POLYTECHNIC OF TURIN

Master's Degree in Data science and engineering

Master's Degree Thesis

DA-PanopticFPN: a panoptic segmentation model to bridge the gap between simulated and real autonomous driving perception data

Supervisors

Prof. Nicola AMATI

Prof. Carla Fabiana CHIASSERINI

Prof. Massimiliano GOBBI

Candidate

Christian CANCEDDA

April 2022

Acknowledgements

To my parents who always supported my choices and always helped me achieve my objectives. To my long time friend, more than a decade already, Mattia with whom i shared many adventures, some of which i forgot some parts.
To my long time and distant friends Giacomo and Marco that made many of the past years and especially the last two less stressful, depending on the matches that we played together.
To my friends and colleagues Matteo, Federico, Federica and Martina, met during the Bachelor years in-between entertaining APA lab session.
To Martina, Sofia, Cristiano and Bianca with whom i shared my time during the Master's degree, both in presence and in video calls. One peculiar acknowledgment also to the iron lifted in the past two years, that helped me get through lasting epidemic.

And finally a major thanks to my supervisors, the Professors Nicola Amati, Carla Fabiana Chiasserini and Massimiliano Gobbi, who gave me the opportunity and freedom, under their guidance, to develop this research dissertation.

The code of all the work is made available at: https://github.com/Chris1nexus/da-panopticfpn

Table of Contents

List of Tables IX

List of Figures X

1 Introduction 1

1.1 Simulation for autonomous driving 1
1.2 Research setting and main motivations 3
1.3 Thesis outline 5

2 Related works 7

2.1 Autonomous driving scene understanding 7
2.1.1 Object detection 9
2.1.2 Semantic segmentation 9
2.1.3 Instance segmentation 10
2.1.4 Panoptic segmentation 10
2.2 Scene understanding datasets 11
2.2.1 Synthetic autonomous driving datasets 12
2.2.2 Real world autonomous driving datasets 13
2.3 Deep learning perception 15
2.3.1 Convolutional neural network structure 16
2.3.2 Basic mathematical operations 18
2.3.3 Backbone architectures 27
2.3.4 Feature pyramid network architectures 33
2.3.5 Multi-task learning 39
2.3.6 Instance segmentation models 40
2.3.7 Semantic segmentation models 43
2.3.8 Panoptic Segmentation models 45
2.4 Self-supervised Domain Adaptation 47
2.4.1 Domain adaptation methods 48

3 Synthetic-to-Real dataset creation methodology 49
3.1 Generation of a label-matched simulated and real world image dataset 49
3.2 Synthetic dataset creation 50
3.2.1 The CARLA simulator 50
3.2.2 Synthetic driving scenarios 53
3.2.3 Sensor setup 55
3.2.4 Synthetic images and ground truth formats 57
3.3 Synthetic-to-Real dataset 58
3.3.1 Multi dataset label matching 62

4 Model design 69
4.1 Introduction 69
4.2 Backbone 71
4.2.1 ResNet feature extractor 71
4.2.2 FPN 74
4.3 Semantic segmentation head 76
4.4 Mask R-CNN head 77
4.4.1 Region Proposal Network 77
4.4.2 ROI align 79
4.4.3 Bounding Box regression and Object classification branch . . 80
4.4.4 Instance segmentation branch 81
4.5 Domain Adaptation branches 82
4.5.1 Adversarial Self-supervised Domain Adaptation 82
4.5.2 Gradient Reversal Layer 83
4.5.3 Gradient Reversal training schedule 84
4.6 DA-PanopticFPN model implementation 84
4.6.1 Domain classifier implementation 88
4.7 Optimization of the model parameters 91
4.7.1 Introduction 91
4.7.2 Semantic Segmentation loss 92
4.7.3 Mask R-CNN loss 93
4.7.4 RPN loss 95
4.7.5 Domain classifiers loss 96
4.7.6 Joint loss 98

5 Experiments 99
5.1 Experimental context 99
5.2 Metrics: panoptic quality PQ 100
5.2.1 Segmentation quality SQ and recognition quality RQ 101
5.3 Experiment design 101
5.4 Model training setup 104

5.5 Experiment results 106
5.5.1 Result summary 110

6 Conclusions and further work 116

Bibliography 119

List of Tables

3.1 Synthetic dataset mapping from default labels to label-matched version 63
3.2 Cityscapes mapping from default labels to label-matched version . . 65
3.3 BDD100k mapping from default labels to label-matched version . . 67

5.1 Domain adaptation experiment metrics, sorted by decreasing panoptic quality on Cityscapes validation set(cPQ) 107
5.2 Domain adaptation experiments reported in **percentage change of the metrics with respect to the domain adaptation baseline(baseline 1)**, sorted by decreasing panoptic quality on Cityscapes validation set(cPQ) 108
5.3 Domain adaptation experiments reported in **percentage change of the metrics with respect to the Cityscapes baseline**(baseline 2) , sorted by decreasing panoptic quality on Cityscapes validation set(cPQ) 109

List of Figures

1.1 SAE Levels of Driving Automation 2

2.1 This figure illustrates the relationships between semantic segmentation, instance segmentation(and the simpler object detection), with the panoptic segmentation objective [50] 11
2.2 Example of a computation graph 17
2.3 Illustration of the 2D convolution operator, from the paper [79] . . 20
2.4 LeNet-5 architecture[85] 28
2.5 AlexNet architecture[84]. Each block represents a convolutional layer, while subsampling layers are denoted by the written captions. The terminal fully connected layers are instead denoted in the image as dense, an alternative nomenclature to the layer type 28
2.6 Bottleneck block proposed in [93]. Credits [93] 30
2.7 Side by side comparison between the bottleneck block of the ResNet architecture on the left and the ResNext bottleneck with cardinality $C = 32$. Image from [94] 32
2.8 FCN architecture. Image from [102] 33
2.9 U-Net architecture. Image from [103] 34
2.10 FPN architecture. Image from [107] 35
2.11 U-Net(top-left), FPN(top-right) and FCN(bottom) architectures. Respective credits[103] [107] and [102] 36
2.12 Side by side comparison of FPN on the left and PANet on the right. Image from [110] 37
2.13 NAS-FPN architecture. Image from [110] 38
2.14 BiFPN architecture. Image from [110] 38
2.15 EfficientDet architecture. Image from [110] 39

3.1 A diverse set of scenes can be captured as images by the CARLA simulator: weather and daylight conditions, sun glares, road reflectance of light and traffic density are only a few of the features that can be simulated (source[6]). 50

3.2 Multi camera sensor setup employed in Tesla vehicles 55
3.3 Multi-view camera frame: front-left (top-left), front (top-center), front-right(top-right), rear-left (mid-left), rear (center), rear-right (mid-right), long range(bottom) 57
3.4 Left camera frame: image (left), instance mask(center), semantic mask(right), panoptic mask (bottom) 58
3.5 Synthetic dataset sample with its label-matched panoptic mask . . 63
3.6 Cityscapes sample with its label-matched panoptic mask 66
3.7 BDD100k sample with its label-matched panoptic mask 68

4.1 Residual function of a Resnet architecture, implemented as a neural network computation graph [93] 72
4.2 Side by side comparison of the residual learning block and its more efficient Bottleneck version [93]. 73
4.3 Side by side comparison of the bottleneck based resnet models: ResNet18,ResNet34,ResNet50,ResNet101, ResNet152 [93]. 73
4.4 Semantic segmentation branch 76
4.5 Self-supervised adversarial domain adaptation through gradient reversal. Credits: [140] 82
4.6 Semantic segmentation branch 85
4.7 DA PanopticFPN architecture 87

5.1 Polynomial decay schedule, obtained with the provided configuration 105
5.2 DA PanopticFPN architecture with the optimal domain classifiers attached respectively to C_2 and C_5 110
5.3 PQ metrics of the best model on validation Cityscapes(left) and BDD100k(right) 111
5.4 PQ metrics of the best model on validation Cityscapes(top) and BDD100k(bottom) 111
5.5 Train and validation losses: the train total loss on Synthetic-CARLA(bottom), validation total loss Synthetic-CARLA(top-left), Cityscapes (top-center), BDD100k(top-right) 112
5.6 validation Cityscapes panoptic predictions of the best configuration 113
5.7 validation BDD100k panoptic predictions of the best configuration 114
5.8 validation Synthetic-CARLA panoptic predictions of the best configuration 115

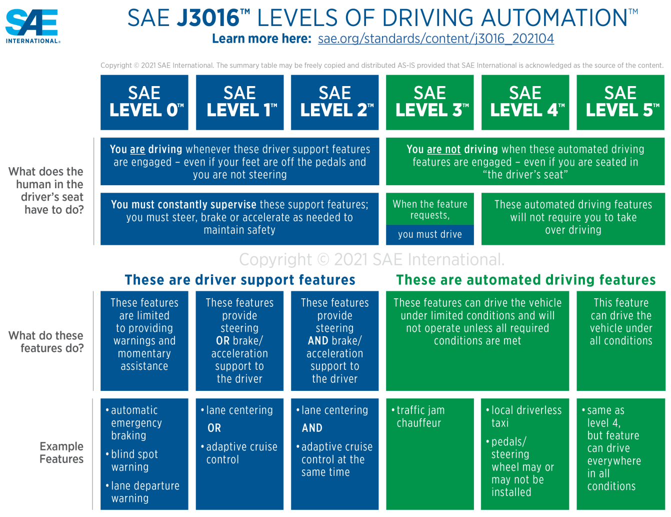
Chapter 1

Introduction

1.1 Simulation for autonomous driving

The concept of digital twin has garnered lots of attention with the advent of scalable cloud infrastructures and IoT technologies[1]. As its name suggests, such virtual system is designed with the capabilities of mutual improvement with its real world counterpart, by means of continuous bidirectional exchange of information.
In the context of autonomous vehicle systems, the self-driving architecture greatly benefits from the existence of its digital twin. As a matter of fact, all its components, from perception, to path planning and control, can be simulated and improved in virtual environments, although up to the level of realism offered by the employed simulator.
However, simulation is not just a tool to achieve marginal refinements of the self-driving architecture.
Safety and reliability of autonomous vehicles are of paramount importance, and simulation allows to provide guarantees also in regards to such aspects. As it has been quantitatively shown, by means of statistical analyses, in the **research report of RAND corporation**[2], alternative methodologies to on-road test drives are required to ensure such properties of the autonomous driving software: simulation is one of the possible solutions mentioned in their research. In their work, they provide proof that to demonstrate a given level of reliability of autonomous vehicles with respect to the 2013 reference human driving failure rate of 1.09 fatalities per 100 million driven miles, with a reliability threshold $R = 99.9999989\%$ and a statistical confidence level $C = 95\%$, 275 million failure-free miles would have to be driven: «If one considers a fleet of 100 autonomous vehicles test-driven 24 hours a day, 365 days a year at an average speed of 25 miles per hour, this would take about 12.5 years. [2]» In the case that the degree of precision (e.g., if we wish to estimate the failure rate to within 20%, $precision = 0.2$) with which such

failure rate of the autonomous vehicle is estimated, then the time required to run the vehicles for the needed mileage (which under this requirement increases to 8.8 billion miles) is of 400 years[2]. This is valid in the case that $precision = 0.2$.
Furthermore, similar time periods are required to also verify reliability guarantees related to the % improvement of the autonomous vehicle system with respect to the human-driving failure rates: in the same experimental setup, with the aforementioned autonomous fleet, 225 years of continuous driving would be required to verify a %20 improvement over the human driving performance.
Clearly, all of these estimates are a statement on the need of simulation to address all aspects related to the development, deployment and validation of autonomous vehicles.
All of the mentioned issues have to be properly addressed in order to reach the level 5 autonomy level defined in the **SAE Levels of Driving Automation**[3], shown in figure 1.1.
As the SAE level from 0 to 2 defined conditions under which a human driver is still in control of the vehicle, the aforementioned arguments become of increasing relevance starting from the SAE level 3 to 5, under which the vehicle drives itself in full autonomy. Excluding level 3, which categorizes all autonomous vehicles which can still require human intervention if certain conditions are met, level 4 and 5 are those of main interest for the development of fully autonomous vehicles.



SAE **J3016**™ LEVELS OF DRIVING AUTOMATION™
Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 1.1: SAE Levels of Driving Automation

Hence, from the statements shown in the RAND corporation report [2] and the requirements for level 4 and 5 autonomy provided by the SAE standard shown in the figure 1.1, it is possible to conclude that simulation plays a fundamental role

for the development entirely autonomous cars.
Indeed, ever since 2016, the year in which the RAND corporation report was published, the self driving sector has dedicated much attention to the development of autonomous driving simulation software.
At the outset, the first self driving simulator research attempts were those of DeepDrive[4] and OpenAI[5] which employed GTA 5 as simulation engine. Then, research moved on to open source solutions such as CARLA[6](Car Learning to Act) which provided a greater degree of customization of the simulation, from the car sensor setup to the dynamics of the environment.
Numerous other solutions have been simultaneously developed, such as AirSIM [7], GAZEBO for self-driving [8] before the industry showed its enterprise solutions.
NVIDIA has been among the pioneers of autonomous vehicle development with its NVIDIA DRIVE sim simulation platform in 2018 [9][10], and its recent follow-up with a synthetic data generation engine[11] NVIDIA replicator for the development of artificial intelligence algorithms for self-driving.
Although the latter provides a perspective biased towards research, indeed also companies directly involved in the automotive sector have acknowledge the fundamental role of simulation.
Tesla[12][13], NIO[14], waymo[15][16][17] and Uber[18] have all employed simulation as a core component of their self-driving technology stack, both for the development of deep neural networks used for scene understanding and also safety and reliability assessment.
Hence, as the autonomous driving software comprises a wide variety of tasks under the domains of perception, planning and control, the applicability of a digital twin capable, by definition[1], of simultaneous improvement of its real counterpart is of paramount importance for development of SAE level 5 autonomous cars.

1.2 Research setting and main motivations

From the broad set of perception tasks which are comprised by the autonomous driving architectures, spanning from lane detection, object detection, semantic segmentation, instance segmentation, multi object tracking and many more, this work focuses on the novel panoptic segmentation task, which entails the joint instance and semantic segmentation prediction of entities in the scene. The panoptic segmentation task, defined in 2019 in the paper[19]from the FAIR research group, presents the premises to achieve greater scene understanding capabilities compared to previous semantic and instance segmentation methods which are traditionally applied separately on images. Such premises are enforced by-design, as the task imposes the joint learning of instances, semantics and their mutual relationships

(e.g. a car is frequently seen close to roads, seldom surrounded by trees).
Many approaches in the literature have provided improvements over the original PanopticFPN[20] model: between the most relevant ones there are PanopticDeeplab[21], EfficientPS [22] and UPSnet [23]. Although enhanced segmentation capabilities are fundamental in the context of autonomous driving perception, these models have to be trained on datasets which are sampled from the target inference setting, thus requiring expensive labeling of real world data.
The simplest way to overcome such issues is to employ transfer learning, by starting from pretrained models which can then be fine-tuned on much smaller datasets, without much overfitting or loss of performance.
However such setting does not inherently allow the integration of simulation and synthetic data in perception models, which as has been described in the above section, are of paramount importance to achieve highly accurate and robust perception systems.
On the other hand domain adaptation and its related techniques provides the means to develop models that are capable to bridge the gap between synthetic and real data, while retaining the benefits of both worlds.
As such, this dissertation brings the panoptic segmentation architectures together with the modifications required to build domain adaptive models, with the objective to develop effective panoptic segmentation models for autonomous driving perception.
This line of work has been motivated both by the high impact that novel sim2real methods can have, given the proven necessity of simulation to build autonomous driving systems, as well as the lack of research on synthetic-to-real domain adaptation for the panoptic segmentation task.
As a matter of fact, only one publication [24] treated domain adaptation for the panoptic segmentation task. The latter considers self-training as a domain adaptation approach, leaving open the research on adversarial based domain adaptation.
Hence, the objective of this thesis is to improve panoptic segmentation models which are meant to perform inference unlabeled real world image data, by training them in the setting of unsupervised adversarial domain adaption, on labeled synthetic images,generated by means of simulation, simultaneously with unlabeled real data. The performance improvement attained by the use of domain adaptation is measured against a set of baseline models which are trained solely on synthetic images and then tested on real world images. The upper bound to the attainable improvement by domain adaptation is instead estimated by training and testing the same deep learning architecture with the actual ground truths of the real world data, which are never employed during domain adaptation.
In this way, it is possible to assess the degree by which the domain adaptive model could improve if it were to completely fill the domain gap which distinguishes the synthetic and the real image domains, thus achieving perfect predictions also on

the real world.

With the aim to build a realistic sim2real model training scenario, it has been necessary to develop the experimental setting under which large quantities of auto-annotated synthetic data are available and can be continuously updated, while real world data comes without annotations and in limited amounts. Thus the research that has been carried out consists of two consecutive components. The first is focused on the development of both the experimental scenario and the associated datasets, while the second is centered around the creation, training and evaluation of the domain adaptive panoptic segmentation model. Although synthetic datasets such as SYNTHIA [25] exist and can be matched with other real world ones such as cityscapes, the objective of this thesis is to demonstrate the real world applicability of domain adaptation to improve perception models, by means of continuous streams of large amounts of synthetic data generated in simulation, paired with unlabeled real world data captured from the cameras of an autonomous driving fleet. To this end, a synthetic data generation tools has been developed by means of the CARLA simulator. Then, the synthetic image dataset has been label-matched with the renowned cityscapes and bdd100k real world image datasets. In this experimental setting, the latter are representative of the unlabeled image data that would be captured a car's cameras in a real world scenario. Downstream to the label matched sim2real dataset, an adversarial domain adaptive modification of the PanopticFPN model has been trained on with ground truth supervision on synthetic data and self-supervision on real data, with BDD100k or cityscapes, and the remaining real dataset of the two has been used as an out of sample dataset to test the domain adapted model on completely out-of-sample data. The novelty of this research stems from the fact that there is little focus on domain adaptative models for the panoptic segmentation task. Most approaches to the synthetic-to-real domain adaptation problem in the autonomous driving setting are still concerned with semantic segmentation [26, 27, 28], instance segmentation[29, 30] or object detection [31, 32, 33]. Other lines of work entail domain adaptation for the LiDAR panoptic segmentation task [34]. Domain adaptation for semantic segmentation and object detection on point cloud data from LiDAR have also been addressed in the recent literature[35, 36, 37], but are out of the scope of this work, due to their use of point clouds instead of image data.

1.3 Thesis outline

With the outlined research context and experimental setting of the domain adaptive panoptic segmentation experiments, this dissertation presents the developed research in seven chapters. The first is the introduction, which motivates this work from the perspective of the scientific relevance, the need of the industry and the

implicit future relevance of synthetic-to-real related research in the field of deep learning, given the presented claims on the absolute need of simulation to achieve full self driving vehicles.
The dissertation then develops in the second chapter: related works. It sets the context of the state of the art and the novel research on autonomous driving perception. Much attention has been dedicated to the lines of work which are related to the presented experimental setting, with focus on the available autonomous driving datasets, the perception tasks, the building blocks of the deep learning architectures involved in this study and the literature of domain adaptation research.
Following the presentation of all the required notions that are employed in this work, chapter three describes the methodology that has been employed to create a multi domain dataset that would satisfy the needs of domain adaptation experiments, namely a source and a target domain on which to operate.
Chapter four is focused on the definition of: the components of the deep learning architecture that has been employed in this work, the implemented modifications needed to perform adversarial domain adaptation, the optimization objective and the model hyperparameters. Following the description of the methodology employed in this work, chapter five presents the outcome of the domain adaptation experiments that have been carried out, focusing on the improvement of the domain adaptive deep learning model with respect to a set of computed baselines.
Chapter six summarizes the results and the future research possibilities that are observed as a consequence of this work.

