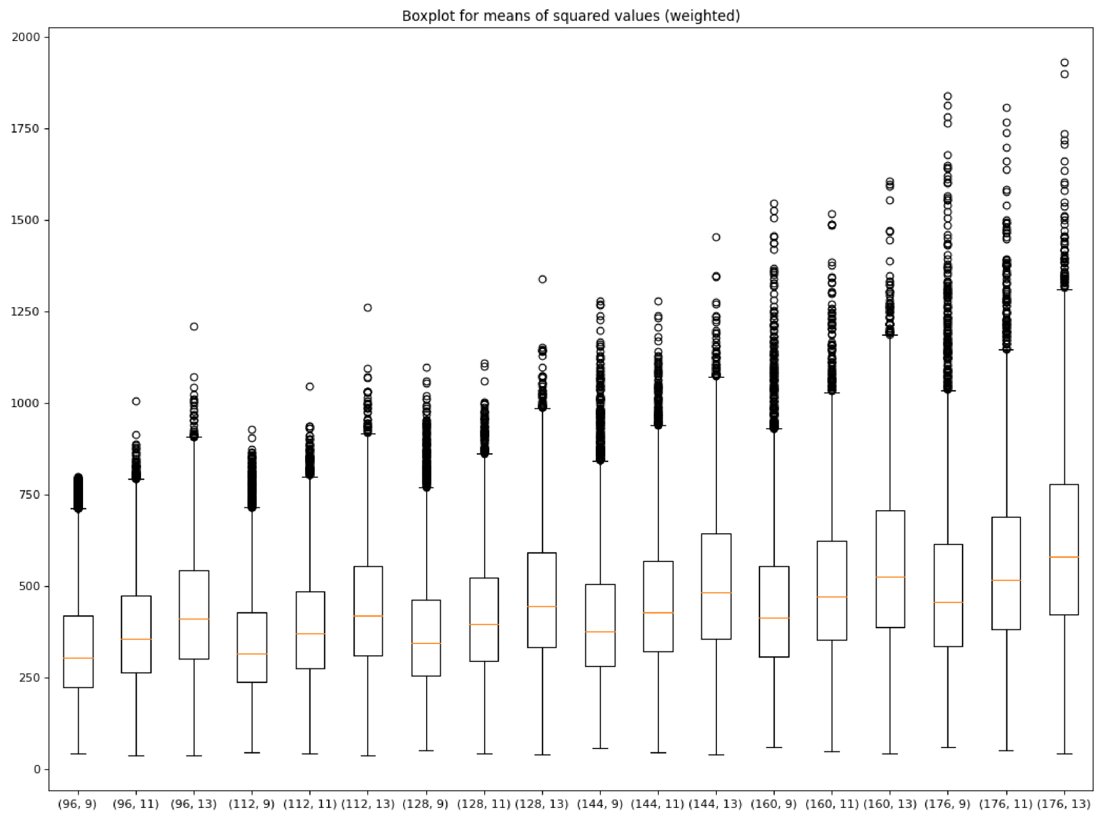

Figure 5.1: Boxplots for weighted MAE. On the horizontal axis there are configurations of the type (*numDisparities,blockSize*). On the vertical axis there are MAE values. Configuration (112,9) gives the lowest overall error.

One last analysis was a simple task, that helped in determining a good trade-off and choosing the final configuration. The whole dataset (30909 images) was analysed, searching for the minimum value of GT distance for each image, and calculating the correspondent minimum required disparity to measure that distance:

$$min\_required\_disp = \frac{f \cdot b}{min(GT)} \tag{5.5}$$

where the $min(\cdot)$ operator find the minimum among all elements of the input matrix.

Afterwards, it was calculated the number of images that required those values of disparity, obtaining Table 5.2, and showing that, e.g., *numDisparities* equal to 96 is sufficient for the 61% of the whole dataset. Because from the boxplots graph the error for 96 and 112 are similarly distributed, as final configuration it was chosen (112,9). From Table 5.2 it is also visible why a good starting value for *numDisparities* is 96, and that 112 is good enough because the 84% of 30909 is still a big number of images.

numDisparities	Occurrences	Rate
80	10	0.03%
96	18781	60.8%
112	25962	84.0%
128	28162	91.1%
144	29979	97.0%
160	30380	98.3%
176	30584	99.0%
192	30687	99.3%
208	30740	99.5%
224	30790	99.6%

Table 5.2: First column is the value of *numDisparities* which have been used. Second column is the number of images that require the related value of *numDisparities* to correctly represent their less distant pixel. Third column is the rate between second column and total number of images (30909).

As mentioned before, the method *stereoSGBM()* has many parameters other than *numDisparities* and *blockSize*. However, in this case the greater part of them is not of interest but for a couple which are thought to control the disparity smoothness, namely $P1$ and $P2$ (see `https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html`). Using these two parameters can effectively reduce amount of noise generated by the standard procedure. Performing a few tests, it was visible that some images, but not all, reached a lower MAE/MSE w.r.t. the case where $P1$ and $P2$ were not used. Despite this result, we'll see that for the final training of the neural network these two parameters are left out, because they led to a lower metrics score.

5.2 Deep learning structure

The next part of the work was focused on the study, use and modification of a deep learning structure, i.e., a deep neural network. To tackle this in Python, it was used PyTorch [47], a popular framework to work in deep learning environments, which is provided of many pre-built classes that ease managing data.

The architecture is an encoder-decoder (see Section 4.1 for the definition), where the encoder is based on MobileNet [48], which is a class of models for mobile and embedded vision applications, and the decoder on DeepLab (Section 3.3.5).

5.2.1 Source code description

The work started from an already existing source code, which has been extended for addressing the computation of estimated depths. The core features using PyTorch are thought to simplify the use of datasets and the processes of training, validation and test of the neural network. Among them, the following stand out:

- **Tensors**: objects used to store multidimensional arrays of any numeric type. They're not directly thought to work in a deep learning framework, but their fundamental feature is they can be run either on CPU or GPU, allowing faster computation in the second case.
- **Dataset**: it's called data primitive. It allows to define a new class for your custom dataset, or even to use pre-loaded ones. It stores samples (like images) and labels (like semantic map).
- **DataLoader**: it's another data primitive. It's used to define an iterable object, which ease accessing dataset's samples, and to set many parameters used during the training/test operations.