Chapter 2

Related works

2.1 Autonomous driving scene understanding

Scene understanding is a field of research which entails a broad set of computer vision related problems. As autonomous vehicles are generally equipped with a diverse set of sensors which comprise multi short, medium and long range cameras, LiDAR, radars and ultra sonic sensors, the research devoted to the optimal methods to extract information from their data and fuse together the provided perspectives, is indeed critical.**Object detection**, **semantic** and **instance segmentation** and **multi object tracking** are some of such tasks which have been found to be fundamental for extraction of semantic information from camera sensor setups employed in autonomous vehicles. Further specializations of such tasks have been employed as a means to achieve the same objectives also by means of LiDAR data, so as to extract a diverse yet robust representation of the semantic objects in the scene.
Such data is then processed and employed by downstream path planning and control algorithms which provide the vehicle with its required world navigation capabilities.
Obviously, such environment sensing capabilities are required to detect obstacles, determine navigable road area and avoidance of zones of risk based on other the position of other agents in the scene.
As driving entails a set of complex decisions and rules that must be respected, the related information has to also be extracted from sensor data: traffic light, traffic sign, intersections, lane lines are all concepts that are necessary for the decision making of the autonomous driving robot.
All of this information is thus employed to concretely define the available moves and the set of constraints to which these are subject to.
Clearly, it is critical for the algorithms to which such responsibilities are delegated

to be perfected to their greatest extent.
Some of the aforementioned tasks have been specialized for domain specific applications, such the lanenet[38] semantic segmentation model which provides accurate detection of lane lines as well as the available drivable area. Other methods such as Track-RCNN[39] aim at tracking moving object instances in the scene and predict their possible future trajectories[39].
The much simpler traffic light[40][41][42]/sign detection[43, 44] and recognition are both addressed by means of standard object detection models such as SSD (single shot multi-box detector) [45].
The latter tasks play only a part in the extraction of data necessary for the decision making of an autonomous vehicle. Intersection detection is a much more semantically complex problem that the perception system has to tackle. Some approaches employ LiDAR point cloud data to determine the presence of intersections[46]. Other methods instead leverage deep learning convolutional models to classify whole images as intersection/not intersection[47], or detect region of interest which contain the intersections[48] in the case of Nvidia WaitNET.
The complexity of such problem inspired also the creation of multi-modal datasets to aid the development of intersection detection algorithms which leverage multiple data sources as a means to robustly locate them.

Hence, scene understanding entails a large and diverse set of tasks which are necessary to estimate the state of the environment in which the autonomous vehicle is located.
In this work, the focus is directed to the extraction of semantic scene information from images. To address this issue, the literature provides a set of approaches formalized under the tasks of object detection, instance segmentation and semantic segmentation. The objective of the former two tasks is of determining the location in image coordinates of the objects of interest: vehicles, pedestrians, traffic lights, traffic signs and so on.
As such, these are the set of countable objects to detect which have to be treated each as a unique and separate instance from all others of the same category.
The literature categorizes all entities which fit such descriptions as **"things"**[49].
On the other hand, uncountable entities such as the sky, road, walkable areas, vegetation and buildings are defined as **"stuff"** [49]. More broadly, all that can be considered as background fits the latter description, although areas of specific interest can still fall under the same category: an example of this is the drivable road area, required to determine the available navigation space of a robot.
The latter problem is often tackled by means of semantic segmentation algorithms instead.
However, such technique is not strictly limited on its own to the segmentation of the background, as objects which fall under the semantic category of the "things" , such

as "vehicles" or "pedestrians", can still be segmented by a semantic segmentation model.
The key difference is that the latter is not capable by design to discern between unique instances of "things", which are generally more useful whenever each can be detected and treated as a separate entity than as a broader category.
Still both "things" and "stuff" are simultaneously present in any image frame, hence their relationships can still provide a richer understanding needed for their detection or segmentation, rather than treating each problem with separate models.
To this end, the **panoptic segmentation**[20] task has been developed.
The study of the two super-categories "things" and "stuff" is encouraged under a framework which requires them to be jointly modeled.
As this task presents the potential to outperform current methods employed in the context of autonomous vehicle perception, this work elaborates on such problem by extending existing models through the use of simulation and domain adaptation techniques as a means to continue novel and still unexplored research on this topic.

2.1.1 Object detection

Given a label set $Y = \{0,1,2,...,N-1\}$ the objective of the object detection framework is to localize in an image, all instances of such objects and possibly assign to each of them a confidence score. This task focuses solely on the search of the entities which are defined as countable objects (or "things")[49] in the literature. Each category can be represented by multiple detections in the same image, and each of the detected instances in enclosed by a bounding box generally described by the coordinate of its top-left corner and its width and height. Different conventions employ the top-left and bottom right points as a means to geometrically describe the bounding box.
Although relatively simple, these specifications lay the foundations for the more complex instance segmentation task, which poses the additional requirement to generate fine-grained pixel-wise masks that cover the detected instance, rather than the coarser bounding box.

2.1.2 Semantic segmentation

Before the instance segmentation task is presented, it is necessary to introduce the semantic segmentation task.
Let the label set $Y = \{0,1,2,...,N-1\}$ of cardinality N, be the set of categories of the entities of interest. Given an input image, and the mentioned label set, the semantic segmentation task requires a provided algorithm to assign to each pixel of the input, one of the possible categories, eventually with pixel wise prediction

confidence scores.
This task is not constrained to the search of either "things" or "stuff"[49]. Nevertheless, the fact that all pixels associated with a given category are predicted with the same label, poses this task at advantage for the search of the uncountable entities (or "stuff"), while categorization of all "things" loses some information as the concept of identification of unique and separate instances of the same semantic class is not required.
This framework provides greater degrees of freedom in terms of the kind of identifiable entities, as it is not anymore constrained to the detection based on rectangular regions. Of course, in such context the scope of the task broadens from the search of single object entities to the understanding of the scene in consideration. Detection of unique object instances is not anymore a specification, as the task focuses on the whole scene understanding, rather than the search of objects of interest.

2.1.3 Instance segmentation

Instance segmentation combines the concepts of object detection and semantic segmentation to tackle in a more fine-grained manner the problem of identification of object instances in images.
This task differs from object detection solely due to the fact that object instances have to now be labeled pixel-wise with their instance ids, so as to generate an instance mask which represents the object.

2.1.4 Panoptic segmentation

The panoptic segmentation task combines the positive aspects of instance and semantic segmentation under the same problem formulation. As instance segmentation provides a framework for the segmentation of object instances and assignment of unique ids to each, besides their broader category label, such id assignment loses its meaning if utilized to segment road area or vegetation and the sky: multiple instances of the sky do not serve a purpose and are misleading for the model which predicts them as separate instances. Hence, these gaps can be filled by employing the problem formulation of the semantic segmentation task: under such context, instances are not relevant, but the segmentation and recognition in the image are the focus.
Hence, the panoptic segmentation framework leverages these features to define a new task to encourage the joint study of the relevant instances ("things") and the uncountable entities ("stuff")[20].
As such, the task is formulated as follows.
Given a set of predefined semantic categories encoded by the set $Y = \{0,1,2,...,N-1\}$ of cardinality N, the task requires a panoptic segmentation algorithm to predict

for each pixel i of an image, a pair $(y_i, z_i) \in Y \times \mathbb{I}$, where y_i represents the semantic class of pixel i and z_i represents its instance id. I is the set of assignable instance ids.
The z_i's group pixels of the same class into distinct segments. Ground truth annotations are encoded identically. A special ground truth label can be assigned to ambiguous or out-of-class pixels, both for the ground truths and the predictions: no penalty is assigned to a model which wrongly predicts the class of a pixel to which such void label has been assigned.
As previously implied, if the entire label set is Y, Y^{stuff} is the set of label of stuff classes, Y^{things} is the set of thing classes, the panoptic task requires that $Y = Y^{stuff} \cup Y^{things}$ and $Y^{stuff} \cap Y^{things} = \emptyset$.
For each image pixel, a class y_i and an instance id z_i are assigned: the latter is considered only in the case that the class prediction belongs to Y^{things}, else it is ignored.
This implies that under such framework, all pixels of stuff classes belong to the same instance, while all pixels which belong to a unique instance must be represented by the same tuple (y_i, z_i).
One final remark is that, as mentioned in [20], the selection of **things** and **stuff** classes is a design choice left to the panoptic dataset creators.

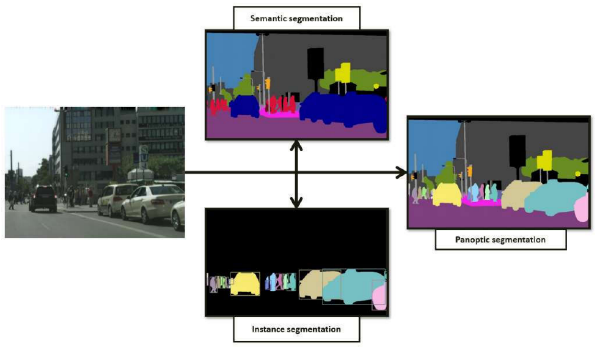


Figure 2.1: This figure illustrates the relationships between semantic segmentation, instance segmentation(and the simpler object detection), with the panoptic segmentation objective [50]

2.2 Scene understanding datasets

The scene understanding problem entails a wide variety of tasks .
In the literature, the MS-COCO object detection dataset[51] has been a major point of reference for computer vision researchers since its first iteration in 2014 as an instance segmentation dataset. After imagenet[52], it represents one of the most

notable datasets in terms of labeling effort scale due to its 328 thousand images and 2.5 million annotated instances. Developed to further research on object detection and instance segmentation, it provides 91 "thing" categories for which instances have been labeled following the COCO annotation convention in JSON format [53]. Afterwards, it has been extended to also address the semantic segmentation of "stuff" categories. The distinction of "things" and "stuff" is not a novelty of the COCO dataset, but a renowned concept from the computer vision literature[54]. Objects which can be identified by a closed shape, without considering possible occlusions, and are countable, are defined as "things". On the other hand, uncountable entities, whose extent is not reducible to a simple enclosing shape, belong to the "stuff" category (e.g. the sky or the grass cannot be identified by a characterizing shape as their spatial extent varies based on the visible area observed in a given image scene).
The distinction between the two categories has been kept also in the way research and the related datasets have been treated in the literature. As such, while the traditional COCO dataset entails object and instance segmentation of "things", the later developed COCO-stuff focuses only on stuff, treating the thing instances without their unique identity and collapsing each under its representative semantic category, alongside other classes for stuff entities.

2.2.1 Synthetic autonomous driving datasets

In the context of autonomous driving, a large and diverse set of tasks have been developed for the purpose of holistic scene understanding. From semantic, to instance segmentation, multi object tracking, trajectory prediction and object detection, each requires specific ground truth annotations in order to be properly studied.
As real world data is often scarce or of limited availability, the use of large synthetic datasets as a means to support smaller real world datasets has become a notable approach in the literature. Hence with the premise to aid the development of self-driving vehicles, preliminary works on the synthetic data generation by means of games have been proposed in the literature. Notably, the GTA5 dataset[55] has been developed in 2016 to aid the study of object detection as well as semantic segmentation in the context of autonomous driving. It provides 25 thousand images, of size 1914x1052, each with its pixel-wise semantic segmentation labels that span over 19 classes. Simultaneously to the GTA5 dataset, SYNTHIA[25] had been developed. The latter has become widely known in the literature, as it entails two hundred thousand 960x720 images with pixel-wise semantic segmentation labels and depth information. It spans scenes from different seasons, weather and illumination conditions, in order to provide great scene diversity. Furthermore it includes 13

classes: sky, building, road, sidewalk, fence, vegetation, lane-marking, pole, car, traffic signs, pedestrians, cyclists and miscellaneous. Follow up work on SYNTHIA, namely SYNTHIA-Cityscapes, provides a Cityscapes label matched version of the former, so as to perform synthetic-to-real studies. SYNTHIA-SF[56], developed in 2017, integrates depth information and semantic segmentation labels with instance segmentation ground truths, while maintaining label compatibility with Cityscapes. The latest version, SYNTHIA-AL instead solely focuses on the study of things, as it provides only instance segmentation, 2D and 3D bounding boxes.
The study of synthetic datasets continued withe the increase in popularity of the CARLA simulator[6], which by-design eases the generation of labeled synthetic autonomous driving data, while providing a greater degree of realism when compared to the previous alternatives such as any of the SYNTHIA variations [25] or GTA5 [55].
Hence, recent works employed CARLA to generate broad and diverse autonomous driving centric datasets. Of these, the IDDA dataset[57] is notable as it provides over one million images captured under different weather, environmental conditions, city and rural areas scenarios and multiple camera views.
One major issue of such dataset is the lack of instance annotations, which are required to study things, both from the perspective of object detection and instance segmentation.
An alternative synthetic dataset generated by means of the CARLA simulator, namely KITTI-CARLA[58], provides a KITTI-label[59] matched dataset with similar sensor setup, LiDAR point clouds and object detection bounding boxes. This however lacks semantic segmentation annotations.

2.2.2 Real world autonomous driving datasets

Autonomous driving scene understanding has been studied from the perspective of a diverse set of tasks. To support such studies, many real world datasets have been developed to support the research. Of these, one of the most notable in the literature is the KITTI dataset, created in 2013 by Andreas Geiger et al.[59].
As it is a diverse driving scene dataset, which comprises synchronized multi-sensor data of images from multiple cameras, LiDAR point clouds, IMU readings, 2D and 3D bounding box annotations, it has been one of the first to support research on driving scene perception.
Such pioneering work on driving dataset generation spurred much of the following research on development on the topic, with the objective to increase the richness and diversity of such data collections.
At the same time, there has been an increase in complexity of the researched scene understanding tasks, which evolved from object detection to the more complex pixel-wise instance annotation and semantic segmentation. Hence, the research

needs to expand upon such topics stimulated the development of novel datasets that would support such novel research challenges.
The renowned Cityscapes instance, semantic segmentation and depth estimation dataset[60], published in 2016 has ever since been a point of reference for novel research on scene understanding. In 2020 it has also been extended so as to provide panoptic segmentation ground truths. Scenes were captured in 50 different cities in Germany, during different seasons and months of the year. However no diversity in terms of weather conditions had been included, as adverse weather conditions have been explicitly avoided.
It provides two subsets, a smaller split with fine-level annotations, and a larger one with coarse labels.
The training-validation-test split for the former consist of 2975, 500 and 1525 images respectively.
Each is built by uniformly sampling images of cities from small to large scale, keeping a whole city under the same split.
The coarse Cityscapes dataset of 20 thousand images, is instead provided as a whole, with the purpose of serving as additional training data for the Cityscapes fine-annotated split.
In total there are 30 classes for both, but their benchmark evaluation set is reduced to 19 due to the rarity of the removed classes. To further increase the scale of driving datasets, Mapillary vistas[61] had been developed. It entails 25 thousand images, of which 18 thousand are for training, 2 thousand for validation and 5 thousand for testing.
Mapillary includes 66 semantic classes, of which 37 are dedicated to object instances, thus increasing the diversity of the recognizable scene entities with respect to Cityscapes.
As such, semantic and instance segmentation and object detection are considered as the main research tasks supported by this dataset. Its further extension Mapillary Vistas 2.0, developed in early 2021, integrated its previous version with the support to the novel panoptic segmentation task.
Although Cityscapes and Mapillary were considered large scale datasets at the time of their creation, there has been a push to research and develop even larger and diverse datasets, adapted to address more diverse multi-task learning contexts.
Hence, not only with the purpose to address instance and semantic segmentation, but also other tasks such as lane detection, drivable area segmentation, 2D and 3D object tracking and depth estimation. Thus, to fill the gaps in the literature related to the lack of datasets which resolve all the aforementioned issues, the BDD100k[62] dataset has been developed in 2018. Multiple splits of the source video sequences have been created, in order to satisfy different tasks. The largest split consists of 100 thousand images labeled for object detection (10 provided object classes), lane marking(9 lane type classes) and drivable area (three categories including

background) segmentation, divided into train, validation and test subsets of 70,10 and 20 thousand images respectively.
A smaller split of 10 thousand images provides more fine-grained annotations for semantic (19 semantic classes), instance (8 instance classes) and panoptic segmentation (40 overall categories), divided into 7 thousand images for training, 1 thousand for validation and 2 thousand for the test set.
In the case of panoptic segmentation, the 40 semantic classes are subdivided into two subsets dedicated respectively to things and stuff groups. Regarding thing classes, 9 instance categories are provided, while for stuff there are 31 semantic entities.
As an improvement over the previous datasets, the scenes have been captured under a diverse set of weather and illumination conditions based on time of the day, considering also diverse scene types: from cities, isolated streets, residential areas, and highways.
As such datasets have become established points of reference for image scene understanding research, the follow-up publications focused on the instance, semantic, panoptic segmentation and motion forecasting from the perspective of both LiDAR and images.
Among these, there is the Argoverse dataset [63] for 3D tracking and motion forecasting. Similarly, NuScenes[64] provides annotations for panoptic segmentation through LiDAR point clouds, as well as 2D and 3D object detection ground truths. On the same line of work the Waymo open dataset provides two versions, namely the motion and perception subsets, whose main objectives are object tracking and trajectory forecasting for the former, while the latter treats image and LiDAR data for object detection related tasks.

2.3 Deep learning perception

As the panoptic segmentation task has garnered the attention of the research community, a variety of models have been developed to tackle the problem.
In this work, the PanopticFPN[65] model has been employed. It has been published simultaneously to the panoptic task[20], both by the FAIR group (Facebook AI Research), with the objective to develop a powerful and still efficient baseline model, in terms of time complexity of its inference computations.
From the subsequent development to the task publication in 2019, the literature provides two main ways to develop panoptic segmentation models: convolutional neural network based or transformer architecture based.
Although this work focuses on the former of the two, it is worth mentioning the most recent alternative state of the art approaches approaches.

The convolutional models follow a widely used approach in the literature to the design of convolutional neural network architectures: a backbone component is dedicated to the extraction of task specific semantic features from images, then its results are employed by one or more downstream convolutional networks to which correspond one or more downstream tasks e.g. instance and semantic segmentation. Such models can be further subdivided in: models which have as a by-design objective the instance segmentation/ semantic segmentation, and are complemented by an auxiliary semantic/instance neural network branch, respectively. In both cases, the backbone is shared, implying that its computed convolutional features must be sufficient and general enough to address the downstream tasks.
The same holds also for a recent(mid 2021) set of methods which address both tasks at once, through a unified deep learning architecture: it is the case of the SPINnet [66] and Panoptic FCN [67] The former is the case of the EfficientPS[22] and PanopticFPN[65], which extend the architecture of the Mask R-CNN [68], an instance segmentation architecture, with an auxiliary semantic segmentation branch. The other is instead the context of the PanopticDeeplab[21] and its extended SWideRNets[69] architectures. The Deeplab model, which these newer architecures extend, was designed for semantic segmentation and improved to also tackle the panoptic task.

Instead, the set of transformer-based[70] models employ convolutional networks only as a means to generate features which are then utilized by vision-transformer[71, 72] architectures to perform the panoptic segmentation task, such as in the Panoptic SegFormer[73].

In order to more easily perform an assessment of the impact of domain adaptation on the panoptic segmentation task, the **PanopticFPN has been selected as the baseline architecture to which the required modifications have been applied**. This has been done also to avoid any possible complex interactions between the domain adaptation module and any of the modifications employed in any of the mentioned lines of research. In order to present the components of the developed modification of the PanopticFPN architecture, this section provides a review of the state of the art of the literature regarding the main building blocks that constitute domain adaptive Panoptic FPN developed in this thesis.

2.3.1 Convolutional neural network structure

At their core, convolutional neural networks (abbreviated to ConvNets) can be represented by **computation graphs** 2.2: directed graphs whose nodes represent **arbitrary operations** applied to their inputs, represented by edges pointing a given node, while its outputs are described by edges pointing outwards. Therefore,

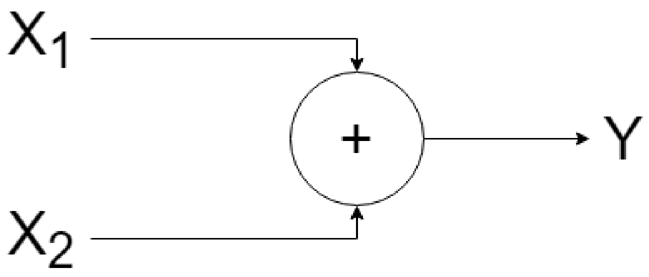


Figure 2.2: Example of a computation graph

each node represents an operation to be applied in the broader context of the network: computation nodes are generally referred to as **layers** in the artificial intelligence literature.
Hence in the execution of a computation graph from inputs to outputs, namely a **forward pass** through the neural network, the neural architecture amounts to the application of a composite function

$$\mathcal{F}(X) = \mathcal{F}_n \circ \mathcal{F}_{n-1} \circ ... \circ \mathcal{F}_1(X)$$

applied to its input X.
Each of such $\mathcal{F}_i$ represents a layer, which can have **0 or more parameters** W_i objective of an optimization routine.
These are optimized with the aim to approximate, by means of the network, an unknown objective function $Y(X)$, whose samples y_i are the ground truths associated with each sample $x_i \in X$ in the dataset X.
Such optimization is guided by a loss function $\mathcal{L}$, which based on the learning objective e.g. classification, detection segmentation, measures the error between the approximate mapping, from inputs X to ground truths Y, $\mathcal{F}(X, W_n, W_{n-1}, ..., W_1)$ represented by the network and the ground truth samples from the function Y

$$\mathcal{L}(\mathcal{F}, \mathcal{Y})$$

Such mathematical formalization is fundamental both to define neural networks with modularized structures of the layers and to efficiently optimize parameters by means of the backpropagation algorithm [74].
As this structure lends itself to the implementation of novel operation to add to neural networks, much research has been devoted to the definition of custom operations that would yield better models as a result of their use.
Therefore the following subsections will present with a bottom up perspective, from the basic operations to the more complex ones at the neural network level, the defining components of the current state of the art deep learning models.

2.3.2 Basic mathematical operations

Each layer of a neural network entails a specific **operation** and a set of **operation-dependent parameters** W, that from a general point of view, have to be optimized as a means to render such operation meaningful in the context of the learning problem at hand. This is because such parameters are initialized either by means of arbitrary but empirically effective sub-optimal starting conditions such as the Xavier[75] or He[76][77] initialization or from data-dependent already optimized parameters [78] .
From this general perspective, layers of a neural network are stacked one after the other so as to compose complex non-linear functions capable of generalizing and learning from the most complex datasets and in a broad variety of problem settings: classification, object detection, semantic segmentation, instance segmentation, image and audio generation and so on. Given, the context of this work, for the following subsection it is necessary to define the shape of the input data.
An image, or more in general a feature map, is represented by **tensor of shape** $C \times H \times W$, in which H and W are the height and width of the image, while C stands for the number of channels (or modalities of the input data) e.g. $C = 3$ for RGB images.

Linear

Linear projection by means of a tensor of weights $W \in \mathbb{R}^{N_{in} \times N_{out}}$, with $N_{in}$and N_{out} representing the shape of the input and the desired output shape respectively, is the simplest operation that can be applied to a set of input samples.
In the case of images, such operation quickly becomes computationally expensive as it entails the use of flattened versions of image tensors. As such weight tensors are rarely sparse, its memory representation poses a major resource constraint. This sets a major bottleneck of the operation on flattened tensor.
Hence, Linear layers are generally used downstream to convolutional neural networks, solely to perform final operations after the dimensionality of the data is already greatly reduced.

Convolution

The convolution operator is the core component of convolutional neural networks. Much of its relevance in such models is due to its capability to exploit **spatial correlation** and **locality of entities** represented in digital signals such as audio and images, in order to extract meaningful patterns from the data.
Convolution is a **dyadic or binary** operator, denoted with $*$ which when applied

to its two inputs $f$and g produces one resulting output $f * g$.
Although convolution is generalized to be applied to continuous and discrete functions, the latter is the version which is relevant in computer vision applications. In such field of application, discrete convolution can operate on N-Dimensional inputs, although the most common ones for computer vision tasks are the 2-D and 3-D discrete convolution operators.
In such case of 2D convolution, the aforementioned operands f and g are defined in the most general 2D setting :

- f is represented by $n\_channels$ **input feature maps**, formally is a tensor

  $$f \in \mathbb{R}^{n\_channels \times H \times W}$$

  with H and W its height and width (e.g. an RGB image is a tensor of shape $3 \times H \times W$, as it has the R, G, and B planes, each with $H \times W$ pixels)

- g is a collection of **kernels** (or **filters**), which is **convolved with the input feature maps** f, and is a tensor

  $$g \in \mathbb{R}^{n\_out\_channels \times n\_in\_channels \times H_{ker} \times W_{ker}}$$

  . W_{ker}, H_{ker} are height and width of the kernel, with $H_{ker} \leq H$ and $W_{ker} \leq W$.
  THe dimension$n_in_channels$ is determined by the number of channels $n\_channels$ of the input feature maps with which g is convolved, such that $n\_in\_channels = n\_channels$.
  The remaining dimension $n\_out\_channels$ defines the **desired number of channels of the output feature maps** o resulting from the convolution

  $$o = f * g$$

  between input feature maps$f$ and the collection of kernels g.

Given these definitions, the 2D convolution algorithm can be described.
Firstly, the steps of the simplest setting of 2D convolution are described, to then describe the way by which that same procedure is applied considering the aforementioned general setting.
The 2D convolution of a single input feature map $1 \times H \times W$ with a kernel$H_{ker} \times W_{ker}$, requires the latter to slide over all **valid locations** of the input. Such valid locations determine the shape of the output feature map.
The kernel is centered at a valid location, and the locations of the input feature map that are within the kernel receptive field of size $H_{ker} \times W_{ker}$ are multiplied by the values of the kernel, to then be summed to produce the aggregate value associated with the corresponding location in the output feature map.
This is repeated for all valid locations of the input feature map.

Extension of this operation to the general 2D convolution case in which the input has $n\_channels$ feature maps and the number of desired output feature maps is $n\_out\_channels$, requires few additional steps to the algorithm.
For an $n\_channels \times H \times W$ input feature map, $n\_channels$ distinct kernels of shape $H_{ker} \times W_{ker}$ are each separately convolved as described above, to produce $n\_channels$ preliminary output feature maps, which are element-wise summed together to produce one final output feature map. Thus, to generate the desired $n\_out\_channels$ output feature maps, the previous step is repeated $n\_out\_channels$ times with $n\_out\_channels$ different collections of such kernels. This procedure is illustrated in 2.3, in which an input feature map of shape $2 \times 5 \times 5$ is convolved with a $3 \times 2 \times 3 \times 3$dimensional collection of filters (dimensions follow the convention $n\_out\_channels \times n\_channels \times H_{ker} \times W_{ker}$) so as to obtain an output feature map of shape $3 \times 3 \times 3$ (convention used to represent the dimensions of the output feature map is the same as that of the input feature maps).

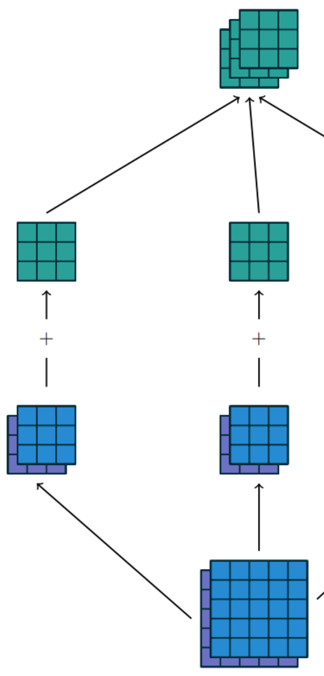


Figure 2.3: Illustration of the 2D convolution operator, from the paper [79]

As previously mentioned, only the valid locations are considered in the computation of the convolution.
Such spatial locations are all valid zero-indexed H_c, W_c kernel center cell coordinates of the input feature maps planes $H \times W$, and all their associated channels if valid, that allow the entire kernel receptive field to lie within the input feature map

planes. Such conditions can be expressed for kernels whose H_{ker}, W_{ker} are both odd(in practice even sized kernels are not used due to the pixel shift problem), as the joint fulfillment of the following conditions:

$$\begin{aligned}
1)0 &\leq H_c - (\lfloor H_{ker}/2 \rfloor - 1)\\
2)H_c + (\lfloor H_{ker}/2 \rfloor - 1) &\leq H\\
3)0 &\leq W_c - (\lfloor W_{ker}/2 \rfloor - 1)\\
4)W_c + (\lfloor W_{ker}/2 \rfloor - 1) &\leq W\\
5)0 \leq H_c &\leq H\\
6)0 \leq W_c &\leq W
\end{aligned}$$

Here, $(\lfloor W_{ker}/2 \rfloor - 1)$ and $(\lfloor W_{ker}/2 \rfloor - 1$ represent the extent of the kernel excluding the center location. As such, conditions 1 and 3 are lower bounds to the validity of the extent of the receptive field of a kernel given its center location H_c, W_c. Conditions 2 and 4 are instead upper bounds to the validity of the extent of the receptive field of a kernel given its center location H_c, W_c. The two remaining bounds are related to the possible locations of the kernel centers.
Each of such conditions is appropriately modified by the following set of **convolution operator parameters**, which define the way in which the **computation is carried out**, as well as the **shape of the output feature maps** of the operation

- stride: determines how kernels are slid over the input feature maps. Standard convolution sets stride to 1 so that the center of the kernel is moved by one unit, either in width or height, while it is slid. A stride ≥ 1 implies that the convolution skips possible center locations, by moving the kernel center of stride units, with respect to its preceding cell location. Thus, it provides a means to downsample the input feature maps by computing the convolution between inputs and the kernel in a coarse manner.
  It can also be interpreted as the distance between two consecutive kernel center locations [79]

- padding: the number of zero valued cells that are added to the extent of the input feature maps planes $H \times W$. This effectively increases the width and height of the input feature maps by adding a number of cells equal to the padding parameter both before the first and after the last cell. Thus, H becomes $H + 2 * padding$ and W becomes $W + 2 * padding$. It is generally set as $padding = 0$, a condition known as valid padding.

- dilation rate: to increase the receptive field of the kernels by employing the atrous convolution algorithm[80]. Dilation achieves this objective by inserting $dilation\_rate - 1$ zeros between each element of the kernel, thus effectively enlarging its size without requiring larger kernels. Thus, standard convolution sets the dilation rate to 1.

The use of any combination of these parameters affects the output feature maps shape as follows:

$$H_{out} = \lfloor \frac{H_{in} + 2 * padding_H - dilation_H \cdot (H_{ker} - 1) - 1}{stride_H} + 1 \rfloor$$
$$W_{out} = \lfloor \frac{W_{in} + 2 * padding_W - dilation_W \cdot (W_{ker} - 1) - 1}{stride_W} + 1 \rfloor$$

Regularization

Overfitting is a well known issue of deep neural networks. In order to limit its impact, a variety of approaches have been proposed in the literature. Of these the most cited and effective works include Dropout regularization layers[81] and batch normalization [82]. Such methods are essential for training neural networks that can generalize to test data, as they are prone to overfitting due to their large number of parameters.
Furthermore, these methods provide solutions to co-adapation and covariate shift problems that affect neural network models.

Co-adapation is a phenomenon that can affect multiple neurons of the neural networks(parameters of the kernels in the case of convolutional networks), causing them to learn highly correlated representations that are only meaningful jointly with other neurons(which are the parameters of the same or of one or more distinct kernels). Such behavior is undesired, as it wastes representational power. Ideally, each of the network parameters should represent independent and fundamental patterns extracted from the data.

As neural networks are in essence composite functions, each of which is represented by layers and its zero or more learnable parameters, the inputs of each layer can either be the data, in the case of the first layer, or a representation dependent on the learnable parameters of its preceding layers in all other cases.
Although the distribution of the input data does not change, the change in the learnable parameters as a result of their optimization during training of the neural network causes a shift in the distribution of the inputs to the intermediate network layers, which becomes increasingly more noticeable in the layers at greater neural network depth.
This causes instabilities during training as the reference of the optimization routine continuously changes, resulting in failure or too slow convergence to an optimum of the parameters[82].

Dropout regularization is a technique that prevents overfitting and co-adaptation

by dropping, with probability p, the neurons of the network at training time. Turning off a neuron results in changing the interconnection between the neurons of the neural network layers, thus enforcing each of the remaining units to not rely on the values learned by other neighboring neurons.
At inference time instead, all neurons are employed but the weights associated with outward connections are multiplied by the selected probability of dropping units, in order to take into account for the reduced number of units employed during training.

The **batch normalization** algorithm limits the impact of covariate shift on model training and regularizes the learned model[82].
By defining the computation at an arbitrary layer of a neural network as $z = g(W \cdot u + b)$, in which W are the weights,b the biases, u the layer inputs and $g(\cdot)$ a non-linear function, such as the sigmoid and ReLU, the batch normalization operator $BN(\cdot)$ is applied to the result of $x = W \cdot u + b$, before the non-linear function is applied.
It is necessary to note that such representation is valid also for convolutional neural networks, as convolution is implemented as matrix multiplication by the im2col algorithm.
The batch normalization operator $BN(\cdot)$ applied to each sample x_i of a batch $\mathcal{B}$ of $|B|$ samples with $i \in \{1, ..., |\mathcal{B}|\}$, each of shape $n\_channels \times H \times W$, separately to each feature map $k \in \{1, ..., n\_channels\}$ of x_i as follows:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_{\mathcal{B}}^{(k)}}{\sqrt{\sigma_{\mathcal{B}}^{2}{}^{(k)} + \epsilon}}$$

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

Here, the estimators $\mu_{\mathcal{B}}^{(k)}$ and $\sigma_{\mathcal{B}}^{2}{}^{(k)}$ are computed so as to batch-normalize each pixel of a given input feature map, separately for each of the $n\_channels$ feature maps. Hence, as each channel is considered separately, the mini-batch size is of $|\mathcal{B}|$ samples each of size $H \times W$.
Thus the k-th channel estimates are computed with the following estimators:

- $$\mu_{\mathcal{B}}^{(k)} = \frac{1}{|\mathcal{B}| \cdot H \times W} \sum_{i=1}^{|\mathcal{B}| \cdot H \times W} x_i^{(k)}$$

  is the batch estimator of the mean of the distribution of the inputs

- $$\sigma_{\mathcal{B}}^{2}{}^{(k)} = \frac{1}{|\mathcal{B}| \cdot H \times W} \sum_{i=1}^{|\mathcal{B}| \cdot H \times W} (x_i^{(k)} - \mathcal{B}^{(k)})^2$$

  is the biased estimator of their variance.

The per-channel parameters $\gamma^{(k)}$, $\beta^{(k)}$ with $k \in \{1, ..., n\_channels\}$ are learned during training.

During inference, the behavior of a batch normalized network is changed as described below.
The population estimates of the k-th channel $E\{x^{(k)}\}$ and $Var\{x^{(k)}\}$ are computed from a set of accumulated training batches $\mathcal{B}_j$ $j \in \{1, ..., N_{batches}\}$, each of size $|\mathcal{B}|$,each with their $\mu_{\mathcal{B}}^{(k)}$, ${\sigma^2_{\mathcal{B}_|}}^{(k)}$:

$$E\{x^{(k)}\} = E_{\mathcal{B}}\{\mu_{\mathcal{B}}^{(k)}\}$$

$$Var\{x^{(k)}\} = E_{\mathcal{B}}\{{\sigma^2}_{\mathcal{B}}^{(k)}\}$$

Then, the per-channel population statistics of the inputs $x^{(k)}$ are substituted to the $\mu_{\mathcal{B}}^{(k)}$, ${\sigma^2_{\mathcal{B}}}^{(k)}$ in the computation of the normalized input $\hat{x}^{(k)}$ at inference time as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E\{x^{(k)}\}}{\sqrt{Var\{x^{(k)}\} + \epsilon}}$$

This way, $y_i^{(k)}$ is the normalized feature map according to the statistics computed at the given layer.

Pooling

The objective of the pooling operator is to apply a non-linear transformation that generally downsamples its inputs, as a means to obtain in its outputs invariance with respect to small translations of such inputs[79].
In the most general setting, the pooling operator acts similarly to convolution: a window of size $H_{pool} \times W_{pool}$ is slid over the input feature maps and each value within its receptive field is aggregated to produce a single output value, according to a selected transformation. The most common aggregate operations are max and average pooling.
The output feature maps resulting from the application of pooling are dependent on:

- padding: the number of zero cells to add to both borders of the input feature maps. To further clarify, setting the horizontal and vertical padding $padding_H = padding_W = 2$, implies that the input feature maps becomes of shape $(H + 2 * padding_H) \times (W + 2 * padding_W)$, with all added cells set to zero.

- stride: distance in number of cells, with respect to height or width without considering diagonal sliding, between the application of two pooling operations. Vertical and horizontal strides are generally set to H_{pool} and W_{pool} respectively, such that pooling is applied to non-overlapping patches of the input feature maps.

The output feature maps that result from the use of a pooling operation have the following

$$H_{out} = \lfloor \frac{H_{in} + 2 * padding_H - 1}{stride_H} + 1 \rfloor$$
$$W_{out} = \lfloor \frac{W_{in} + 2 * padding_W - 1}{stride_W} + 1 \rfloor$$

Pooling is applied separately to each input feature map, hence it has no interaction on the number of input and output channels, thus $n\_out\_channels = n\_in\_channels$.

Activation functions

Activation functions are a fundamental component of neural networks that allows them to learn complex non-linear patterns from the data. These are applied after some other operation, such as projection by a weight matrix W or convolution with a collection of kernels, has been applied to the input data.
The most common ones are:

- ReLU (rectified linear unit)[83]

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

- LeakyReLU

$$LeakyReLU(x) = max(0, x) - 0.01min(0, x)$$

- hyperbolic tangent(tanh)

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

- sigmoid

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

Each is applied element-wise to its inputs.
While sigmoid and hyperbolic tangent are part of the past literature, ReLU[83] and LeakyReLU are the most relevant and largely used activation functions for the intermediate layers of neural networks. These have been developed as a means to keep the positive aspect of providing non-linearity of sigmoid and hyperbolic tangent, without their downsides such as input saturation and computational complexity.
Other activations such as softmax are generally employed after the final layer of neural networks as a means to normalize their outputs in the range [0,1], which allows for their interpretation as class membership probabilities. With an output of vector $v \in \mathbb{R}^K$, the application of softmax produces:

$$softmax(v_i) = \frac{\exp^{v_i}}{\sum_{j=1}^{K} \exp^{v_j}}$$

Transposed convolution

Transposed convolution is the inverse operation to that of convolution. It is not a proper inverse transformation, as it only allows to recover the original shape of the output of a convolution operation. Hence, it effectively serves the purpose of upsampling its inputs based on a set learnable kernel parameters.
Without any of such parameters, the operation becomes a simple upsampling operation by means of traditional nearest or alternatively bilinear upsampling algorithms.
As is the case for convolution, the computation of its transposed version can be influenced by a set of optimizable parameters:

- stride: stride of the associated convolution operation.
- padding: parameter to set in order to compute the required **input padding** parameter, also according to the chosen dilation rate.
- input padding: computed as $in\_padding_H = dilation_H * (H_{ker} - 1) - padding_H$ or $in\_padding_W = dilation_W * (W_{ker} - 1) - padding_W$, defines the number of zero cells to add to each border H and W respectively, of the input feature maps.
- output padding: additional zero-padding to add to each border of the width and height dimensions of the input, separately. It is generally set to zero
- dilation: distance added between between each element of the transposed convolution kernels

The output shape of the tensor resulting from the transposed convolution computation is:

$$\begin{aligned} H_{out} &= (H_{in} - 1) \times stride_H - 2 * in\_padding_H + \\ &\quad dilation_H \times (H_{ker} - 1) + out\_padding_H + 1 \\ W_{out} &= (W_{in} - 1) \times stride_W - 2 * in\_padding_W + \\ &\quad dilation_W \times (W_{ker} - 1) + out\_padding_W + 1 \end{aligned}$$

2.3.3 Backbone architectures

Deep learning models have become a staple technique in the computer vision literature. Ever since the Alexnet deep architecture[84] has outperformed traditional computer vision methods in the Imagenet recognition challenge[52] in 2012, deep learning models have become a staple technique of the computer vision research. Nevertheless, pioneering works and the early proofs of the potential of neural networks date back to the 1986 with the invention of the backpropagation [74] algorithm that allows the deep networks to learn. At the time, the

This subsection provides a literature review on the deep learning architectures that are employed as feature extractors for a broad variety of learning problems. Since the pioneer work of Hinton in 1986 and LeCun in 1998 about backpropagation[74] and handwritten digit recognition convolutional networks[85] respectively, the main design philosophy to the development of neural networks has been to implement them following a layer-like structure. Each layer is generally represented by a set of mathematical operations and optionally a set of parameters to be optimized. As layers are stacked on top of each other, the order in which their operations are performed is inherently defined. Such order of computation is mathematically represented by a computation graph that also provides a logic representation of all dependencies between computations. Hence, given a set of inputs X and their associated ground truths T, the network is built to output a prediction Y based on its optimized set of parameters. Such optimization is carried out so as to minimize a loss function$\mathcal{L}$ (characteristic of the chosen learning problem) that provides a measure of the error between ground truths and the predictions of the network. Ever since, In this general setting, neural networks have found application in a wide variety of fields, with the research on sequence[86, 87, 70], convolutional and graph[88] based neural models, but at the core they still share the same layer-like structure.