The code is enabled to load data from both SELMA and Cityscapes datasets, even if only the first one was used almost for the whole project. Some fast experiments were done on the second at the very end. Furthermore, there are many parameters from command prompt: used dataset, whether to re-scale or crop images and relative numeric factors, used encoder and decoder and size of batches for the DataLoader, to name a few. Furthermore, the code can automatically access three files, containing information about which images to use for training, validation and test. The three splits are as follows: 24735 samples for training set, 3087 samples for validation set and 3087 for test set, for a total number of 30909 samples. All the three split sets are randomly generated.

Moreover, in order to evaluate the metrics discussed in Section 3.1, thus to give a score to the trained model, the code makes use of the so called confusion matrix. The idea is to create a matrix having segmentation classes on both vertical and horizontal axis, one representing the true labels and the other representing the predicted ones. A score is given to each cell: the more this matrix is similar to a diagonal matrix, the more the model predictions are good, because it means a high score is assigned to the cell that matches a prediction with its correct label. The confusion matrix is computed after each epoch of training process, using output predictions, and it allows to easily calculate the metrics mentioned above.

When training a machine learning or deep learning model, two core concepts are: the loss function, which has to be minimized, and the optimizer, an algorithm used to change parameters inside the neural network (such as weights) and to minimize the loss function. The are several existing solutions for both these objects. The main idea is to take predictions (y_{pred}) outputted by the model and compare them with true labels (y_{true}). Some commonly used loss functions are: L1Loss and L2Loss, similar to the cost functions in Equation 5.2, and CrossEntropyLoss:

$$\begin{aligned}
\mathbf{L1Loss} &= \sum_{i=1}^{C} |y_{pred,i} - y_{true,i}| \\
\mathbf{L2Loss} &= \sum_{i=1}^{C} (y_{pred,i} - y_{true,i})^2 \\
\mathbf{CrossEntropyLoss} &= \sum_{i=1}^{C} w_i \cdot \log\left(\frac{\exp(y_{pred,i})}{\sum_{j=1}^{C} \exp(y_{pred,j})}\right) \cdot y_{true,i}
\end{aligned} \tag{5.6}$$

where C is the number of classes and w_i is a possible weight for each class. For the training of this project's model it was used the last one. The logarithmic argument in the cross entropy loss is the output of the *Softmax* layer: the model gives a prediction score, for each class, to each input sample. These scores are normalized by the *Softmax* to make them compatible with a probability distribution (the sum of all class scores must be equal to 1), using exponential functions. In Figure 5.2 there is an example for the image classification task to have a better understanding of this.

$\Rightarrow$

Class	Score	Softmax
Dog	5.0	0.943
Cat	2.0	0.047
Bird	0.5	0.010

Figure 5.2: Application of the Softmax activation function.

Also for model optimizers there are many available choices. Gradient Descent (GD), and its stochastic variant (SGD), is one of the first and most basic algorithm, still widely used in many applications, also during back-propagation in neural networks, for its low complexity of implementation and computation. Another one is Momentum: it introduces an additional hyper-parameter, namely γ, and is used to reduce oscillations, parameters variance and convergence time w.r.t. GD/SGD. For training the current model the chosen optimizer is named Adam [49], which is easy to implement and requires low memory occupation. It is appropriate for the current task, because it works well for problems having a large amount of data, parameters and gradients noise.

5.2.2 Source code modification

The pre-existing code could already load images differentiating among rgb images, depth-maps and semantic-maps. One thing it was missing is the possibility to perform online estimation of depth (during training/test operations). For this reason, the code responsible for depth estimation, i.e., the one developed in Section 5.1, has been imported inside the other one and managed to run during training/test steps. Three new parameters were defined settable from the command prompt: one for enabling online computation of depth, the other two to determine the value of *numDisparities* and *blockSize* inside the *stereoSGBM()* method. From previous analysis, the best combination for the two parameters has been found to be (112,9), determined on the MAE/MSE cost functions for the depth estimation task, and for this reason a new dataset containing all estimated images (30909 as SELMA) has been created. However, this does not ensure the same combination to be the best for the semantic segmentation task. Thus, enabling online computation of depth allows to perform different estimations without storing a complete new dataset for each combination, which would imply an enormous occupation of memory. Adding the computation of depth-maps for each couple of images as an online operation is more time consuming than reading an already existing sample. As an example, if without online computation a single test operation took more or less three minutes, after adding online computation it required roughly ten minutes. Of course, the time-demand problem is subjective to the task and external parameters.

Afterwards, results of training and test steps have been analysed using the visualization toolkit TensorBoard (`https://www.tensorflow.org/tensorboard`), which is provided of many tools to check model's parameters (e.g., weights, biases, loss function) and to show images of input, output (prediction) and reference (GT) samples. This has been quite a helpful step, because allowed to notice an inconsistency between input/reference samples and output ones, which were wrongly cropped and rescaled, leading to poorer model's performances.

Finally, one important feature is the possibility to keep track of results (score tables) also after the environment, where the training is performed, has been closed. Of course, the model has to be saved, but also creating a log file is fundamental. This allows to know the evolution of the training during each epoch. A useful guideline, applied during this work, is to create different directory paths based on date when process is started, whether training or test is performed, and other parameters, so to ease the search for desired log file. Also, creating a log directory is essential to use the above mentioned TensorBoard, which reads information from there.

6
Results

In this chapter we analyse some results obtained during estimation of depth-maps, of training the neural network on them, and relative tests.

Figure 6.1 shows an example of disparity-map from a random sample of SELMA. The parameters used are the optimal ones (*numDisparities* = 112, *blockSize* = 9). Note that this is considered a disparity-map because near objects are brighter than far objects, implying pixels of the first group have a higher value of gray.