As a matter of fact, such structure has been at the center of the research on deep computer vision models since the early works on convolutional networks for handwritten digit recognition[85], LeNet invented by Yann Lecunn in 1989 [89, 90]. In LeNet-5 convolutional layers are alternated to subsampling layers. After each pair a hyperbolic tangent function is employed to squash their output values in a range $[-A, A]$, in which A is a scaling coefficient that regulates the amplitude of the hyperbolic tangent. The predictions are the results of fully connected layers attached to the last subsampling layer, as seen in 2.4.

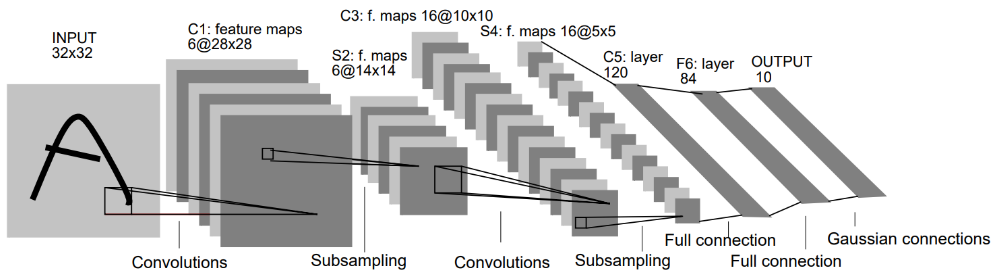


Figure 2.4: LeNet-5 architecture[85]

The repetition of the convolution-subsampling-activation pattern as a block multiple times in neural network architecture has since then become a design pattern which has also been used in the following works.

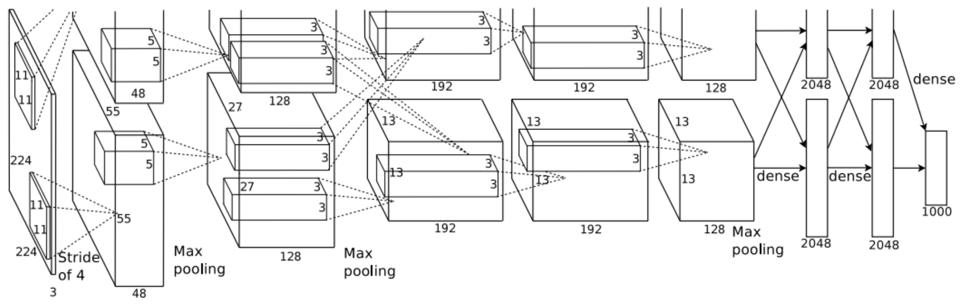


Figure 2.5: AlexNet architecture[84]. Each block represents a convolutional layer, while subsampling layers are denoted by the written captions. The terminal fully connected layers are instead denoted in the image as dense, an alternative nomenclature to the layer type

The AlexNet architecture2.5 by Krizhevsky et al. [84] followed similar design principles, although the network follows a custom design in the way layers are stacked one after the other. It is composed of five convolutional layers, of which only the first,second and fifth are followed by subsampling with max-pooling. The first and second max pooling layer are also followed by a novel local response normalization layer. All convolutional layers and fully connected ones are followed by ReLU non-linearity, a modification with respect to LeNet-5 which employed the

hyperbolic tangent function.
The former of the two has been found, as noted in [84] to help the learning procedure, as no saturation of the inputs can happen with ReLU which is unbounded, while the hyperbolic tangent can lead to loss of information if its inputs become to large, as its range of values is bounded between $[-A, A]$.
Contemporaneously to AlexNet, the paper from Zeiler et al. in 2013 [91] provided novel insights on the inner workings of neural networks. Such knowledge has been fundamental for the following work on state of the art deep learning architecture research.
As the research showed that convolutional layers, from the inputs to the outputs, learn increasingly more complex visual patterns from the dataset on which the network is trained. Such knowledge is represented within the learned parameters of the filters in each convolutional layer. As noted in both works, stacking convolutional filters on top of each other has the effect of obtaining a larger receptive field in filters which are convolved with outputs of preceding convolutional layers.
As such, the research from Zeiler et al.[91] implied that **increasing the depth of deep models would allow to learn even more complex visual patterns**, thus resulting in possibly greater classification performance. Here **depth** is intended as the number of layers of the convolutional network.
As expected, the following top performing model VGG-16 and VGG-19 [92] were built by focusing on the idea of increasing neural network depth to achieve greater classification accuracy. A further improvement of this model over AlexNet is the use of 3x3 convolutional filters at most, in order to achieve the greater network depth as well as a large reduction in the number of trainable parameters of the network: alexnet employed a larger variety of convolutional filters of size 11x11,5x5 and 3x3.
With the increase in depth, the problems of vanishing/exploding gradient and network degradation had become the focus of the following research. As network depth increases, backpropagation of errors during training results in greater probability of the backpropagated gradients to reduce to zero due to numerical underflow or explode to infinity due to the large number of multiplications that gradients go through in networks with many stacked layers.
Furthermore, it was found that hundred layers deep architectures show the degraded performance phenomenon, that instead does not appear in their shallower, yet deep counterparts[93].

With the paradigm shift brought by the **residual learning framework** published by He et al.[93]in 2015, many of the aforementioned problems have been solved.
In essence, the residual framework shifts the learning problem of groups of convolutional layers from the approximation of a generic function $\mathcal{H}(X)$ of its inputs X ,

to the learning of a residual function $\mathcal{F}(X) = \mathcal{H}(X) - X$.
Thus, learning is facilitated, as each subnetwork of convolutional layers learns representations which are referenced to its inputs, instead of arbitrarily complex patterns which can emerge at high network depth.
Indeed, for this reason the use of residual learning allowed for much deeper networks compared to the preceding works. From the VGG-16 and VGG-19, ResNet showcased stable and marginally increasing performance even when implemented with 200 layers, although with largely increased computational costs.
As a means to modularize and reduce the computational complexity of the ResNet architecture, the work of He [93] also showcases the novel BottleNeck block, shown in figure 2.6.

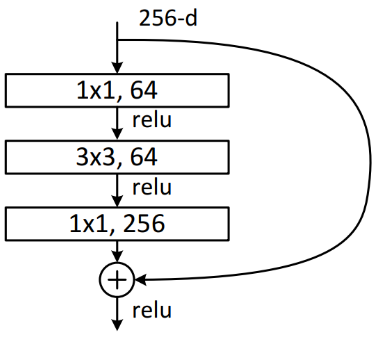


Figure 2.6: Bottleneck block proposed in [93]. Credits [93]

It implements residual learning within such block that reduces the channel dimensionality of its inputs by means of 1x1 convolutions before the application of computationally more expensive 3x3 filters. Then these are expanded again to the original channel dimension before the block outputs are computed.
Hence, as input and output channel-dimension compatibility are solved, this structure lends itself to the modularization of deep learning models by serial or parallel stacking of bottleneck blocks.

This represents the major paradigm shift in the way neural networks are designed. The focus has ever since shifted from the careful engineering of each of the stacked layers of neural networks to the development of **modular blocks** which can be stacked on top of each other thanks to their modularity in terms of compatible input and output tensor shapes.
This allowed to center the attention of the research on network level configuration parameters such as its **width, depth, resolution and cardinality of the bottleneck blocks**, as well as on the design of novel modular computation blocks,

following the principles of the residual blocks.
Width is defined in terms of the number of channels of each convolutional layer. **Depth** regards the number of layers of the network in consideration. **Resolution** is related to the size, in terms of height H and width W of the input images(tensors of shape $n\_channels \times H \times W$, in which $n\_channels = 3$ for RGB images) accepted by the networks. **Cardinality** is a notion introduced in [94], to denote th number of residual functions employed in the bottleneck blocks of the Resnets, set to one in the original ResNet paper [93]. Width has been one of the first changes brought to ResNets by [95], which showed that the power of ResNet resided in the residual learning rather than their achievable depth, as their Wide Resnet showed greater representation capabilities with **increased width** and **reduced depth to 16 layers**.
Contemporaneously to WideResnets[95], the line of work of ResNext[94] brought to the attention of researchers the **cardinality** dimension of neural networks which proved to be of equal importance to those of **width** and **depth** of deep architectures.
Xie et al 2017 [94] modified the bottleneck layers to allow the network to learn an arbitrary number of residual functions $\mathcal{F}_i$ at each bottleneck block. As a consequence, the bottleneck of a ResNext architecture computes the function $\mathcal{F}(X) = \sum_{i=1}^{C} \mathcal{F}_i(X)$ in place of the original resnet residual function. Thus the bottleneck output is expressed as

$$Y = X + \sum_{i=1}^{C} \mathcal{F}_i(X)$$

, in which X represents the standard identity mapping employed in the residual bottleneck blocks and C represents the aforementioned **cardinality**(the number of $\mathcal{F}_\rangle$ functions in each bottlneck), to be set at a network level hyperparameter of ResNext architectures.
With such modifications in place, the ResNext bottlenecks are represented by the computation graph shown in the figure 2.7

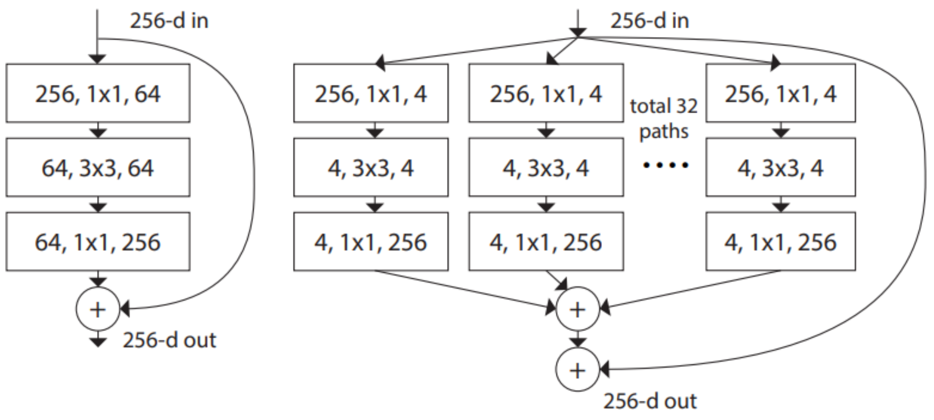


Figure 2.7: Side by side comparison between the bottleneck block of the ResNet architecture on the left and the ResNext bottleneck with cardinality $C = 32$. Image from [94]

As scaling of **depth**, **width** and **cardinality** have been explored, the **resolution** dimension had been developed as yet another line of work in Squeeze-and-Excitation networks[96].
The latter provided another means of separately scaling neural network capabilities by addressing novel network level properties.
Although not explored in this work, a notable mention has to be given to the novel works on the EfficientNet[97] architecture which provided the means to simultaneously scale **depth, width and resolution** at once.
Such problem that previously had been addressed purely by neural network engineering, is by means of the EfficientNet architecture bypassed through its compound scaling method.
Given a user defined parameter ϕ that regulates the available resources, either in terms of computation latency or FLOPS (floating point operations per second), α, β, γ found by grid search the scaling coefficients of depth, width and resolution, d, w, r respectively, can be determined:

$$\begin{aligned} depth : d &= \alpha^{\phi} \\ width : w &= \beta^{\phi} \\ resolution : r &= \gamma^{\phi} \\ &\quad s.t. \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ &\quad \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

From the determined scaling coefficient and a baseline architecture determined based on resource constraints by neural architecture search, the objective architecture is found with the defined specifications satisfied by construction.

2.3.4 Feature pyramid network architectures

The concept image pyramids dates back to early computer vision methods for feature extraction at multiple scales from a single image. Pioneering works of such techniques are the pyramid methods for image processing[98] and the SIFT algorithm [99] for extraction of semantic features. The latter showcases one of the first examples of generation of feature maps from image pyramids, as one of the first steps of such algorithm is to produce at multiple scales of the input image, feature maps that result from the difference of two copies of the input image at the same scale, but each convolved with Gaussian filters with different standard deviations[100].
It had since been known that the image pyramid data structure generated from a single input allowed to find distinct patterns that improve the performance of digital image processing and analysis techniques [98].

As such, the same concepts have also been transposed to deep learning models for computer vision.
SPPnet[101] proposed by He et al. in 2014, is one of the first architectures to employ pyramids of image features extracted by convolutional layers to tackle classification and detection problems by harnessing multi-scale representations.
Similar approaches have soon after been implemented for semantic segmentation, with the **fully convolutional network FCN** architecture [102], shown in 2.8.

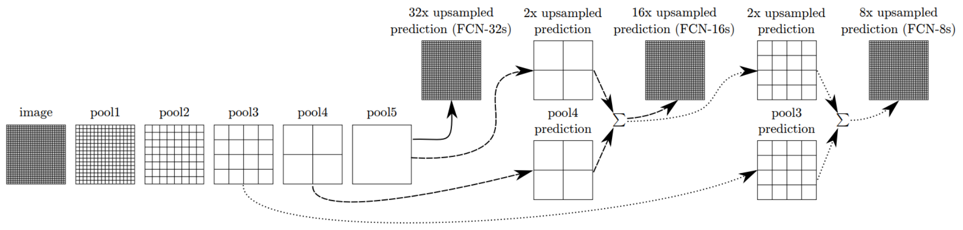


Figure 2.8: FCN architecture. Image from [102]

At the core, its novelty on the semantic segmentation performance is represented by its capability to progressively merge feature maps computed at different neural network depths, starting from the deepest layer, into new intermediate feature maps that are themselves merged with their immediate predecessor at lower network depth. Such procedure is repeated until the predecessor at $network\_depth = 3$

is used. The result of the last computation represent a semantically rich output feature map that is used to predict the semantic segmentation mask, thus using multi scale feature information to enhance the quality of its predictions.
A staple of semantic segmentation, namely the U-Net[103] architecture employed the same design principle of FCN to generate semantic segmentation outputs by means of a novel multilayer encoder-decoder fully convolutional network.

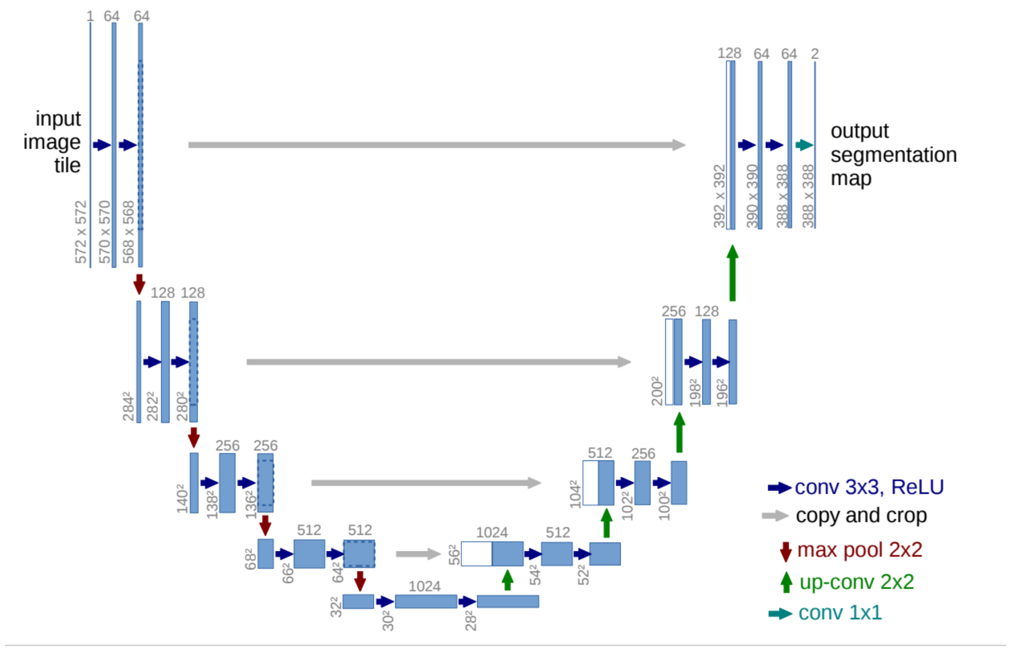


Figure 2.9: U-Net architecture. Image from [103]

As can be seen from 2.9, U-Net is similar to FCN due to its feature merging mechanism which joins upsampled feature maps at different levels of the decoder, with the feature map of the encoder at the same level in terms of encoder-decoder network depth.
Its distinctive feature from FCN is the capability to be trained with a relatively limited number of samples, as the U-Net is designed to be applied on small biomedical image datasets.

Afterwards, the concept of pyramids of feature maps had been employed also for object detection models. Multi scale object detection approaches such as MS-CNN[104], proposed ways to employ feature pyramids for detection, in contrast to previous methods that either used only the deepest and simultaneously semantically rich output layer of convolutional networks or upsampled the input image and performed detection on copies of different scale of the same image[105][106]. The

need to use multi-scale representation, by means of any of such methods, appeared from the fact such methods were ineffective to detect small to mid scale objects, due to the sole use of the deepest convolutional layers that have a too large receptive field[91], or by the fact that computing the detections over multiple copies at different scales of the same input image was computationally inefficient.
Still, no feature fusion mechanism such as the one of the FCN had yet been developed.
One further step ahead to solve the computational inefficiency of convolutional detectors, as well as the poor performance on detection of objects at all scales, has been achieved by the Single-Shot multibox Detector SSD[45] architecture. Its main advancement has been the one-stage object detection by means of the multi-scale feature maps of its VGG16 backbone feature extractor[92].

The major advancement in deep learning object detection models had been achieved by the **feature pyramid networks** FPN developed by Lin et al. [107]. With the objective to develop a neural architecture capable of robust multi-scale object detection, the FPN architecture is built with a set of design principles that reflect those of the FCN[102] shown in 2.8, and the U-net [103] model. It is defined by two main components: a bottom-up pathway and a top-down pathway.
As the bottom-up pathway generates output feature maps at multiple scales, from high resolution generic details to low resolution but object-level patterns, the top-down pathway merges them so as to create semantically rich features at all levels of the original feature pyramid.
The bottom-up pathway consists of the selected output feature maps of a backbone architecture of choice (e.g. ResNet50-101, ResNext, EfficientNet without its fully connected layers), while the innovative top-down feature hierarchy is designed based on the choice of bottom-up feature maps.
To implement the network of feature pyramids, as suggested by its name, lateral connections between the two pathways and top-down connections between neighboring levels of the top-down pathway are designed and employed as shown in figure 2.10

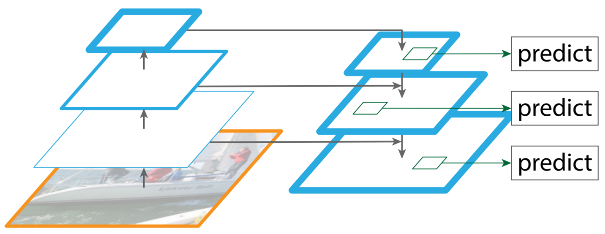


Figure 2.10: FPN architecture. Image from [107]

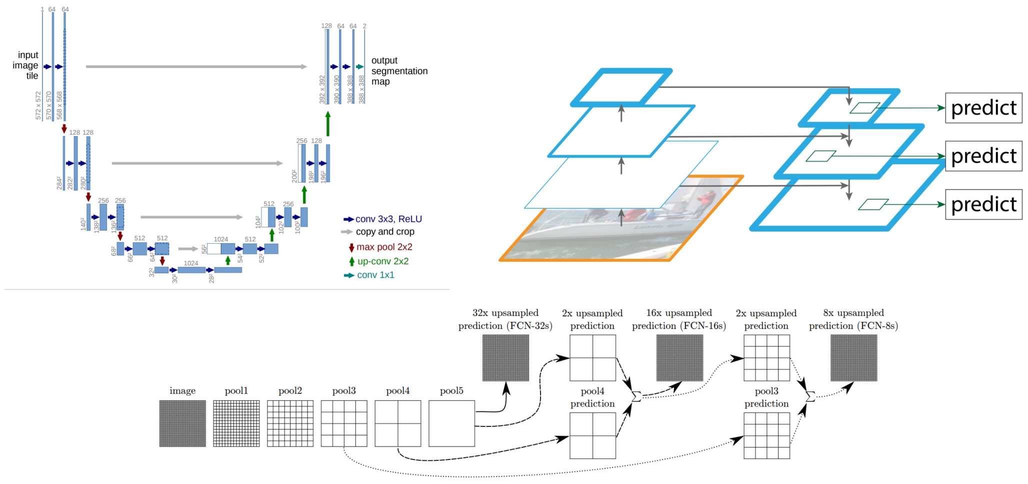


Figure 2.11: U-Net(top-left), FPN(top-right) and FCN(bottom) architectures. Respective credits[103] [107] and [102]

As is shown in figure 2.11, at an architecture level, U-Net, FCN and FPN adopt the same feature merging mechanism, starting from the output feature maps of the deepest layer of their respective backbone architecture and progressively merging those rich image-level semantic features with the fine-detail level features of the layers closer to the network inputs[91].
The stark difference between the three, from a model architecture perspective, is in the way the lateral and in-between merged feature map connections are designed. **U-Net** simply crops feature maps coming from lateral connection in order to match width and height of the feature maps with which it is merged by simple channel-wise concatenation. The merged result is then forwarded to two convolutional layers, each with 3x3 filters followed by ReLU non-linearity. Such output is then upsampled by a factor of 2 by bilinear interpolation and then passed to a convolutional layer with 2x2 filters that halve the number of input channels. The procedure is repeated at each level of the decoder part(the right side of the U-shaped architecture).
On the other hand, **FCN** upsamples by a factor of 2 the feature maps by means of deconvolutional layers [108] and then performs element-wise summation with the feature map of the backbone architecture with the same dimensionality.
Similarly, **FPN** model upsamples top-down feature maps by a factor of 2 and then convolves with 3x3 filters, to then element-wise sum it with the same level feature map of the bottom-up pathway. The latter is prior to the sum, convolved with 1x1 filters in order to match the number of channels of the features with which it is element-wise summed.
The promising results shown by feature pyramid networks[107] motivated further research developments on such models.

A fundamental extension of the bottom-up to top-down FPN architecture, is the PANet[109] which develop a further bottom-up pathway which is however built from the preceding top-down pathway so as to generate even richer merged feature maps.
Hence, as seen in comparison figure between FPN and PANet 2.12, which represents the backbone features of bottom-up pathway with the leftmost circles, these are merged as in the standard FPN model on the left, by the same top-down pathway. The novelty of the PANet architecture can be observed on the rightmost bottom-up pathway, marked with red bottom-up connections, which further merges the combined features obtained by the top-down pathway in the middle of the PANet architecture.

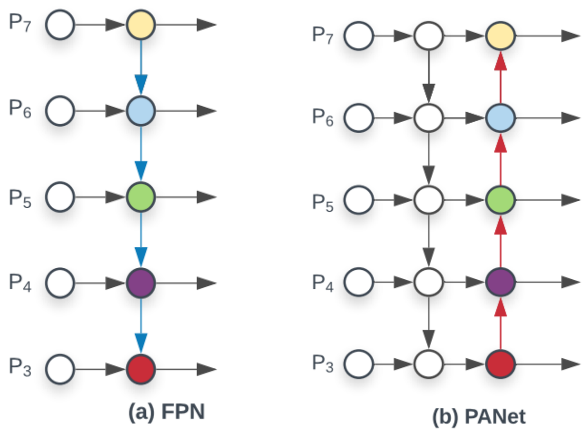


Figure 2.12: Side by side comparison of FPN on the left and PANet on the right. Image from [110]

With proper construction of the top-down into bottom-up pathway implemented in the PANet architecture, this novel block which comprised the two pathways lends itself well to the repeated use of such structure.
Indeed, the follow-up model NAS-FPN[111] shown in figure 2.13 considers such construction as a new type of deep neural network layer, which is attached to the selected feature maps of the initial bottom-up pathway $p2$ to $p7$. As can be seen, the novel layer can be repeated multiple times by simple connection of repeated top-down into bottom-up layers to the right side of the preceding its preceding merged feature maps.

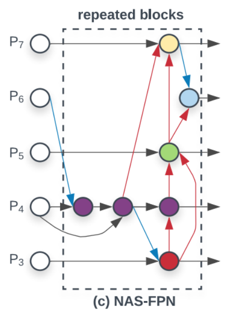


Figure 2.13: NAS-FPN architecture. Image from [110]

Furthermore, NAS-FPN utilizes neural architecture search to determine the optimal set of connections between the repeated top-down and bottom-up pathways, so as to find the optimal problem specific architecture.
However, as the NAS-FPN is built by means of the neural architecture search, Tan and Pang [110] argue that such model, although powerful is often impractical due to the large number of parameters it contains, hence it is impractical in any resource constrained environment.
As such, they develop the bidirectional FPN architecture BiFPN [110] shown in figure 2.14, as a modular and simplified detection model, with respect to NAS-FPN.

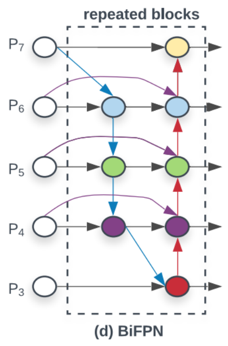


Figure 2.14: BiFPN architecture. Image from [110]

Furthermore, they use such baseline template to develop a family of FPN models, EfficientDet (figure 2.15), which can be scaled according to the available resource in which it is employed, through a compound scaling coefficient. As such, they re-use the same principles of the EfficientNet family of feature extractor to overcome issues that hindered previous FPN based detection models.

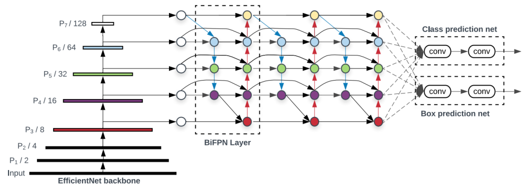


Figure 2.15: EfficientDet architecture. Image from [110]

2.3.5 Multi-task learning

The premise of a unified deep learning model that is capable of tackling a diverse set of tasks with high effectiveness through its jointly learned representations, has attracted the interest of much of the recent research.
As it allows to greatly reduce model complexity inference time by employing one model instead of multiple architectures, each specific for a single task, it is also of major interest for resource-constrained real world applications.
A staple of multi-task learning are the object detection deep models.
The widely renowned object detection models Fast R-CNN[106] and its improved version Faster R-CNN[112] showed that simultaneous learning of classification, bounding box regression and objectness classification (in the case of Faster R-CNN, for object or background binary classification of proposed regions of interest), is feasible and allows the unified model to achieve remarkable detection quality.
Although such tasks are by-design required to generate detections, the multi-task learning problem has been more broadly studied also in the case of joint learning of less strictly bound, multi-task contexts.
The most widely addressed multi task setting is that of joint object detection and semantic segmentation[113, 114, 115, 116, 117, 118], as these methods observe that a joint learning approach simultaneously improves the detection and segmentation metrics. Such empirical findings are also supported by the fact that context surrounding objects aids in detecting them by means of contextual reasoning [49] e.g. a car is seen most commonly surrounded in the lower part by road-like features and much less commonly surrounded by vegetation, or pedestrians are generally observed closer to each other and in the sidewalks of the scene.
The typical convolutional architecture followed by such methods is to employ only the fully convolutional parts of feature extractor models (such template is also named backbone) such as ResNet and to attach separate re-engineered task-specific sub-networks (also named branches) to either its intermediate convolutional layers or to its terminal end.[113, 114, 115, 116, 117, 118]
Each of such branches is dedicated to its assigned task and learns task-specific patterns, but the shares the same upstream common backbone component with other tasks.

However the decision of attaching a branch to a given layer remains arbitrary in such models. Instead, it has been shown by Srivastava and Misra[119], that such selection should be task dependent and is crucial to improve the performance of multitask models.
Their Cross-Stitch network architecture aimed at addressing such neural network engineering issue, as the proposed model learns end-to-end the optimal multi-task architecture, by determining through learned weights the optimal way to separate task specific from task agnostic features.
From an architectural point of view, the Cross-Stitch network provides multiple insights on the multi-task learning problem.
However this improvement does not address a fundamental issue about training multi-task neural networks: how should each task-specific loss be weighted when added to the joint loss?
Indeed, it can happen that one or more of the errors computed between prediction-ground truth pairs, represented by the task-specific loss, can overwhelm with its contribution those of the other tasks in multi-task loss.
This point has been addressed by Kendall in 2018 [117], who developed a task-uncertainty based loss-reweighting mechanism to rescale task-specific losses so as to obtain separate contributions at the same scale in the total loss.
Although such approach provides a promising mechanism to avoid the manual assignment of weights to each component of the loss, much of the following literature has still employed grid search-like approaches to find the optimal weighting of each loss component.
Such approach is also followed by much of the literature developed afterwards.
Examples of such statement from the literature that are related to the work in this dissertation, are PanopticFPN [65], Panoptic Deeplab[21] and the EfficientPS [22] models, which respectively tune by grid-search the weights or assign equal weights to each of the components of the total loss.

2.3.6 Instance segmentation models

While semantic segmentation is a comprehensive approach to holistic scene understanding, one its major limitation to this end is the inherent lack of ways to differentiate between entities whose more appropriate representation is that of separate and distinct objects.
As such, semantic segmentation is effective to segment the background or other uncountable entities, known as stuff, which are well represented by amorphous regions e.g. the sky, the road and the vegetation.
On the other hand objects with well defined shapes, categorized in literature as things [49], benefit from object detection and recognition.

However the latter provides only coarse annotations without the fine-granularity provided by segmentation-based algorithms.
For these reasons, the line of work on instance segmentation garnered much attention in scene understanding research.
As the task requires to discriminate between instances belonging to the same semantic category, a variety of methods exist in the literature to address the complexity of instance segmentation. The main subdivision can be done over the methodology that is employed to generate instance proposals and the associated semantic class: two-stage and one-stage methods.
The two-stage models are the most widely researched and the ones which provided the highest performance among all.
Two-stage methods can be further categorized based on the segmentation pipeline:

- **object-detection based**: in which a detector determines object bounding boxes and the class of the object within it, followed by instance segmentation of the area of the input image within the associated bounding box

- **segmentation based**: a model generates instance segmentation proposals and per-instance category labels are assigned based on pixel-wise classification(semantic segmentation) of the whole scene.

Object detection based methods employ a first stage which generates candidate box proposals, most commonly by means of the region proposal network RPN [112] of the Faster R-CNN architecture, which are then used to create image crops that are fed to a downstream instance segmentation network.
Such component is responsible for the generation of object proposal for each pixel of its input feature maps, each of which is assigned a set of rectangular anchors(in essence, location independent bounding boxes) of multiple scale and size. After the assignment anchors become localized with respect to the assigned feature map pixel. Then for each of such preliminary assignments, the RPN predicts an objectness probability and bounding box coordinates. The former allows to determine whether an object is within the proposed region, while the box coordinates are used to refine the anchor location with respect to the associated ground truth.
The RPN-based approach has been followed to build the fully convolutional instance segmentation network FCIS[120], which utilizes the position sensitive score map approach first seen in the instance proposal model InstanceFCN[121], to assemble instance segmentation starting from the object proposals obtained by the RPN.
Following the same philosophy, Mask R-CNN[68] has become a top performing instance segmentation architecture still for the modern standards, by adopting the default Faster R-CNN[112]. Its ROI pooling layer is replaced by a novel ROI align layer to finely localize the predicted instance masks on the input image and an instance segmentation branch is added in parallel to the Faster R-CNN box

regression and classification branches.
As has been mentioned, Mask RCNN employs the same upstream RPN module of Faster R-CNN to generate the candidate object proposals, which are then used to jointly perform instance segmentation, bounding box regression and classification.

On the same line of work another notable method is Hypercolumn[122], which employs object and instance proposals generated by multiscale combinatorial grouping [123] that are then passed to R-CNN[105]. Its filtered bounding box predictions, by means of non-maximum suppression, become the hypercolumn inputs that are required to predict the instance mask.
Both instances proposals and the respective bounding box are considered to generate the image crop which is fed to a downstream convolutional architecture for segmentation.
A feature vector called in fact hypercolumn is created from the convolutional feature maps of the chosen convolutional architecture, all rescaled to a common size so as to obtain the required hypercolumn vector.
Such data structure provides multi scale feature representations which are fed to multiple classifiers, whose outputs are aggregated to then generate the final instance segmentation. However such method assumes that the upstream detector can also provide the category associated with the segmented instance.

As Mask R-CNN has become a staple of instance segmentation, research efforts have also been dedicated to alternative methods with premise to possibly obtain comparable results. Deep watershed transform DWT[124] is one of such methods. All object instances are considered as energy basins, which the DWT two-stage deep model separates by means of a per-basin predicted direction of descent of the energy, by means of its direction network DN. As the heatmap of the energy distribution is obtained through the second stage represented by the Watershed Transform Network WTN, a thresholding procedure based on the watershed algorithm is applied in order to finally separate all found instances.
A category is assigned to each of the instances by means of a preliminarily computed semantic segmentation mask which allows to filter out all image regions unrelated to the categories of interest from the input to the DWT network.
SSAP[125] adopts a similar procedure, but the semantic segmentation branch and the affinity pyramid used to compute preliminary per-pixel affinities share the same upstream convolutional feature maps.
Both semantic and per-pixel affinities are merged by a cascaded graph partition module which formulates the instance mask generation as an graph partition optimization problem.

Another novel line of work is that of dense sliding-window instance segmentation models.

Firstly initiated by the DeepMask[126] instance proposal model, TensorMask[127] is the first instance segmentation model which simultaneously predicts instance masks and the associated category without any additional semantic segmentation pipeline or upstream object detector.
It achieves this by means of a 4D tensor representation of shape $V \times U \times H \times W$ that allows to model both the spatial location of instances on the $H \times W$ image planes, simultaneously to the relative instance mask positions on the $V \times U$ planes. Such model represents a notable innovation as it achieves comparable results to Mask R-CNN with a novel approach to instance segmentation, thus opening possible avenues for improvements over the current state of the art detection-based models.

2.3.7 Semantic segmentation models

With the objective to assign pixel-wise semantic labels to all pixels of the input image, the state of the art approaches in literature employ **fully convolutional neural networks**[102](FCN) as a means to learn the mapping from input pixels to output pixel categories.
The prevalent approach to the implementation of FCN semantic segmentation network is based on the principles of the Autoencoder model[128]: an input is compressed to a **latent embedding** representation by means of the learned parameters of an **encoder network**, to then be decompressed to the original input size through a **decoder network**.
The decoder output is generally task-dependent, whether it is for the purpose of denoising inputs [129], extraction of the compressed embeddings obtained between the encoder and the decoder[130, 131, 132] or semantic segmentation[103, 102, 133, 21].
For the purpose of image semantic segmentation, segmentation models output tensors of shape $H \times W \times n\_classes$, in which H and W are the height and width of the input image grid, and $n\_classes$ pixel-wise class logits which are utilized to determine the semantic label of each pixel.
In practice, this procedure can also be described as pixel-wise classification.

In the literature, it is widely understood that to achieve optimal segmentation performance, deep networks are required to present great localization capabilities of semantic entities[134]. To achieve this, ParseNet[135] adopts a modified FCN [102] in which its features that are utilized to perform pixel-wise classification are merged with global context ones(those of the terminal pooling layer) by means of a contextual module, before they are employed for segmentation.
Thus, the local features are combined with global context to generate intermediate multi-scale features that allow for better segmentation outputs.
Following the same principles, the DeepLab architecture[134] has been developed.

It consists of a fully convolutional segmentation architecture and a downstream fully connected conditional random field (CRF), that allows the refinement of the segmentation outputs of the network by considering global context of the predicted pixel classes. The fully connected CRF achieves such functionalities by considering the degree of correlation between all possible pairs of pixels. Hence it solves conflicts in the predictions by considering color intensities of the pixels and the associated predicted labels, building an energy function which is to be minimized so as to eliminate discrepancies in the predictions. This way, boundaries of entities in the scene are refined and improved.
Furthermore, DeepLab extends such implementation by considering also multi-scale feature aggregation, naming such model DeepLab-MSc, by providing to the pixel-wise classification layer a concatenated set of features obtained at multiple depths of the convolutional backbone.
With the aim to jointly consider local and global context, DeepLab has also been extended to the use of atrous convolution, DeepLab-LargeFOV as a means to consider broader receptive field of the convolutional filters used in the network, without compromising its computational complexity.
As the most promising results have been obtained by the use of atrous(also named dilated) convolutions, DeepLab has been further extended with a novel atrous spatial pyramid (ASPP)[133] pooling which allows to generate multi-scale features from the same input, by employing multiple filters with different dilation rates.
Thus, the receptive field of the employed filters is artificially enlarged so as to cover broader areas of the input, without requiring additional parameters. This is due to the fact that atrous convolution pads filters with zeros in-between the original filter elements, until the desired filter size is obtained.

One critical distinction between the atrous convolution-based models, such as the family of DeepLab architectures, and encoder-decoder based segmentation deep models is in the way predictions are computed. The DeepLab-based models generate segmentation predictions at a subsampled scale of the inputs at training time, while during inference the output of the semantic segmentation is bilinearly upsampled. As described in [134], such operation does hinder the model performance.
However, another line of work on segmentation models employs a more intuitive approach to segmentation, by means of encoder-decoder architectures.
In such context, inputs are progressively downsampled as they are passed through the encoder part, to then be upsampled by learned deconvolution[102, 136] or bilinear upsampling[137, 135, 103] until the original image shape is recovered. Then, the resulting output of the decoder is used to perform pixel-wise classification.

2.3.8 Panoptic Segmentation models

Panoptic segmentation models push the envelope of holistic scene understanding by inheriting the components of previously separately studied semantic and instance segmentation architectures.
Although interest in such line of work has recently been revived by the research of Kirillov et al. [20, 65], pioneering work on panoptic segmentation dates back to 2005 with the study of Tu and Chen [138] on Bayesian methods to unify semantic segmentation, detection and recognition.
However only with the demonstrated breakthrough performance of deep learning architectures for computer vision, the panoptic segmentation task garnered much research attention.
With such promising premises, researchers quickly developed novel approaches to the task.
As the task involves the joint understanding of objects and their surrounding background, instance and semantic segmentation research has been utilized to build unified multi-task models capable of simultaneous detection of instances, with their associated category, and segmentation of the background scene.
One of the first approaches to the problem has been the PanopticFPN model[65], which consists of a modified state of the art instance segmentation architecture, Mask R-CNN, to which a semantic segmentation branch has been added. With the joint prediction of instances and of the semantic segmentation, respectively by means of the Mask R-CNN and the newly added segmentation branch, a downstream post-processing algorithm merges the two separate sets of prediction into a final panoptic mask, solving any possible overlaps between them.
After such early work, the literature has quickly been enriched by a variety of panoptic segmentation models which improved upon this baseline architecture.
Two main lines of work have developed as a consequence, **proposal-based**(or top-down) and **proposal free models**(or bottom-up)[21]. The distinction stems from the methodology that is employed to detect instances in the scene.
Proposal-based models employ the Mask R-CNN [68] pipeline. In essence, it consists in the detection objects, to which a 2D bounding box is assigned, and the image region enclosed in such rectangular area is segmented downstream by a convolutional network to produce the respective instance segmentation.
The naming originates from the fact that Mask R-CNN bases its detection methodology on the proposal of a large number of bounding boxes which are then ranked, filtered and refined before they are finalized as model predictions.
Under this category, there are UPSNet[23], EfficientPS[22] and the previously described PanopticFPN[65].
UPSNet and EfficientPS improve upon the latter by implementing a parameter free end-to-end module that allows the in-network generation of the panoptic

segmentation prediction.
It achieves such objective by merging the instance mask logits with those that match the respective bounding box on the semantic segmentation mask, to then concatenate the merged instance logits with the semantic segmentation logits and generate the final prediction by selection of the highest confidence score class from the mentioned concatenated logits tensor.
While UPSNet merges the instance and its semantic segmentation logits by simple element wise sum over the spatial locations of the respective masks, the EfficientPS adopts a more complex consensus mechanism based on accordance between mask and semantic predictions.
Another model, CVRN[24] employs a proposal-based approach by modifying the PanopticFPN architecture to address the domain adaptation problem.
On the other hand, **proposal-free** models bypass the use of the Mask R-CNN pipeline by employing alternative methods to predict instance segmentations.