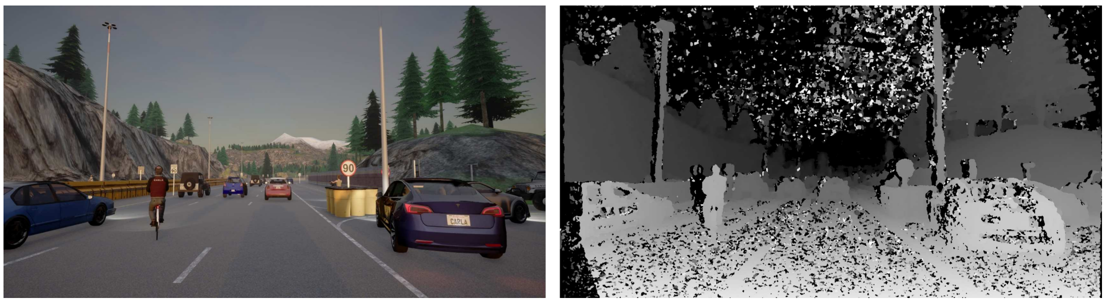

Figure 6.1: SELMA sample (on the left) and relative disparity-map (on the right).

Figure 6.2 shows two problematic situations. On the left, the disparity-map has been generated using a very low *numDisparities* = 16, thus nearest objects are not identifiable. On the right, the map is generated using a too high value of *blockSize* = 35, leading to destruction of objects shape.

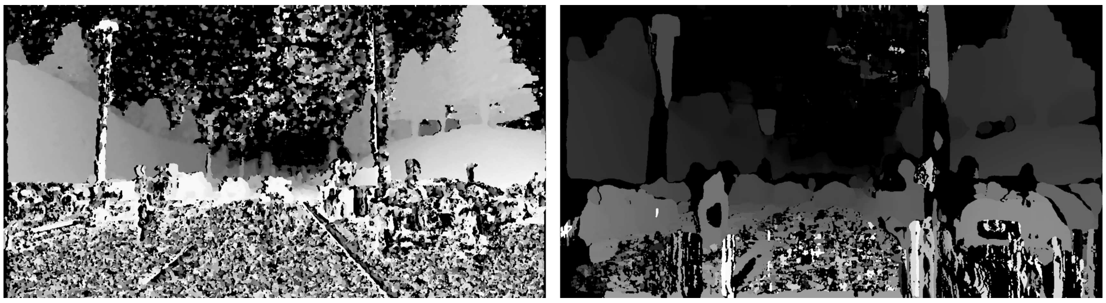

Figure 6.2: Erroneous depth estimation: too low numDisparities (on the left) and too high blockSize (on the right).

Figure 6.3 is the same estimation of Figure 6.1 performed using the parameters $P1$ and $P2$ mentioned in Section 5.1. Using specific values for these parameters should lead to an optimal smoothing action.

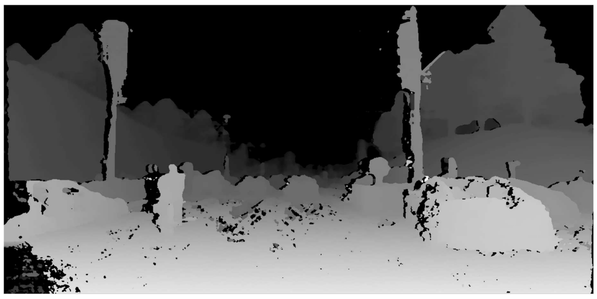

Figure 6.3: Applying best smoothing. P1 and P2 work as penalties for disparity change of pixels in the window. P1 is penalty by ± 1 while P2 is penalty by more than 1. P2 must be greater than P1.

Figure 6.4 shows ideally how a disparity-map should appear: noiseless, with perfect edges, generally shaped like the original image. The estimated version shows instead the presence of many noisy artifacts, due also to stereo vision it-self (e.g., that shadow-like effect visible on the left of various objects and in the bottom left corner of the image).

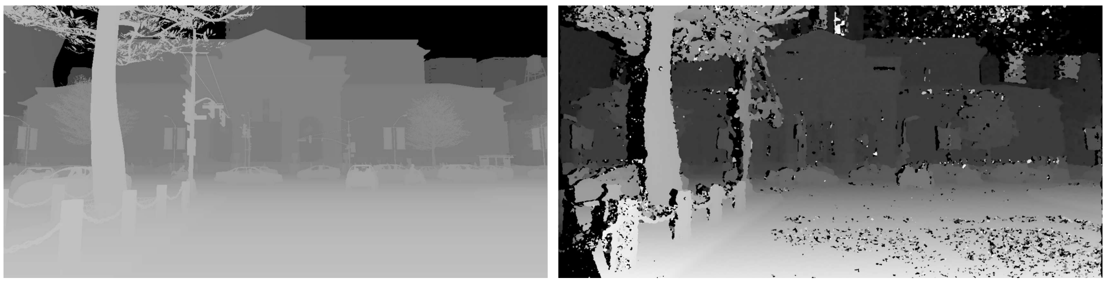

Figure 6.4: Ground truth disparity-map (on the left) and estimated one (on the right).

Table 6.1 shows the training scores for each class, and their average, for the three metrics: Pixel Accuracy (Recall), Pixel Precision and Intersection over Union.

Class	PA%	PP%	IoU%
road	99.5	99.3	98.8
sidewalk	95.3	96.3	91.9
building	96.7	96.8	93.7
wall	96.7	95.8	92.8
fence	80.4	81.7	68.2
pole	73.3	83.3	63.9
traffic light	73.2	82.3	63.2
traffic sign	81.1	89.5	74.0
vegetation	94.2	92.3	87.3
terrain	91.4	94.2	86.5
sky	98.8	98.8	97.6
person	85.1	82.8	72.3
rider	77.9	81.7	66.3
car	97.5	96.7	94.4
truck	82.4	84.3	71.5
bus	91.8	85.2	79.1
train	92.2	90.3	83.9
motorbike	83.1	85.4	72.7
bicycle	55.7	59.8	40.5
Average	86.6	88.2	**78.9**
Std.Dev.	11.4	9.3	15.1

(a) Training score GT.