Proposal free methods instead recur to novel techniques to tackle the instance segmentation sub-task of the panoptic segmentation problem.
The most notable methods employ the concept of instance center of mass to detect object instances and obtain their respective segmentation.
DeeperLab[139] is the first to use such concept, by framing instance segmentation as a keypoint prediction problem. In such case, the model consists of a semantic segmentation branch and four keypoint-prediction related branches. Each of the four is used to predict a type of relationship between each pixel and the keypoints of the associated instance. By means of these prediction heads, a class agnostic instance is generated. Its semantic label is obtained by majority vote through the semantic segmentation prediction obtained by the parallel segmentation head.
A similar approach named PanopticDeeplab [21], which consists of a common backbone architecture to model commonalities between instance and semantic segmentation, with two separate instance and semantic branches that learn task-specific representations, employs the same concept of instance center of mass prediction.
Instance centers are predicted as an image level heatmap based on a 2D gaussian encoded ground truth heatmap. Redundancies are then suppressed by keypoint-Non-maximum suppression. The remaining centers are then utilized to perform regression of the offset of each pixel to the nearest instance center. The resulting class agnostic instances are then given a semantic label through a majority vote(same as [139]) which associates the most common category of the semantic segmentation branch prediction, considering only the pixel-wise semantic labels that are within the predicted instance mask pixels. Thus, the panoptic prediction is determined.

In the same context of proposal-free methods, SSAP[125], treats instance segmentation as graph partition optimization problem, aiming at separating pixels which belong to different instances by means of its develop concept of pixel-wise affinity pyramid.

2.4 Self-supervised Domain Adaptation

Deep Learning-oriented approaches are notoriously data hungry, which translates to the need for a considerable amount of training data even for trivial tasks. Collecting data for training more complex tasks can become unfeasible in practice, as suggested in the work of [2]. The authors prove that it is required to collect for more than 600 years of driving hours, to attain acceptable performances with self-driving vehicles. A natural fallback is simulation, which in turn, entails a difference between the samples generated in the simulation environment and the ones collected in reality. Samples synthetically generated through simulation are said to belong to a domain which is *shifted* from the domain of real data. Domain Adaptation (DA) techniques offer a set of strategies to correct this unwanted discrepancy, with the goal of learning a feature representation which is *meaningful* for the main machine learning task, while being *invariant* from the domains samples come from.

More in general, domain adaptation is a powerful framework which can be applied to any scenario in which there is a shift between a *source* domain, namely the training set, and the *target* domain, commonly identified with the test set. We assume that the model is trained on points sampled from a dataset $\mathbf{x} \sim X \subseteq \mathbb{R}^n$ with corresponding discrete labels $y \in Y$, belonging to a finite set $Y = \{1, 2, \ldots, L\}$. We identify two distributions for the source domain and the target domain, $\mathcal{S}(x, y)$ and $\mathcal{T}(x, y)$ on $X \times Y$, and their difference is addressed as *domain shift*.

The ultimate goal of the domain adaptation task is to exploit the information retained in the examples belonging to the source domain in order to make prediction on the target domain. To this end, self.supervised domain adaptation techniques introduce an auxiliary task which is solved in tandem with the main machine learning one. The auxiliary (or pretext) task introduces an additional term in the overall loss which does not depend on the labels set Y, and has the goal of indirectly forcing for a reduction of the domain shift, during training. As a result, the auxiliary task allows to reformulate the optimization problem with the goal of learning features which are discriminative and domain-invariant at the same time. Self-supervised domain adaptation defines a pretext task which can be solved without the need of costly human labeling. In fact, the objective function to minimize can be designed to be deducible from the structure of data. An example is training a classifier to discriminate from samples belonging to either source or target domains, as proposed by [140].

2.4.1 Domain adaptation methods

The methods to implement self-supervised domain adaptation can be divided into three broad classes. A first class aims to reduce the divergence between the source and the target domains in some feature space. To achieve that, we optimize for some measurement of distributional discrepancy. One example of measurement is the maximum mean discrepancy (MMD). It is defined by a feature map $\phi : X \to H$, where H is called a reproducing Hilbert's space. MMD represents the discrepancy between two distributions as the distance between the mean of the embeddings of the samples belonging to source and target domains

$$MMD(S,T) = ||\mathbb{E}_{\mathbf{x}\sim S}\left[\phi(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim T}\left[\phi(\mathbf{x})\right]||_H$$

In the case of deep learning, the MMD objective is computed on some latent space resulting from the transformations applied by the layers of an Artificial Neural Network to the input feature space, before the classification layer. This term is added to the loss an minimized during the overall training task. Some authors employing this technique in their works are [141, 142, 143, 144].

Another approach of achieving self-supervised domain adaptation involves resorting to adversarial methods, taking inspiration from the idea initially introduced by Generative Adversarial Networks (GANs). In fact, domain adaptation can be achieved by training a domain discriminator and a generator (features extractor) in an adversarial fashion, with the result that the generator produces indistinguishable features representations for source and target samples, achieving the task of learning domain-independent representations for samples. This is also the approach implemented n this work, following the intuition of [140].

The third class of domain adaptation techniques relies on solving one or more pretext tasks, besides the main machine learning task, involving both source and target samples. The key intuition behind these methods is that solving the auxiliary tasks for samples belonging to both domains entails learning robust domain-independent features. For instance, the work of [145] propose to identify the rotation of source and target images. The work of [146] extends this idea by solving K pretext tasks at the same time, obtaining a stronger domain-invariant features representation, at an higher computational cost during training. The authors of this work include in their method the usage of MMD in the process of hyperparameters tuning, which makes this method a bridge between two classes of approaches to address the task of domain adaptation.

Chapter 3

Synthetic-to-Real dataset creation methodology

3.1 Generation of a label-matched simulated and real world image dataset

As the panoptic segmentation task requires instance and semantic segmentation ground truth formats, it has been necessary to define a synthetic-to-real autonomous driving scene dataset with built in capabilities to perform panoptic segmentation by design.
Furthermore, domain adaptation sets the need of multiple datasets originated necessarily from different sources, but which share a common underlying scene context.
On top of these requirements, all semantic categories of all ground truths of the different datasets have to be matched, such that each class is represented by the same numerical identifier across the different datasets.
As the problem of interest of this work entails the use of domain adaptation to mitigate the distributional shift between **synthetic and real data** on the **panoptic segmentation task**, the **source dataset has to be generated in simulation**.To this end, the CARLA simulator has been employed as a means to capture the most diverse set of driving scenarios from different camera perspectives. Such dataset consists of an order of magnitude more images than the chosen real world datasets from the literature, Cityscapes and BDD100k(its 10k samples variant), in order to assess whether the diversity provided by simulation can indeed provide benefits to a domain adapted deep learning model.
Nevertheless, such diversity is only presented in terms of storage, as **during training of the deep learning model, the per-batch of samples synthetic to**

real ratio is kept 1:1, to avoid over-representation of the synthetic data and consequent loss of relevant information from the real world images.
The **target domain** is the real world data, as is mentioned above.
Hence, the latter has to either be manually gathered from a real driver perspective and then be labeled, or belong to already labeled real world datasets presented in the computer vision literature.
For ease of implementation, two renowned multi task learning driving datasets from the literature, **Cityscapes**[60] and **BDD100k**[62] have been utilized to create the customized real world datasets of this experimental setting.

3.2 Synthetic dataset creation

3.2.1 The CARLA simulator

The CARLA[6] simulator provides a broad set of tools and capabilities to perform simulation regarding a wide variety of autonomous driving task.
As it is by-design integrated with the Unreal4 game engine, it has the capabilities to render high quality simulated scenes, with an high degree of realism3.1.

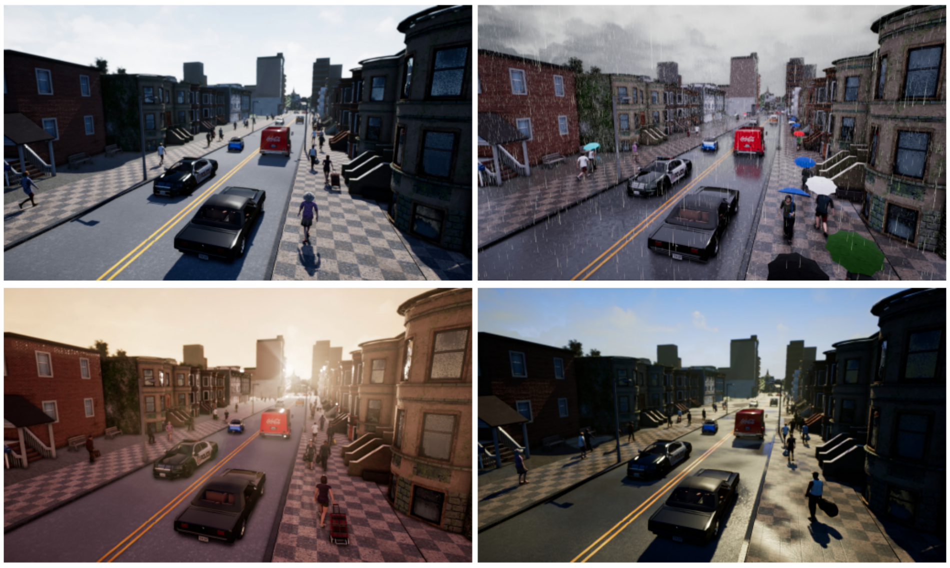


Figure 3.1: A diverse set of scenes can be captured as images by the CARLA simulator: weather and daylight conditions, sun glares, road reflectance of light and traffic density are only a few of the features that can be simulated (source[6]).

By means of its modular design, it is possible to configure:

- simulation sampling frequency (at least 10 Hz)
- the map design employed during a given simulation, by selection of any of the already provided maps or custom designed ones.
- traffic density
- pedestrian density
- type of the simulated vehicles
- weather condition
- daylight condition
- randomness of the behavior of driving agents and pedestrians
- texture variability of objects in the scenes, while maintaining semantic coherence

As CARLA is designed to support the simulation of autonomous driving related tasks, the simulator comes with the capabilities to capture data from the following sensors :

- RGB camera
- depth camera
- LiDAR sensor
- GNSS sensor
- IMU sensor
- Radar sensor
- Event camera

Furthermore, the simulator allows to automatically generate a ground truth annotation associated to the raw data captured from each of the mentioned sensors. To this end, CARLA provides the following annotation tools:

- image semantic segmentation annotator (RGB,depth,Event camera annotations)
- image instance segmentation annotator (RGB,depth,Event camera annotations)

- LiDAR point cloud semantic annotator (LiDAR)
- Optical flow annotator (for RGB,depth,Event camera annotations)

Hence, the **RGB camera** and the relative **semantic and instance segmentation annotator** have been used for data collection purposes. Due to the fact that the camera is simulated, its intrinsic properties are manually selected prior to the simulation. The pinhole camera model[147] is employed to define the necessary **camera intrinsic parameters** matrix. **The extrinsic parameters** do not have to be estimated, as each camera has been manually placed at set locations on the chassis of the simulated vehicle.
As such, the following camera parameters are considered for the definition of the sensor setup:

- width: Image width in pixels.
- height: Image height in pixels.
- FOV_x: Horizontal field of view in degrees.

Hence, the intrinsics camera calibration matrix can be directly computed from the aforementioned user-defined values:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where, c_x,c_y, f_x and f_y are respectively the camera optical centers ,expressed in pixel coordinates, and the camera focal lengths.
These are computed by the relationships with the aforementioned user-defined camera parameters:

$$\begin{aligned} f_x &= width/(2.0 * \tan(FOV_x/2 * 2 * \pi/360.0)) \\ f_y &= height/(2.0 * \tan(FOV_y/2 * 2 * \pi/360.0)) \\ c_x &= width/2 \\ c_y &= height/2 \\ FOV_y &= 2 * \arctan(\tan(FOV_x/2) * width/height) \end{aligned}$$

For increased realism of the camera output images, the CARLA simulator provides the following set of options to add noise or distortions:

- Vignette: darkens the border of the screen.
- Grain jitter: adds some noise to the render.

- Bloom: intense lights burn the area around them.
- Auto exposure: modifies the image gamma to simulate the eye adaptation to darker or brighter areas.
- Lens flares: simulates the reflection of bright objects on the lens.
- Depth of field: blurs objects near or very far away of the camera.

All of these options are turned off in this experimental setting, to avoid any confounding effect due to such selection on the deep learning model performance during the synthetic-to-real domain adaptation.
Although noise would allow the deep model to learn more robust features from the data, it could also hinder the training process, hence these are turned off.

3.2.2 Synthetic driving scenarios

The CARLA simulator provides a set of already built maps of plausible real world scenarios.
The available selection entails:

- Town01, A basic town layout consisting of "T junctions".
- Town02, A smaller version of the Town01 map.
- Town03, The most complex town: a city-like environment with a 5-lane junction, a roundabout, multiple road levels, a tunnel and an urban periphery.
- Town04, A rural town with country roads, an highway and a small town.
- Town05, A Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction. Useful to study lane change scenarios.
- Town06, Long highways with many highway entrances and exits.
- Town07, A rural environment with narrow roads, barns and hardly any traffic lights.
- Town10, A city environment with different landscapes such as an avenue, a promenade, a city hall, a train station and a sea scenery.

For the purpose of generating a wide variety of scenarios, in this study **Town03** and **Town10** have been selected as base environments for the simulations, mainly due to their inherent complexity.
In a preliminary version **Town04** had also been included, in order to add rural

areas to the variety of scenes, but the simulator produced broken instance and semantic annotation for terrain at distance greater than 1km. Rendering only semantic classes within 1km radius from the camera.

Considering the selected Town03 and Town10 driving scenarios, diversity in the captured data has been achieved from the perspective of actors in the scene by simulating a wide variety of classes of vehicles, from trucks to motorbikes, bikes and passenger cars. Furthermore a variety of pedestrian models have been also simulated: from adults to elders and children.
Regarding these, a subset of all vehicles and pedestrian have been automatically sampled by the simulator and added to the scenario, based on the selection of the number of desired actors:

- Town03, 50 vehicles and 34 pedestrians
- Town10, 45 vehicles and 40 pedestrians

To attain diversity also in terms of driving scenes, the simulation has been set up in order to capture with equal probability scenes of overcast, thunderstorms, wind, fog or clear weather conditions at all hours of the day: from dawn, to sunset and night-time.
The weather conditions have also been simulated considering different levels of impact on the driving scenario.
To simulate different weather and illumination conditions, the following modifiable parameters have been utilized (description taken from the [6] documentation):

- cloudiness: values range from 0 to 100, being 0 a clear sky and 100 one completely covered with clouds.
- precipitation: rain intensity values range from 0 to 100, being 0 none at all and 100 a heavy rain.
- precipitation deposits: determines the creation of puddles. Values range from 0 to 100, being 0 none at all and 100 a road completely capped with water. Puddles are created with static noise, meaning that they will always appear at the same locations.
- wind intensity: controls the strength of the wind with values from 0, no wind at all, to 100, a strong wind. The wind does affect rain direction and leaves from trees, so this value is restricted to avoid animation issues.
- fog density: fog concentration or thickness. Values range from 0 to 100.
- wetness: road wetness intensity. Values range from 0 to 100.

- sun azimuth angle: values range from 0 to 360. Zero is an origin point in a sphere determined by Unreal Engine.
- sun altitude angle: values range from -90 to 90 corresponding to midnight and midday each.

All such conditions have been **simultaneously implemented** in the simulation of the Town03 and Town10 scenarios.

3.2.3 Sensor setup

The simulator adopts a **left-handed coordinate system**, with:

- x: forward
- y: left
- z: up

Based on such definition, the sensor setup has been attached to a Tesla model3 template vehicle provided by the simulator.
Such sensor setup for the purpose of the creation of a dataset is also inspired by the way in which camera images have been gathered in the renowned KITTI dataset [59] and the more recent Waymo perception dataset [148].
The sensor setup follows the standard camera setup of a Tesla vehicle, shown in figure 3.2.

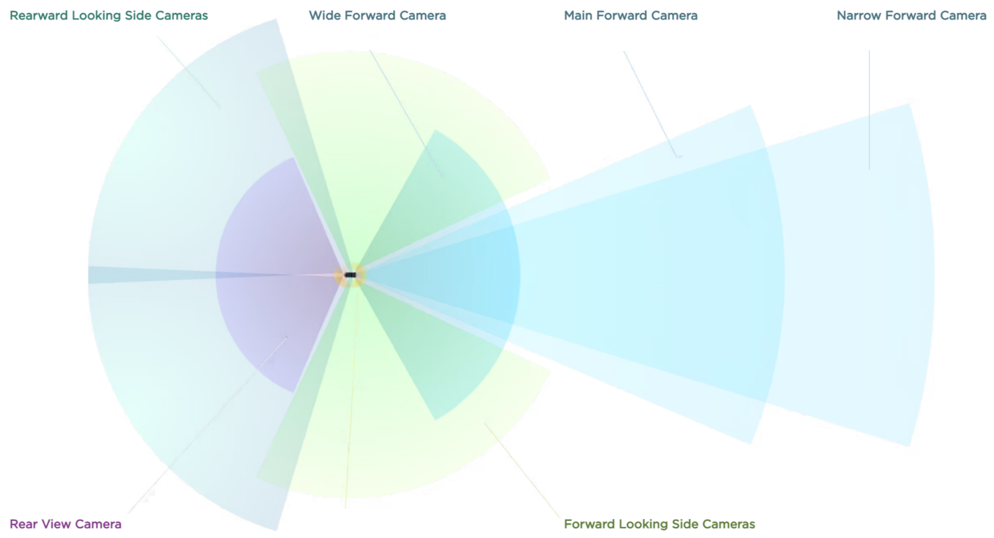


Figure 3.2: Multi camera sensor setup employed in Tesla vehicles

Hence, the implemented sensor setup that has been used to gather simulated data is an approximated version of the one employed by Tesla [149] shown in figure 3.2.
It has been chosen because it provides a wide variety of different perspectives on the driving scene, with overlapping camera fields of view.
Fixed image width $W = 960$ and height $H = 540$ have been set for all camera sensors.
The horizontal FOV of each camera is instead configured following the setup shown in figure **??**.
All coordinates of the sensors are given in the left-handed coordinate system described above, in vehicle coordinates with respect to its ground-level center point, so as to be on the outer surface of the vehicle chassis.
To each camera has been associated an automatic instance and semantic segmentation annotator provided by the simulator.
Thus, each frame capture by each sensor comes directly with its set of labels.
As such, the following cameras have been attached to the Tesla model 3 simulated vehicle:

- front camera
  - $FOV_x = 90$
  - placement: (x=-0.325, z=1.65, y=-0.06)
- long range front camera
  - $FOV_x = 50$
  - placement: (x=-0.325, z=1.65, y=-0.06)
- front left side camera
  - $FOV_x = 90$
  - placement: (x=-0.36, z=1.12, y=-1.1,yaw=-23.62)
- front right side camera
  - $FOV_x = 90$
  - placement: (x=-0.36, z=1.12, y=1.1,yaw=23.62)
- rear left side camera
  - $FOV_x = 75$
  - placement: (x=1.29, z=1., y=-1.1,yaw=-(180-70.2))
- rear right side camera

  - $FOV_x = 75$
  - placement: (x=1.29, z=1., y=1.1,yaw=(180-70.2))
- rear camera
  - $FOV_x = 133$
  - placement: (x=-2.4, z=1.,yaw=180)

The vehicle has been set in autopilot mode, selecting the deterministic behavior provided by the CARLA simulator.
This mode utilizes Unreal engine to determine the position of obstacles, road landmarks and the optimal way to move around in the environment.
The simulation for each scenario has been run capturing synchronized frames from all cameras every 0.02 seconds.
CARLA 0.9.13 has been used to run the simulation.
An example of the multi view camera images captured by the setup is shown in figure 3.3

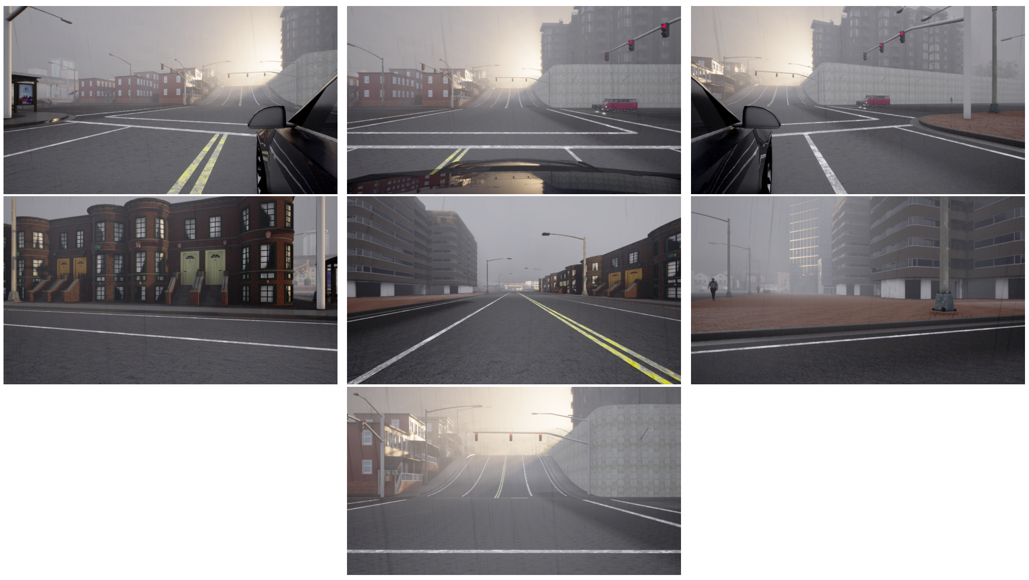


Figure 3.3: Multi-view camera frame: front-left (top-left), front (top-center), front-right(top-right), rear-left (mid-left), rear (center), rear-right (mid-right), long range(bottom)

3.2.4 Synthetic images and ground truth formats

The synthetic images generated by the camera setup previously described are formatted according to the RGB convention (an alternative is BGR).