Class	PA%	PP%	IoU%
road	98.4	98.7	97.1
sidewalk	84.4	93.3	79.6
building	91.1	90.1	82.8
wall	88.1	92.8	82.5
fence	63.4	71.5	50.6
pole	61.8	69.3	48.5
traffic light	53.0	67.2	42.1
traffic sign	59.0	81.3	51.9
vegetation	87.4	82.9	74.0
terrain	82.1	83.2	70.4
sky	96.8	97.1	94.1
person	65.2	72.2	52.1
rider	63.0	73.3	51.2
car	94.8	90.7	86.4
truck	63.6	81.5	55.6
bus	77.8	61.6	52.4
train	76.9	89.0	70.3
motorbike	60.5	78.4	51.9
bicycle	36.0	45.4	25.1
Average	73.9	80.0	**64.1**
Std.Dev.	17.1	13.5	19.5

(b) Training score ED.

Table 6.1: Scores for the network trained with GT data (a) and with ED data (b). Training is done in many epochs, these values come from the best one.

Scores are coloured depending on how far their value is from the average. Distance from the average is determined by the standard deviation. Colour-code is shown in Figure 6.5, where a Gaussian is represented being μ the average value and σ the standard deviation.

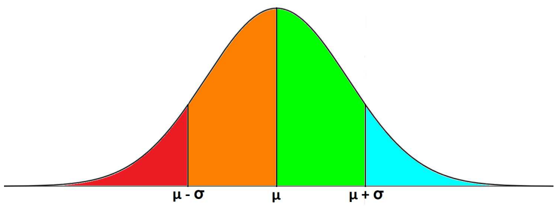

Figure 6.5: Colour code for tables containing scores. Values are grouped depending on how much they differ from the average.

Table 6.2 shows the test scores for the metrics when the network is tested on the same type of data used for training (GT $\rightarrow$ GT, ED $\rightarrow$ ED).

Class	PA%	PP%	IoU%
road	99.5	99.2	98.7
sidewalk	95.2	96.1	91.7
building	96.9	97.0	94.0
wall	96.5	95.5	92.4
fence	80.6	82.1	68.6
pole	73.8	83.4	64.4
traffic light	73.3	81.1	62.6
traffic sign	80.8	89.8	74.0
vegetation	94.1	92.3	87.3
terrain	91.3	94.3	86.5
sky	98.8	98.8	97.6
person	85.9	83.4	73.3
rider	79.0	82.0	67.3
car	97.6	96.7	94.4
truck	85.1	84.0	73.2
bus	93.0	92.9	86.8
train	92.0	88.3	82.0
motorbike	80.8	84.5	70.3
bicycle	58.0	61.3	42.5
Average	87.0	88.6	**79.3**
Std.Dev.	11.0	9.1	14.8

(a) Test score GT $\rightarrow$ GT.

Class	PA%	PP%	IoU%
road	98.3	98.7	97.0
sidewalk	83.8	93.2	78.9
building	91.1	91.1	83.7
wall	87.5	92.1	81.4
fence	63.6	71.7	50.8
pole	61.8	69.6	48.7
traffic light	52.1	67.0	41.5
traffic sign	57.7	81.3	50.9
vegetation	87.5	82.6	73.9
terrain	82.1	82.9	70.2
sky	96.9	97.1	94.2
person	67.3	73.5	54.1
rider	64.0	73.2	51.9
car	94.9	90.9	86.7
truck	71.4	82.2	61.9
bus	81.4	65.7	57.1
train	76.0	88.4	69.1
motorbike	51.4	76.2	44.3
bicycle	37.0	44.2	25.2
Average	74.0	80.1	**64.3**
Std.Dev.	17.4	13.4	19.5

(b) Test score ED $\rightarrow$ ED.

Table 6.2: Scores for the GT network tested on GT data (a) and the ED network tested on ED data (b).

These tests confirm the efficacy of the network, in both cases. Using GT data the mIoU's score reached almost 80% which is a very good result, and using ED data 64% is quite satisfying, considering that no adaptation techniques are applied, thus this result is likely to be improved.

Table 6.3 shows the test scores for the metrics when the network is tested on a different type of data from the training one (GT $\longrightarrow$ ED, ED $\longrightarrow$ GT).

Class	PA%	PP%	IoU%
road	5.8	98.7	5.8
sidewalk	0.1	91.6	0.1
building	1.0	22.7	1.0
wall	1.0	62.5	1.0
fence	0	61.0	0
pole	2.8	61.4	2.7
traffic light	4.4	31.6	4.0
traffic sign	3.6	91.2	3.6
vegetation	86.8	14.3	14.0
terrain	2.8	70.9	2.8
sky	66.9	76.8	55.7
person	10.6	58.8	9.9
rider	10.3	82.7	10.0
car	0.1	75.8	0.1
truck	0	4.3	0
bus	0	0.1	0
train	0	0	0
motorbike	0.8	8.1	0.7
bicycle	0	6.7	0
Average	10.4	48.4	**5.9**
Std.Dev.	23.9	35.1	12.7

(a) Test score GT $\longrightarrow$ ED.

Class	PA%	PP%	IoU%
road	10.5	89.8	10.4
sidewalk	47.5	84.2	43.6
building	95.8	74.7	72.3
wall	57.3	82.9	51.2
fence	40.1	80.8	36.6
pole	32.2	75.0	29.1
traffic light	33.3	73.8	29.1
traffic sign	52.7	84.8	48.2
vegetation	67.9	42.7	35.5
terrain	80.3	22.4	21.2
sky	98.6	93.0	91.8
person	45.8	89.3	43.5
rider	44.0	83.8	40.6
car	68.1	94.8	65.6
truck	73.1	14.6	13.8
bus	14.9	25.1	10.3
train	0.4	50.8	0.4
motorbike	24.9	84.8	23.8
bicycle	8.2	39.2	7.3
Average	47.1	67.7	**35.5**
Std.Dev.	28.7	26.2	23.8

(b) Test score ED $\longrightarrow$ GT.

Table 6.3: Scores for the GT network tested on ED data (a) and the ED network tested on GT data (b).

Test of Table 6.3a shows the inefficacy of the network trained on GT data when used on ED data. Many classes (train, bus, truck) are not even recognized (scores equal zero are coloured in black). This is a consequence of the domain shift issue mentioned in Chapter 4; a visual change in the data can mean a catastrophic failure of the model. Table 6.3b shows a test in the opposite situation. Although mIoU = 35.5 is not a good result w.r.t. the supervised situation, it is way better than the previous case; some classes are well recognized considering the mIoU of the network from training is 64%. This is particularly good knowing that the model is trained source-only, and it suggests that improvements of the network are possible if adaptation techniques are applied.

Table 6.4a shows what happened when the network was trained using ED-maps calculated using the parameters $P1$ and $P2$ for best smoothing (see Figure 6.3); despite the removal of noise, performance are less effective than without using those parameters. Table 6.4b shows instead a test of the neural network trained on ED-maps (the one of Table 6.1b) on the Cityscapes dataset.

Class	PA%	PP%	IoU%
road	98.1	98.5	96.7
sidewalk	86.1	90.7	79.1
building	89.7	88.3	80.1
wall	86.4	90.4	79.1
fence	59.2	66.4	45.6
pole	50.4	67.7	40.6
traffic light	37.4	69.1	32.0
traffic sign	51.5	80.5	45.8
vegetation	85.0	80.0	70.1
terrain	76.8	84.1	67.0
sky	96.1	95.5	92.0
person	59.1	72.8	48.4
rider	60.4	72.2	49.0
car	94.4	89.8	85.3
truck	64.0	65.2	47.7
bus	79.0	61.9	53.2
train	73.3	77.9	60.7
motorbike	60.0	72.2	48.7
bicycle	37.6	44.1	25.5
Average	70.8	77.2	**60.3**
Std.Dev.	19.2	13.6	20.6

(a) Train on EDs generated with $P1$ and $P2$.

Class	PA%	PP%	IoU%
road	65.4	97.2	64.2
sidewalk	9.9	69.6	9.5
building	64.4	65.8	48.3
wall	12.7	10.2	6.0
fence	0.1	92.1	0.1
pole	16.3	30.4	11.9
traffic light	3.0	7.1	2.2
traffic sign	1.8	51.8	1.8
vegetation	48.8	51.2	33.3
terrain	75.2	2.5	2.5
sky	73.9	33.4	29.9
person	16.8	62.0	15.2
rider	2.4	28.6	2.3
car	11.3	94.4	11.3
truck	18.1	18.0	9.9
bus	0	0	0
train	0	0	0
motorbike	0.5	4.9	0.5
bicycle	1.0	59.2	1.0
Average	22.2	41.0	**13.1**
Std.Dev.	27.7	33.5	18.1

(b) Test score ED-SELMA $\longrightarrow$ ED-Cityscapes.

Table 6.4: Scores for training on ED data obtained through best smoothing (a) and test on Cityscapes (b).

It is interesting to notice the result on the Cityscapes dataset. Despite the low score, again here no adaptation strategies have been applied, and here not only the target dataset is different from the source one, but they're also of different type (SELMA synthetic, Cityscapes real-world). The domain shift is then very hard to handle, but considering that the GT based network reached roughly 2-3% mIoU (results are not reported) on the same task, this is a pretty good result for future developments.

7 Conclusion

This thesis tackled two main tasks: first, the analysis of depth estimation through stereo vision, code development for research of best parameters of the stereo vision algorithm on the SELMA dataset, and the study of deep learning frameworks, training and testing a convolutional neural network, designed with the autoencoder architecture, on both ground truth and estimated depth-maps.

The first part was a preliminary analysis and it required a certain amount of work to be completed. Code development has not been straightforward, many versions have been worked out, not always working solutions, thus implying the necessity of debugging them. In addition, a lot of time was spent by the application to run on a CPU device. Considering the *stereoSGBM()* method good enough, not particular care was taken on refining the disparity-maps (or the depth-maps). However, post-processing ideas can be thought to further optimize the maps that are fed as input to the neural network, and additional researches can be done to effectively understand whether a parameter configuration of the estimation method is better than another for the semantic segmentation task.

The second part concerns the introduction of depth-map computations inside the training and test of the neural network. Initially, an estimated version of SELMA was created in order to quickly access to estimated samples, but some data-type problems occured while saving the maps, thus making the whole new dataset useless. Adding the online computation of depths helps avoiding this problem and allows to perform estimation with different parameter configurations. Unlike the first part, this was done on a computer equipped with a GPU, which is able to perform fast computation of images. From a practical point of view, this was almost mandatory because of time-demanding reasons.

Despite obtained results can be considered good for a master thesis project, there's still a lot of room for improvements and different solutions. For example, deep learning can be exploited to perform stereo vision too, and many works have already addressed this task. One of the most recent and promising model is CREStereo [50]: the authors explain as their solution outstand all previous ones, but also that it is very computational demanding. In fact, one problem with deep learning structures can be the heavy computation they need (intuitively, the deeper the havier). Making such structures lighter is then of fundamental importance if it is required to run them on mobile devices.

Another possible approach is monocular depth estimation: in this case stereo vision is no more used. Instead, depth-maps are generated using a single image and exploiting domain adaptation strategy called multi-task learning (or multi-tasking learning). Its analysis though is out of the scope of this thesis.

References

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[3] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[4] M. Toldo, A. Maracani, U. Michieli, and P. Zanuttigh, "Unsupervised domain adaptation in semantic segmentation: a review," *Technologies*, vol. 8, no. 2, p. 35, 2020.