The semantic segmentation masks are encoded within RGB $H \times W$ images, by encoding at each pixel of the R channel of spatial size $H \times W$ the semantic class associated with each pixel. The G and B channels are unused and set to the pixel value 0.

The instance segmentation masks are encoded within RGB $H \times W$ images, by encoding at each pixel of the R channel of spatial size $H \times W$ the semantic class associated with each pixel. The **G and B channels are utilized** to encode the instance id.

An image sample with its associated ground truth instance and segmentation masks is shown in the figure 3.4

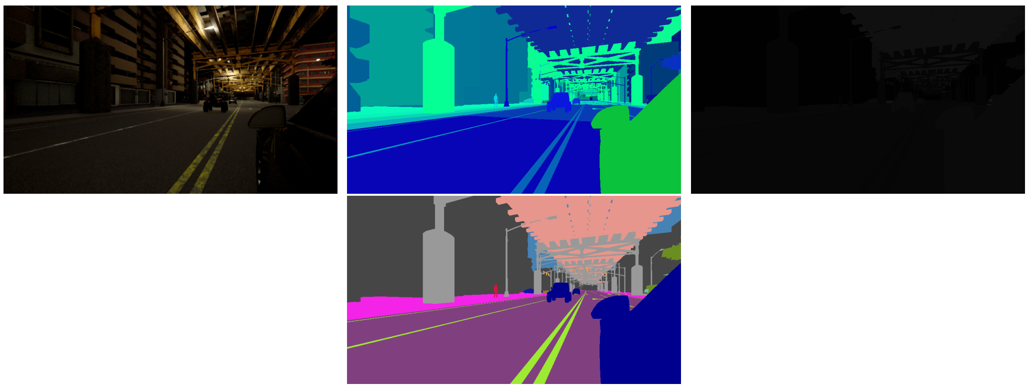


Figure 3.4: Left camera frame: image (left), instance mask(center), semantic mask(right), panoptic mask (bottom)

3.3 Synthetic-to-Real dataset

With the obtained instance masks by means of the described simulations, it has been necessary define a procedure to match the default semantic classes provided by the CARLA simulator to those of the selected real world datasets: Cityscapes[60] and BDD100k[62].
A common subset does not exist, however one can be generated by aggregating together the most closely matching semantic classes, separately for each dataset.
Hence a common subset of labels for thing and stuff semantic classes has been defined.
Then, Cityscapes, BDD100K and the generated synthetic datasets have been relabeled according to such defined subset.
To achieve this, an automated script simply modifies the semantic id of each ground truth of each dataset to the associated one that is within the common label subset. The results are saved and defined as label-matched datasets.

However, the DA-PanopticFPN deep learning model that is described in the Model design chapter requires a specific label format in order to be trained.
Firstly, all instances and semantic classes have to be represented in COCO-Detection and COCO-stuff format[53], respectively. In essence, this requires to transform the instance and semantic segmentation masks to a JSON file that contains the COCO-Detection representation for a given dataset:

- its set of images, each associated with a unique id and its file path
- the set of semantic categories considered in the dataset, with the metadata for each
- the mask annotations, each with:
  - the associated image id
  - the annotation id
  - the semantic category
  - area covered by the mask
  - the segmentation represented by RLE encoding or by its bounding polygon
  - bounding box (valid also for stuff categories)
  - the iscrowd boolean flag that can only be true for thing classes, if the considered instance is deemed to be unrecognizable due to too much overlap with other elements in the scenes, or it is too occluded
- the semantic categories considered in the dataset

After this procedure is completed, for all datasets, the panoptic masks can be generated according to the COCO-Panoptic format[20, 53].
Once again, this convention requires the generation of a JSON file which contains the details of the COCO-Panoptic datasets.
However, besides the images associated to the panoptic masks, this format expects also the creation of panoptic masks in image format, hence the RLE encoding is not sufficient anymore.
The COCO-Panoptic protocol requires a JSON file containing:

- the set of images, each associated with a unique id and its file path
- the segment annotations(an abstraction that is used represent a mask by its semantic category or its instance id), each with:
  - the associated image id
  - the panoptic mask annotation file path

  - segment information, one for each entity in the scene, regardless of thing or stuff class:
    * segment id, required to retrieve the entity within the panoptic mask image file (.png masks are required to avoid lossy compression of other image formats)
    * the semantic category
    * area covered by the mask
    * bounding box (valid also for stuff categories)
    * the iscrowd boolean flag that can only be true for thing classes, if the considered instance is deemed to be unrecognizable due to too much overlap with other elements in the scenes, or it is too occluded

- the set of semantic categories considered in the dataset, with the metadata for each

Furthermore, an additional conversion is required to move from the COCO-Panoptic format to one that is usable by the PanopticFPN model.
It entails the conversion as described in the code provided at the github repository of the Detectron2 framework[150].
In practice, it takes the COCO-Panoptic annotations and separates them again into an instance segmentation COCO-Detection JSON and a semantic segmentation dataset, in which all things categories are marked with the semantic category= 0. The stuff semantic categories are instead re-labeled with labels $sem\_category \geq 1$. A special label 255 is associated to pixels that can be ignored during training and evaluation.

To implement this conversion pipeline for the **synthetic dataset**, customized implementations of the following scripts have been utilized:

1. CARLA label re-assignment to the label-matched subset
2. conversion from instance mask (for both things and stuff) images to COCO-Detection format, in order to obtain a result in the aforementioned COCO-Detection convention
3. COCO-Detection $\rightarrow$ COCO-Panoptic, with a customized implementation of [151] that allows to obtain a dataset in the presented panoptic format.
4. COCO-Panoptic to customized PanopticFPN datasets in COCO-Instance and COCO-stuff format[150]

A similar conversion pipeline has been used to convert **Cityscapes** from its standard format to the PanopticFPN specific dataset format:

1. Cityscapes label re-assignment to the label-matched subset
2. conversion from instance mask (for both things and stuff) images to COCO-Detection format
3. COCO-Detection → COCO-Panoptic, with a customized implementation of the conversion script provided by the authors of Cityscapes[152]t.
4. COCO-Panoptic to customized PanopticFPN datasets in COCO-Instance and COCO-stuff format[150]

Finally, the conversion pipeline is applied also to **BDD100k**, in order to convert it to the PanopticFPN specific dataset format:

1. BDD100k label re-assignment to the label-matched subset
2. conversion from instance mask (for both things and stuff) images to COCO-Detection format
3. COCO-Detection → COCO-Panoptic, with a customized implementation of the conversion script provided by the authors of BDD100k [153].
4. COCO-Panoptic to customized PanopticFPN datasets in COCO-Instance and COCO-stuff format[150]

3.3.1 Multi dataset label matching

Following, the label conversion from dataset specific to label-matched version is presented, for each of the datasets considered in this dissertation.
Before proceeding the **label-matched subset** is presented: The **thing classes** that are considered are:

- car
- pedestrian

The stuff classes that are considered are:

- road
- sidewalk
- building
- wall
- fence
- pole
- traffic light
- traffic sign
- vegetation
- terrain
- sky

Synthetic CARLA

The mapping in table **??** is utilized to move from dataset specific, shown in the column **original** labels to the label-matched version (column **label matched**), note that these are not these labels are the ones used before each is converted to the PanopticFPN specific format.

Class	**Original**	**Label matched**
road	7	0
roadline	6	0
sidewalk	8	1
building	1	2
wall	11	3
fence	2	4
pole	5	5
traffic light	18	6
traffic sign	12	7
vegetation	9	8
terrain	22	9
sky	13	10
person	4	11
car	10	13
unlabeled	0	255
other	3	255
ground	14	255
bridge	15	255
rail track	16	255
guard rail	17	255
static	19	255
dynamic	20	255
water	21	255

Table 3.1: Synthetic dataset mapping from default labels to label-matched version

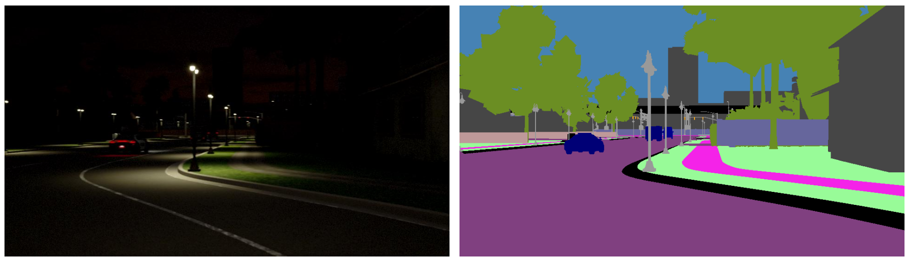


Figure 3.5: Synthetic dataset sample with its label-matched panoptic mask

Cityscapes

The mapping in table **??** is utilized to move from dataset specific, shown in the column **original** labels to the label-matched version (column **label matched**), note that these are not these labels are the ones used before each is converted to the PanopticFPN specific format.

Class	**Original**	**Label matched**
license plate	-1	-1
road	7	0
sidewalk	8	1
building	11	2
wall	12	3
fence	13	4
pole	17	5
traffic light	19	6
traffic sign	20	7
vegetation	21	8
terrain	22	9
sky	23	10
person	24	11
rider	25	11
car	26	13
truck	27	13
bus	28	13
caravan	29	13
train	31	13
motorcycle	32	13
bicycle	33	13
unlabeled	0	255
ego vehicle	1	255
rectification border	2	255
out of roi	3	255
static	4	255
dynamic	5	255
ground	6	255
parking	9	255
rail track	10	255
guard rail	14	255
bridge	15	255
tunnel	16	255
polegroup	18	255
trailer	30	255

Table 3.2: Cityscapes mapping from default labels to label-matched version

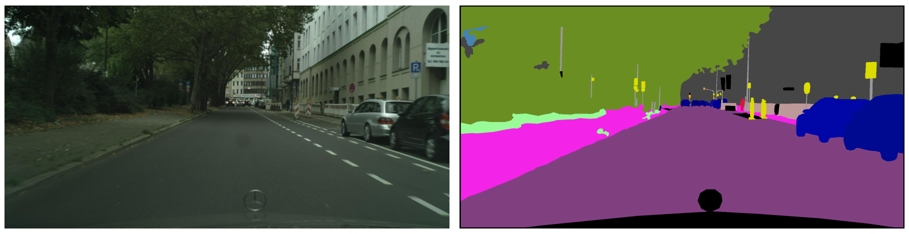


Figure 3.6: Cityscapes sample with its label-matched panoptic mask

BDD100k

The mapping in table **??** is utilized to move from dataset specific, shown in the column **original** labels to the label-matched version (column **label matched**), note that these are not these labels are the ones used before each is converted to the PanopticFPN specific format.

Class	**Original**	**Label matched**
road	7	0
sidewalk	8	1
building	10	2
wall	15	3
fence	11	4
pole	20	5
traffic light	25	6
traffic sign	26	7
vegetation	29	8
terrain	28	9
sky	30	10
person	31	11
rider	32	11
car	35	13
bicycle	33	13
bus	34	13
caravan	36	13
motorcycle	37	13
train	39	13
truck	40	13
unlabeled	0	255
dynamic	1	255
ego vehicle	2	255
ground	3	255
static	4	255
parking	5	255
rail track	6	255
bridge	9	255
garage	12	255
guard rail	13	255
tunnel	14	255
banner	16	255
billboard	17	255
lane divider	18	255
parking sign	19	255
polegroup	21	255
street light	22	255
traffic cone	23	255
traffic device	24	255
traffic sign frame	27	255
trailer	38	255

Table 3.3: BDD100k mapping from default labels to label-matched version

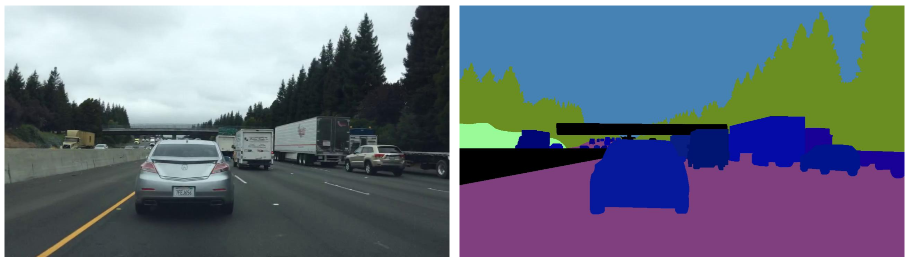


Figure 3.7: BDD100k sample with its label-matched panoptic mask

Chapter 4

Model design

4.1 Introduction

This introductory section serves the purpose of providing a formal description of the design of the Domain Adaptive PanopticFPN neural network architecture, a modification built in this work from the template PanopticFPN model[65].
The objective of this work is the evaluate the improvement in panoptic segmentation performance that a domain adaptive model obtains compared to a baseline trained on synthetic driving data and tested on real world driving data.
Hence, considering as baseline the PanopticFPN[65] architecture, a modified version named DA-PanopticFPN which employs an adversarial self-supervised domain adaptation module has been developed in this dissertation.
The former has been selected as it adopts the minimal set of changes to develop a panoptic segmentation model, favoring a simplistic and minimal design as noted by the authors[65].
It adopts a Mask R-CNN[68] instance segmentation architecture, which consists of an FPN[107] backbone created from a ResNet50[93], and a parallel semantic segmentation branch which is attached to the termination of the FPN backbone.
As such, the effectiveness of domain adaptation can be easily evaluated by avoiding possibly unintended complex interactions that can appear in the case of more convoluted architectures such as PanopticDeepLab[21] or EfficientPS[22].
Extensions to domain adaptive versions of the above architecture is of interest for future work.
The details that are specific to such model will be presented shortly after, the following formalization.

At its core, a neural network can be understood as a set of functions $\mathcal{F}_i(\cdot) i \in \{1, ..., N\}$ each parameterized by zero or more optimizable parameters, which if

present are denoted as weights $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$ and biases $b_i \in \mathbb{R}^{d_{i-1} \times d_i}$, and non-optimizable ones, the hyperparameters.
The ordering with which such functions are applied to the inputs $X \in \mathbb{R}^{d_0 \times d_1}$ implicitly defines a composite function

$$\mathcal{F}(\cdot) = \mathcal{F}_n \circ \mathcal{F}_{n-1} \circ ... \circ \mathcal{F}_1(\cdot)$$

which represents the network.
Hence the composite function $\mathcal{F}_\theta$, which depends on the parameter set $\theta = \{W_1, ..., W_N, b1, ..., b_N\}$, maps inputs X to outputs Y as:

$$\mathcal{F}_{W_1,...,W_N,b1,...,b_N} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_n}$$

The objective of such complexly built non-linear composite function, is to approximate by means of its optimizable parameters, the mapping between inputs X and the ground truths Y , which are sampled as M tuples $(x_m, y_m)m \in \{1, ..., n\_samples\}$ from the non-directly observable data distribution $P(X, Y)$.

The design of neural networks follows the common guiding principle to employ one or more backbone feature extractors(the latter in the case of multi-modal data or of an ensemble of models), to automatically learn relevant patterns from the input data, and one or more task-specific branches which are attached in an arbitrary manner to the backbone/s outputs.
In the setting of multi-task learning, it is common practice to define a common backbone feature extractor which is tasked to automatically learn by backpropagation the representations which are desirable for all the selected tasks, while the task-specific network heads (also called branches) learn specialized representations that are only useful on their own end.
In such case, the previously mentioned composite function representation of the neural network can be updated to accomodate this general setting.
Let the following definition be given:

- $\theta_\mathcal{F}$ be the learnable parameters of the backbone
- $\theta_{\mathcal{G}_i}$ be the learnable parameters of the i-th task-specific branch

Furthermore let:

- $\mathcal{F}(\cdot; \theta_\mathcal{F})$ be the backbone model
- $\mathcal{G}_i(\cdot; \theta_{\mathcal{G}_i})$ $i \in \{1, ..., n\_tasks\}$ be the task-specific heads

The neural network outputs can be defined, with the associated dependencies between computations:

$$Z = \mathcal{F}(X; \theta_\mathcal{F})$$
$$Y_i = \mathcal{G}_i(\mathcal{F}(X; \theta_\mathcal{F}); \theta_{\mathcal{G}_i})$$
$$i \in \{1, ..., n\_tasks\}$$

In which Z is the intermediate output of the shared backbone model, which is fed to each of the $\mathcal{G}_i$ task-specific heads to generate the related predictions.

4.2 Backbone

As the backbone of the DA-PanopticFPN architecture is an FPN model, the latter requires the definition of its own feature extractor which is employed to build its main components: a bottom-up pathway and a top-down pathway.
The former is defined by a selection of the output feature maps generated by the chosen FPN feature extractor. The top-down pathway instead is built according to the way the bottom-up pathway is defined.

4.2.1 ResNet feature extractor

ResNet is a family of deep learning models of varying depth, which are designed following the principles of the [**residual learning**] framework.
In this work, the **ResNet50** variant has been employed, for which the suffix number stands for the depth of the network, in terms of layers.
The layers which define the ResNet are grouped into **5 stages** named $conv1\_x$, $conv2\_x$, $conv3\_x$,
each of varying number of layers.
ResNets with varying network depth maintain the same stage-based structure, and add more layers to each of the aforementioned stages to reach the desired depth.
Although such models are distinguished by their depth, their basic building blocks are the **bottlenecks**, which are represented by groups of convolutional layers.
Each of the bottleneck blocks is implemented following the residual learning design principles, of which the details are provided shortly.

The basic building block of the ResNet architecture **residual learning** is the residual block.
It allows to shift the learning problem tackled by one or more subsets of inner convolutional layers from approximating a generic function $\mathcal{H}(X)$ of its inputs X , to the learning, by each convolutional layer, of a residual function $\mathcal{F}(X) = \mathcal{H}(X) - X$. Such formulation reduces the complexity of the learning problem at each network component which adopts the residual learning paradigm, as the function to be learned $\mathcal{F}$ is now the residual mapping with respect to the layer's inputs and not an arbitrarily complex function. Thus the operation performed by the convolutional layers becomes $\mathcal{H}(X) = \mathcal{F}(X) + X$[93]. Such modified function $\mathcal{H}$ represents the operation which is computed in a residual block [93], with $\mathcal{F}$ representing the residual function learned by one ore more layers in the residual block, while X is the identity reference mapping which is summed to $\mathcal{F}$. In practice, the residual

block which represents $\mathcal{H}(X) = \mathcal{F}(X) + X$[93] is implemented as is seen in figure 4.1.