[5] F. Barbato, M. Toldo, U. Michieli, and P. Zanuttigh, "Latent space regularization for unsupervised domain adaptation in semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2835–2845.

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[7] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[8] P. Testolina, F. Barbato, U. Michieli, M. Giordani, P. Zanuttigh, and M. Zorzi, "Selma: Semantic large-scale multimodal acquisitions in variable weather, daytime and viewpoints," *arXiv preprint arXiv:2204.09788*, 2022.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[10] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[11] Y. Sun, W. Zuo, and M. Liu, "Rtfnet: Rgb-thermal fusion network for semantic segmentation of urban scenes," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2576–2583, 2019.

[12] Y. Zhang, D. Sidibé, O. Morel, and F. Mériaudeau, "Deep multimodal fusion for semantic image segmentation: A survey," *Image and Vision Computing*, vol. 105, p. 104042, 2021.

[13] O. Faugeras and O. A. Faugeras, *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.

[14] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," *arXiv preprint arXiv:1704.06857*, 2017.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[18] M. Everingham, S. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[20] A. Gupta, P. Dollar, and R. Girshick, "Lvis: A dataset for large vocabulary instance segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5356–5364.

[21] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille, "The role of context for object detection and semantic segmentation in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 891–898.

[22] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille, "Detect what you can: Detecting and representing objects using holistic models and body parts," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1971–1978.

[23] J. M. Alvarez, T. Gevers, Y. LeCun, and A. M. Lopez, "Road scene segmentation from a single image," in *European Conference on Computer Vision*. Springer, 2012, pp. 376–389.

[24] R. Zhang, S. A. Candra, K. Vetter, and A. Zakhor, "Sensor fusion for semantic segmentation of urban scenes," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 1850–1857.

[25] G. Ros, S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A. M. Lopez, "Vision-based offline-online perception paradigm for autonomous driving," in *2015 IEEE Winter Conference on Applications of Computer Vision*. IEEE, 2015, pp. 231–238.

[26] G. Ros and J. M. Alvarez, "Unsupervised image transformation for outdoor semantic labelling," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 537–542.

[27] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3234–3243.

[28] A. Asvadi, L. Garrote, C. Premebida, P. Peixoto, and U. J. Nunes, "Multimodal vehicle detection: fusing 3d-lidar and color camera data," *Pattern Recognition Letters*, vol. 115, pp. 20–29, 2018.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[30] W. Yu, K. Yang, Y. Bai, T. Xiao, H. Yao, and Y. Rui, "Visualizing and comparing alexnet and vgg using deconvolutional layers," in *Proceedings of the 33 rd International Conference on Machine Learning*, 2016.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets*. Springer, 1990, pp. 286–297.

[34] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected crfs with gaussian edge potentials," *Advances in neural information processing systems*, vol. 24, 2011.

[35] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

[36] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?" in *2016 international conference on digital image computing: techniques and applications (DICTA)*. IEEE, 2016, pp. 1–6.

[37] X. Shen, A. Hertzmann, J. Jia, S. Paris, B. Price, E. Shechtman, and I. Sachs, "Automatic portrait segmentation for image stylization," in *Computer Graphics Forum*, vol. 35, no. 2. Wiley Online Library, 2016, pp. 93–102.

[38] J. Jiang and C. Zhai, "Instance weighting for domain adaptation in nlp." ACL, 2007.

[39] F. Fang, K. Dutta, and A. Datta, "Domain adaptation for sentiment classification in light of multiple sources," *INFORMS Journal on Computing*, vol. 26, no. 3, pp. 586–598, 2014.

[40] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*. Springer, 2016, pp. 102–118.

[41] G. Neuhold, T. Ollmann, S. Rota Bulo, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 4990–4999.

[42] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 year, 1000 km: The oxford robotcar dataset," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 3–15, 2017.

[43] Y.-H. Chen, W.-Y. Chen, Y.-T. Chen, B.-C. Tsai, Y.-C. Frank Wang, and M. Sun, "No more discrimination: Cross city adaptation of road scene segmenters," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1992–2001.

[44] G. Van Rossum and F. L. Drake, *Python 3 reference manual*. CreateSpace, 2009.

[45] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[46] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 807–814.

[47] N. Ketkar and J. Moolayil, "Introduction to pytorch," in *Deep learning with python*. Springer, 2021, pp. 27–91.

[48] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[50] J. Li, P. Wang, P. Xiong, T. Cai, Z. Yan, L. Yang, J. Liu, H. Fan, and S. Liu, "Practical stereo matching via cascaded recurrent network with adaptive correlation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 263–16 272.

A

Appendix A

This appendix contains examples of the core functions of the project, plus some further examples of estimated disparity-maps.

Listing A.1 shows all imported libraries at some point during code development (not all of them were always called together).

Listing A.2 shows the function used to compute disparity-maps.

Listing A.3 shows the function used to compute depth-maps and create a path to store them, if required.

Figure A.1 shows another example of disparity-maps estimated with different parameter configurations.

Figure A.2 shows another example of disparity-maps estimated with different parameter configurations and best smoothing (parameters $P1$ and $P2$).

Figure A.3 shows a semantic prediction after performing a test in the four situations (whether the network or the test data are GT or ED based).

```python
import numpy as np
import cv2 as cv
import math
import os
from os.path import join
import sys
import pandas as pd
from tqdm import tqdm
import time
import csv
import random
import shutil
import logging
import datetime
from tensorboardX import SummaryWriter
from PIL import Image, ImageFile

#pytorch
import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn
import torch.nn.functional as F
from torch.nn import CrossEntropyLoss, L1Loss, MSELoss
```

Listing A.1: Import of required libraries.

```python
def compute_disparity(imgL, imgR, numDisparities, blockSize):
    #create the stereo object with minimum disparity set to 0
    stereo = cv.StereoSGBM_create(0,numDisparities,blockSize)

    #compute disparity
    #create a version of the two images with all starting columns (the first numDisparities columns) set to 0
    #this way the stereo computation does not clip the final result
    blackL = np.concatenate((np.zeros((imgL.shape[0],numDisparities),dtype=np.uint8),imgL),axis=1)
    blackR = np.concatenate((np.zeros((imgR.shape[0],numDisparities),dtype=np.uint8),imgR),axis=1)
    disparity = stereo.compute(blackL,blackR)

    #now remove the first disp columns, because they have only useless values
    disparity = disparity[:,numDisparities:].astype(np.float32)/16  #here it is assumed that the minimum disparity is 0

    return disparity
```

Listing A.2: Function to compute disparity.

```python
def depth_estimator(lr_path, numDisparities, blockSize, f, b, est_dir = None):

    #left and right images loaded in gray scale
    imgL = cv.imread(lr_path[0], flags=0)
    imgR = cv.imread(lr_path[1], flags=0)
    #call the function to compute disparities
    disparity = compute_disparity(imgL, imgR, numDisparities, blockSize)

    est_path = None
    if est_dir is not None:
        #path where to save estimated images
        est_path = [est_dir+lr_path[0][str.find(lr_path[0],'\\'):-4]+'.png',
                    est_dir+lr_path[0][str.find(lr_path[0],'\\'):-4]+'.png']
        est_path[0] = est_path[0].replace('CAM_FRONT_LEFT','DISPARITY')
        est_path[1] = est_path[1].replace('CAM_FRONT_LEFT','DISTANCE')

    #set the invalid (-1) disparity value to 0, which becomes the common value for
     invalid elements (leads to infinity depth)
    distance = np.zeros_like(disparity)
    mask = disparity > 0
    distance[mask] = f*b/disparity[mask]

    return disparity, distance, est_path
```

Listing A.3: Function to compute depth.

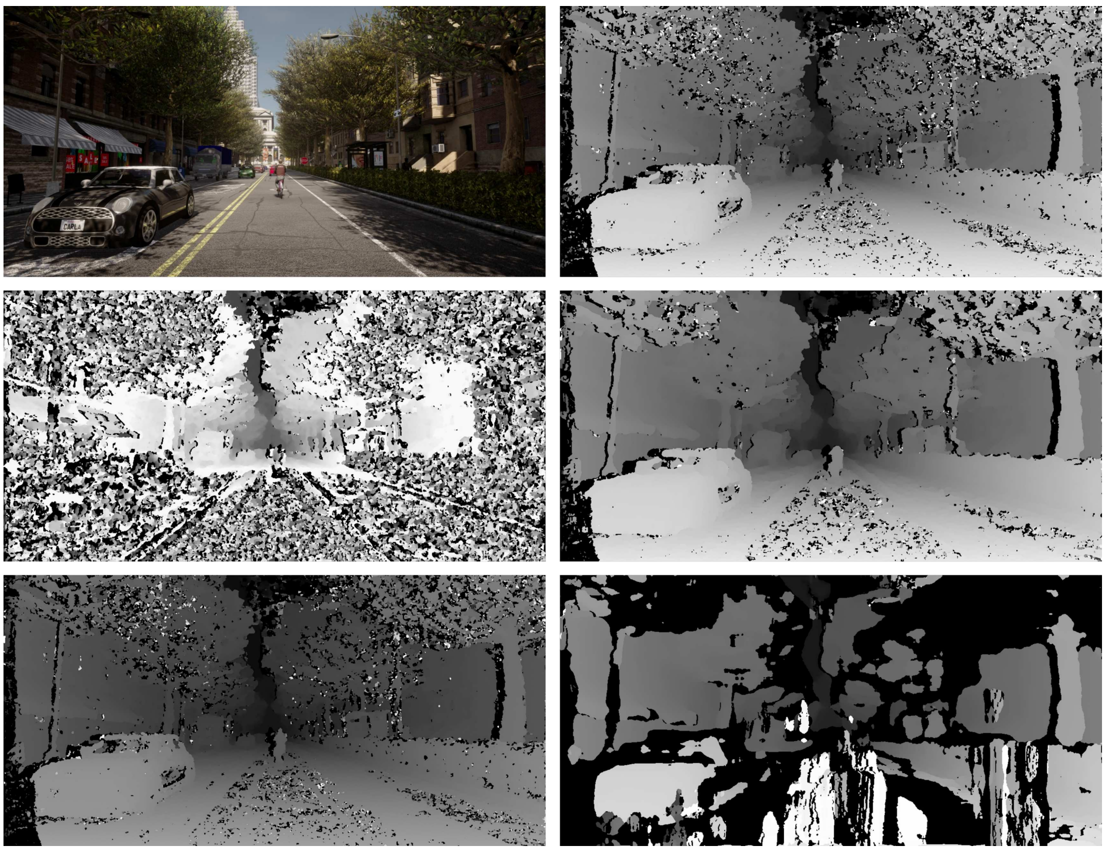

Figure A.1: From left to right and from top to bottom, the parameters configuration is (numDisparities,blockSize)=None,(112,9),(16,9),(112,17),(256,9),(112,35).

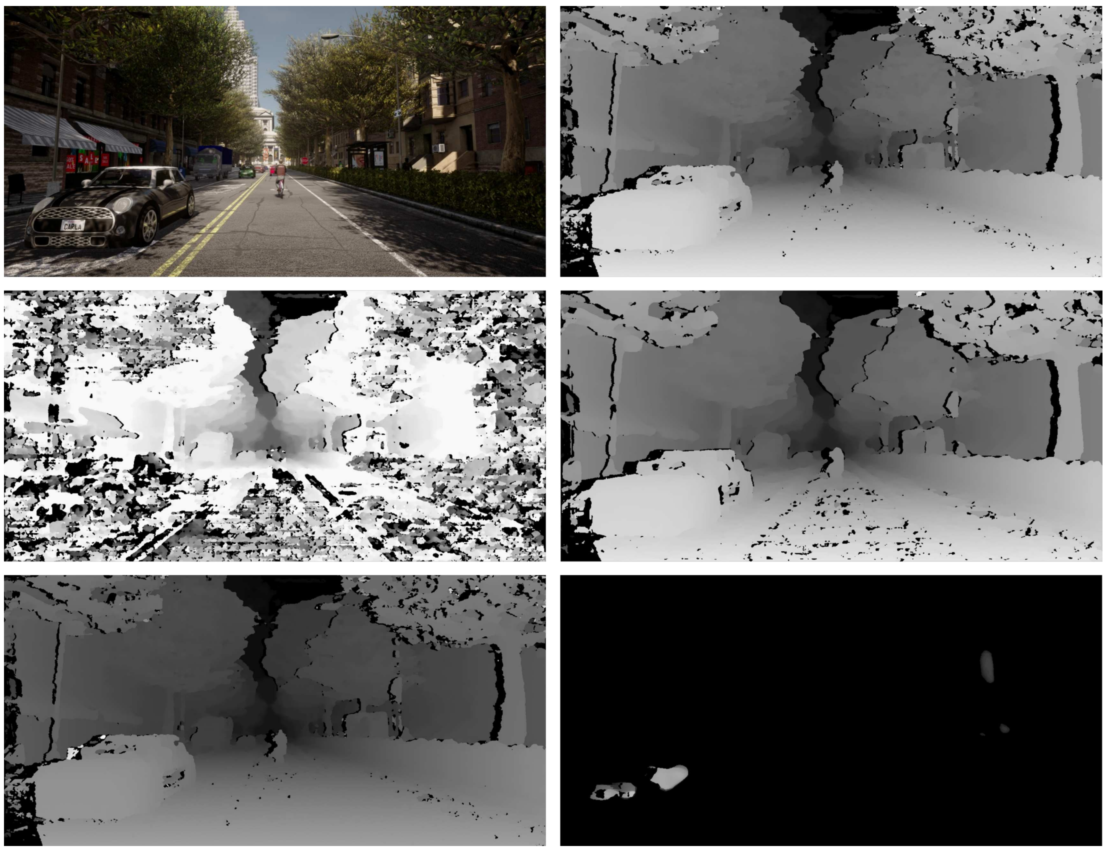

Figure A.2: From left to right and from top to bottom, the parameters configuration is (numDisparities,blockSize)=None,(112,9),(16,9),(112,17),(256,9),(112,35).

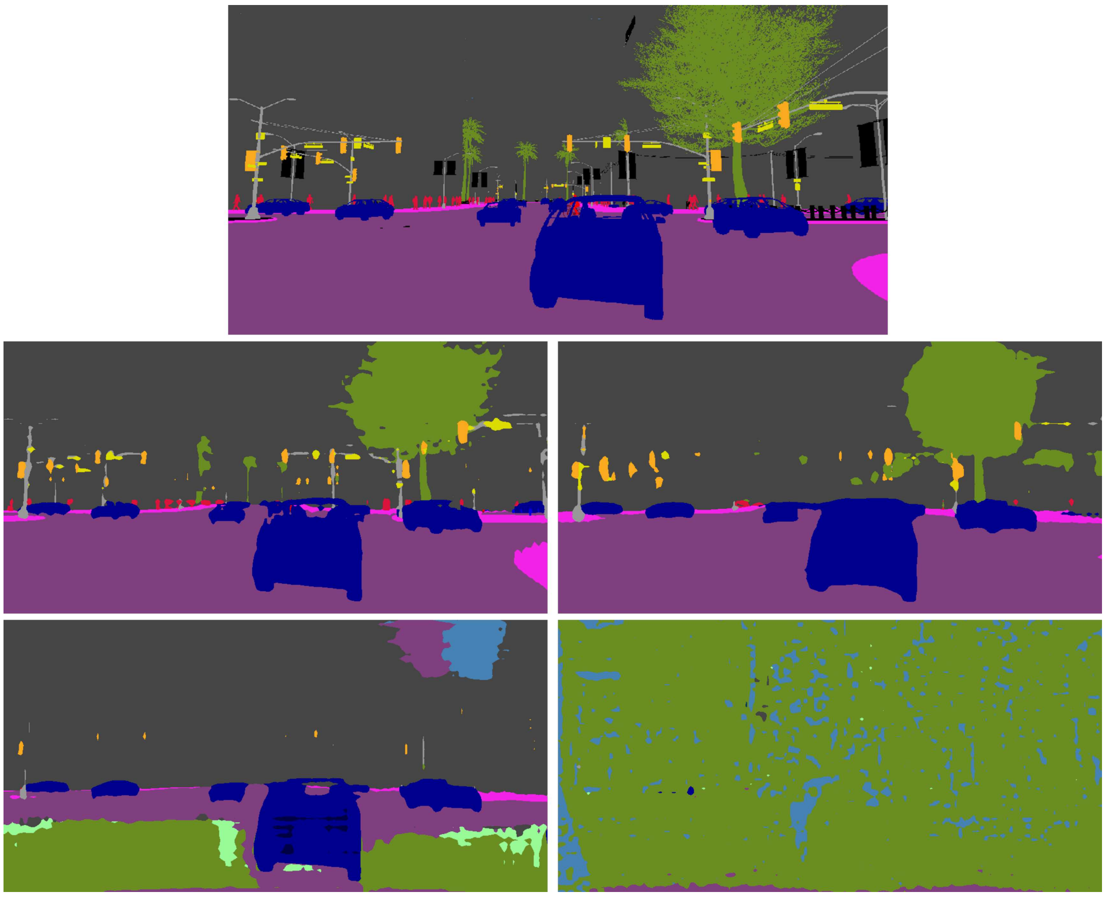

Figure A.3: On top, the semantic labelled GT sample. Then, from left to right and from top to bottom: prediction of GT network on GT data, prediction of ED network on ED data, prediction of ED network on GT data, prediction of GT network on ED data.