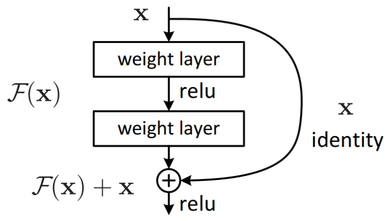

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4.1: Residual function of a Resnet architecture, implemented as a neural network computation graph [93]

Hence, the residual block implemented as the element-wise summation of the identity mapping and a (convolution→ batch normalization → activation) repeated twice becomes the base component of the ResNet architecture.
In practice to reduce computational complexity of the 50 to 200 layers deep ResNets, the **bottleneck module**[93] shown in figure **??** has been designed.
As can be seen in figure 4.2, The fundamental difference between the **residual blocks** and the **bottleneck version**, is on the use by the latter, of 1x1 convolutions as a dimensionality reduction technique, to reduce the number of convolutional filters, before the 3x3 expensive convolutions are applied. The result is then rescaled to the original dimension after the expensive operations have been computed, by another downstream 1x1 convolution layer, attached before the element-wise summation. Each of such convolution operations is always followed by batch normalization and ReLU activation.
Each **bottleneck** follows the same structure, except the ones used at the start of the conv2_x, conv3_x, conv4_x, conv5_x groups shown in the table 4.3that displays the resnet architecture variations.
As each bottleneck is fundamentally composed by a set of convolutional layers and a parallel identity mapping, these bottlenecks have to handle the increase in dimensionality of such convolutional layers,in terms of number of channels of its convolutional filters, by passing the identity mappings to a 1x1 convolutional layer with stride of 2, followed by batch normalization in order to obtain matching tensor dimensionalities for the following element-wise summation4.2.

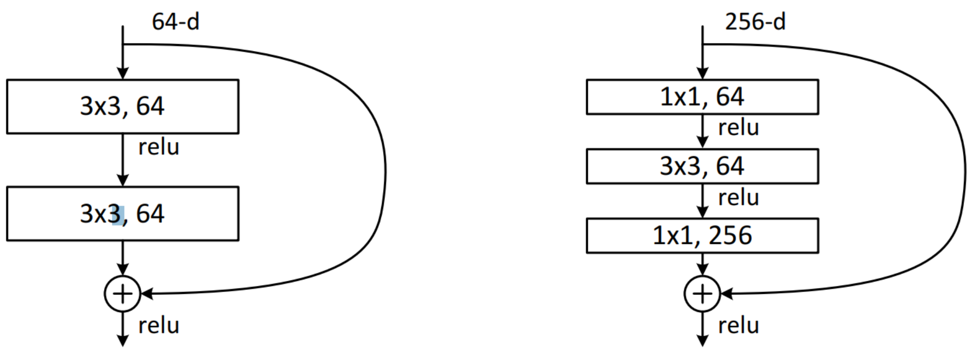
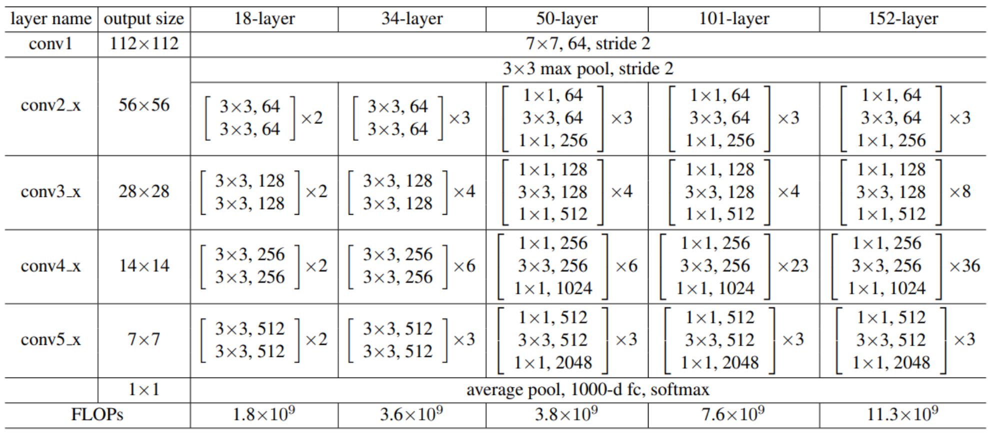


Figure 4.2: Side by side comparison of the residual learning block and its more efficient Bottleneck version [93].

As such, the bottleneck design has been employed in the original paper by He et al. [93], to build ResNet18,ResNet34,ResNet50,ResNet101 and ResNet152.

As the ResNet architectures are a fundamental building block of the overall neural network architecture employed in this work, their structure is reported in the table 4.3.
The subdivision of all bottleneck blocks within the ResNet in **five groups of operations**, is shown in the table with their associated names: **conv1, conv2__x, conv3__x, conv4__x, conv5__x**. As can be seen, depending on the network depth, each of the groups contains a varying number of bottleneck blocks.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.3: Side by side comparison of the bottleneck based resnet models: ResNet18,ResNet34,ResNet50,ResNet101, ResNet152 [93].

4.2.2 FPN

The **FPN** architecture is constituted by four main components: **lateral connections**, **top-down connections**, a **bottom-up** and a **top-down** pathway.
The **bottom-up** pathway consists of a backbone architecture such as ResNet, ResNext or EfficientNet whose objective is the extraction of features from images, without its final fully connected layer of the standard template model so as to obtain a fully convolutional backbone.
Input images are forwarded in such model, as it is drawn, from the bottom which represents the input layer, to the top, which represents the last layer of the network, *hence the naming of* **bottom-up pathway**.
Of all feature maps generated a forward pass of the input images by the backbone layers, a subset is selected to develop the rest of the **FPN architecture**. As is cited in [107], a natural choice is to select the *semantically strong output features* with the **most different feature semantics and scales**, so as to maintain the **largest variety of learned multi-scale features**: from the high resolution features near the input layers, to the coarse object-level features(located at the deepest network layers).
As is common to employ ResNet backbones for FPN models, each of the drawn levels of the bottom-up pathway, would represent in such case the network stage output features of the $conv3\_x, conv4\_x, conv5\_x4.3$, with the extension to additional stages being the straightforward repetition of the drawn model.
As mentioned above, such choice is convenient because each of these feature outputs are separated by many convolutional layers, hence they provide the strongest feature representation at each of the selected network stages, while also being uncorrelated due to the distance that separates them, in terms of number of in-between layers.

As the output feature levels of the bottom-up pathway are defined, the **top-down pathway** can be shaped based on the specifications of the FPN model.
Accordingly, to each of the selected output feature maps of the bottom-up pathway e.g. $conv3\_x, conv4\_x, conv5\_x$ for a ResNet, it is associated a corresponding merged feature map e.g. p_3, p_4, p_5 respectively, in the top-down pathway.
In order to generate such merged maps, the FPN architecture utilizes:

- **top-down connections**: prepares the top-down input required to generate the merged feature map p_i, by upsampling the coarser merged feature map at level p_{i+1} by a factor of 2. Then the result is convolved with 3x3 filters. The output of the top-down connection operation is already set with the predefined number of channels $d = 256$.

- **lateral connections**: 1x1 convolution to reduce channel dimension to a common number of channels $d = 256$

The use of lateral and top-down connections generates two tensor outputs of equal shapes, that are element-wise summed to create the top-down merged feature maps depicted in the right side of the FPN model shown in the figure **??**. The only exception is for the top-down feature map at the coarsest level e.g. p_5 in this example, which is the result of a single lateral connection starting at the corresponding level $conv5\_x$ of the bottom-up pathway.
A further level in the top-down feature pyramid is p_6, which is the result of stride 2 subsampling of the p_5 top-down feature maps.

To summarize, FPN does not explicitly require a specific feature extractor e.g. ResNet50,ResNet101, EfficientNet-B0.
Based on its backbone, a set of bottom-up feature maps that it generates are selected and named as $C_i i \in \{1, ..., N\}$.
Given the selected bottom-up feature maps, a corresponding number of top-down merged features are generated with names $P_i i \in \{1, ..., N\}$.
Based on the level i at which they are generated, they represent the semantically richer version of the feature map of the bottom-up pathway at the same level i. As an example, if the top-down feature map P_2 is the merged feature map which contains multi-scale rich semantics from P_3 and C_2, P_2 represents a semantically rich version of C_2, which only captures high details semantics, without object-level ones [91].

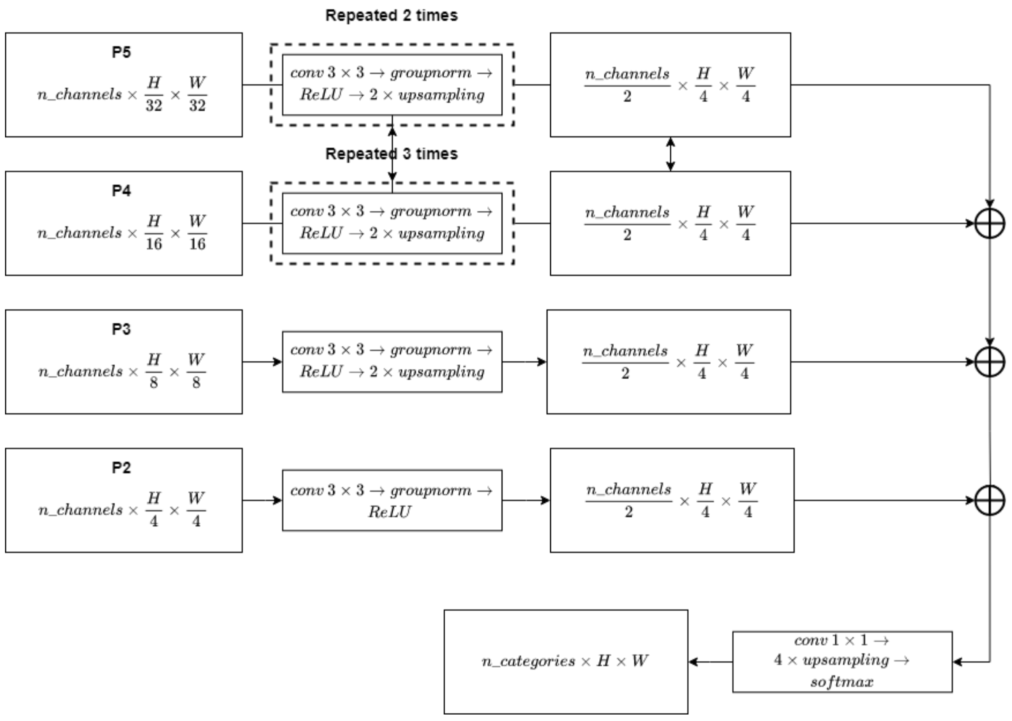
The standard FPN model imposes only two architectural constraints, based on the chosen bottom-up feature maps :

- the merged feature maps generated at the i-th level of the top-down pathway, of shape $n\_channels_i \times H_i \times W_i$ must all have the same number of channels, while the spatial shape $H_i \times W_i$ are adjusted by in-model upsampling.

- the feature maps of the bottom-up pathway must be each downscaled by a factor of 2 with respect to its preceding bottom-up level feature map.
  Thus, if the selected bottom-up feature maps are $\{C_2, C_3, C_4, C_5\}$, the subscript number indicates both the scale factor of the spatial shape $H_i \times W_i$ of the feature maps at that level, respectively $\{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$ of the input size, as well as the level at which the bottom-up feature map is located in the bottom-up feature hierarchy.
  Hence, C_4 are feature maps whose spatial shape is $\frac{H}{16} \times \frac{W}{16}$, with respect to the $H \times W$ input image

4.3 Semantic segmentation head

As panoptic segmentation requires both instance and semantic segmentation, the minimal design of the PanopticFPN architecture[65] implements the latter functionality by designing a task-specific branch.
Their proposed design is implemented in the DA-PanopticFPN model exactly as described in the original PanopticFPN architecture.
The neural network that represents the semantic segmentation branch is shown in figure 4.4.



Repeated 2 times
P5
$n\_channels \times \frac{H}{32} \times \frac{W}{32}$
conv 3 × 3 → groupnorm → ReLU → 2 × upsampling
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
Repeated 3 times
P4
$n\_channels \times \frac{H}{16} \times \frac{W}{16}$
conv 3 × 3 → groupnorm → ReLU → 2 × upsampling
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
P3
$n\_channels \times \frac{H}{8} \times \frac{W}{8}$
conv 3 × 3 → groupnorm → ReLU → 2 × upsampling
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
P2
$n\_channels \times \frac{H}{4} \times \frac{W}{4}$
conv 3 × 3 → groupnorm → ReLU
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
$n\_categories \times H \times W$
conv 1 × 1 → 4 × upsampling → softmax

Figure 4.4: Semantic segmentation branch

Such branch takes in as inputs the top-down feature maps $\{P_5, P_4, P_3, P_2\}$ (listed in top-down order), and applies a block of operations to each, in order to obtain feature maps of equal number of channels and same $\frac{1}{4}$ spatial scale with respect to the input shape.
This is done in order to compute the element-wise sum, which is then bilinearly upsampled by a factor of 4 to obtain an output feature map $n\_categories \times H \times W$ that matches the original input spatial size $H \times W$. Softmax is applied to such output in order to obtain the pixel-wise classification of each element of the original image.

The basic block which connects the feature maps of the FPN top-down pathway to the feature maps of the semantic segmentation branch consists of the following operations applied in order: 3x3 convolution, group normalization[154] (the principle is the same as batch normalization, but channels are separated as groups in order to compute more robust statistics even with smaller batch size), ReLU, bilinear upsampling by a factor of 2.
This block is applied repeatedly in the case of P_5 and P_4 in order to obtain at all levels outputs of matching size. More specifically, in the case of P_5 and P_4, it is repeated respectively three and two times.
Only one block is used for the outward connection of P_3.
The outward connection from P_2 is stripped of the $2\times$ bilinear upsampling as it is already of the desired spatial dimension.

4.4 Mask R-CNN head

The instance segmentation pipeline is implemented exactly as described in the original Mask R-CNN[68] implementation, in the case of an FPN backbone.
The Mask-RCNN pipeline consists of four main stages, attached in series in the provided order:

1. backbone feature extractor: the feature pyramid network in this case
2. region proposal network (RPN) module, whose objective is to generate object proposals
3. ROI align module
4. parallel instance segmentation, bounding box regression and object classification on each ROI provided by the ROI align module

In order to generate the bounding-box, object classification and instance mask predictions, the Mask R-CNN architecture requires object proposals which are generated by means of its RPN module.
Following this brief description, the RPN and the parallel branches of the Mask R-CNN are described.

4.4.1 Region Proposal Network

As the Mask R-CNN operates in an end-to-end manner, the object proposal are generated within the architecture, by means of the RPN module.
As the objective of the RPN is to propose possible bounding boxes, it achieves

this end-to-end by employing the concept of **anchors**. Anchors are defined as the set of location-independent bounding boxes obtained by computing all possible combinations of a chosen set of $object\_scales = \{0.512\}$ and $aspect\_ratios = \{32, 64, 128, 256, 512\}$, resulting in $num\_anchors = \mathbf{card}(object\_scales) \times \mathbf{card}(aspect\_ratios)$ anchors.
These are computed without any prior information on the content of the images, resulting in the **translation invariance property**[112] of such anchors.
RPN associates such anchor set of all possible combinations of scales and aspect ratios to each possible super-pixel of the feature maps which are fed as its input.
This implies that if a feature map is of size $H \times W$, the RPN considers $num\_anchors \times H \times W$ anchors.
All such anchors become localized with respect to their associated feature map location.
As such, the RPN adopts two parallel branches for objectness score and box regression coordinates prediction, for each of such anchors.
The former predicts whether a region of interest contains a foreground object or is background, while the latter predicts a 4-tuple whose elements represent the bounding box in transformed coordinates, that allows to refine the coordinates of the given anchor to those of its associated ground truth box: (t_x, t_y, t_w, t_h)[105].
These values are defined as follows, with x, y defined as the box's center coordinates and w, h its width and height[106, 112]:

$$t_x = \frac{(x - x_a)}{w_a}$$
$$t_y = \frac{(y - y_a)}{h_a}$$
$$t_w = \log(\frac{w}{w_a}$$
$$t_h = \log(\frac{h}{h_a})$$

The variables x, x_a, x^* follow the convention that defines them as predicted box coordinate, anchor box coordinate and ground truth coordinate respectively. This also holds for y, w, h.
By inverting the transformations, these regression targets are used to obtain the bounding boxes in image coordinates.
. Hence, this also implies that anchors which become associated to spatial locations in the feature map, can be tracked back to the input image.
Hence, with the above definitions, it can be noticed that the outputs of the RPN are:

- the objectness score tensor of shape $n\_anchors \times 2$ to represent the probability of object/non object as binary classification

- the bounding box regression tensor of shape $n\_anchors \times 4$, which allows the translation of anchors to the predicted object location

Thus, the RPN can defined by:

1. a convolutional layer with filters of shape 3x3, stride 1 and padding 1. The number of input channels is equal to the number of output channels, which is set to 256
2. a objectness prediction branch, attached to 1). It consists of a convolutional layer with 1x1 filters, stride 1, 0 padding. Its output is of shape $n\_anchors \times 2$ for object/non-object classification.
3. a bounding-box regression branch, attached to 1). It consists of a convolutional layer with 1x1 filters, stride 1, 0 padding. Its output is of shape $n\_anchors \times 4$ for anchor to ground truth bounding box regression.

One final remark is necessary regarding the number of object proposals that are generated by the RPN.
As it predicts object proposals for feature maps from all levels $P_k \in \{2, ..., 6\}$ of the top-down pathway of the FPN backbone, the number of proposals can become exceedingly large e.g. for an input image of shape $H = 768$, $W = 1152$, the RPN predicts 220968 proposals.
Thus, non-maximum suppression is used to filter out too small proposals, the ones with low confidence scores or the ones with too high overlap. Each of these boundaries can be manually tuned by pre-defined thresholds.

4.4.2 ROI align

As object proposal obtained by the RPN module entail a wide variety of scales and aspect ratios, the ROI align layer allows to overcome such issues for the downstream tensor computations.
It considers the object proposal bounding box shape (of variable spatial size $H_b \times W_b$), had been obtained by the RPN, to output a bilinearly interpolated feature map of an a priori selected fixed spatial shape.
The interpolation is obtained from a feature map at the **k-th** level of the FPN top-down pathway, whose scale matches the object proposal shape according to the formula[107]:

$$k = floor(k_0 + \log_2 \frac{\sqrt{H_b \times W_b}}{224}$$

In which $k_0 = 4$[107].
As the FPN feature map selection is based on the architecture definition, the scales

are known a priori.
As in this work the FPN output feature maps are $\{P_2, P_3, P_4, P_5, P_6\}$, the respective scales are known to be $\{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$ of the original input shape.
Thus, ROI align can properly be employed to obtain fixed size feature maps downstream of the varying size object proposals obtained by the RPN module.
In the DA-PanopticFPN, ROI align has been set to generate:

- 7×7 feature maps that are fed to the bounding box and object classification branch
- 14×14 feature maps that are fed to the instance segmentation branch

4.4.3 Bounding Box regression and Object classification branch

Apart for the novel instance segmentation branch and the ROI align layer, the Mask R-CNN architecture adopts the same RPN, bounding box regression and object classification branches as those of the Faster R-CNN model[112].
As such, the only variation consists in the fact that upstream to the two aforementioned branches there is a ROI align layer which generates output feature maps of fixed spatial shape $256 \times 7 \times 7$ from the variable size object proposals.
Hence, starting from such ROI align output feature maps, a subnetwork common to the two branches receives as inputs those as flattened tensors, whose shape then becomes $256 \cdot 7 \cdot 7 = 12544$. It consists of:

1. linear layer 12544×1024, as input and output shape respectively
2. ReLU activation
3. linear layer 1024×1024, as input and output shape respectively
4. ReLU activation

The last layer activation outputs are then fed to the two parallel branches, which are implemented as follows:

- classification layer, with input shape 1024 and output of size $num\_classes + 1$ to account for the background category.
- bounding box proposal regression layer, with input shape is 1024 and the output shape is $n\_classes \cdot 4$.
  This is because to achieve the parallel branch design, a (t_x, t_y, t_w, t_h)4-tuple refinement of the coordinates of each proposal is predicted for the $n\_classes$, excluding the background one [106].
  Then, only the one associated with the predicted class is considered.

Furthermore, downstream to the bounding box detections, a non-maximum suppression procedure is employed to filter out overlapping boxes, too small regions or the ones for which objects have been predicted with low confidence score by the classification branch.

On a final note, it is necessary to remark that although the RPN already performs object/non-object classification and bounding box regression to obtain object proposals, these two layers are required in order to refine such proposals with respect to the predicted object classes, as the RPN proposal represent only rough estimates of the object locations.
The two-stage implementation of the object detector allows it to obtain remarkable detection performance, although at the cost of more computational resources. One-stage detectors[45] avoid such complexity but partially sacrifice quality of the detections. However, implementation of the latter is out of the scope of this work.

4.4.4 Instance segmentation branch

The instance segmentation branch follows the design proposed in Mask R-CNN. Downstream to the ROI align which outputs in this case fixed shape $256 \times 14 \times 14$ feature maps, the branch is implemented with:

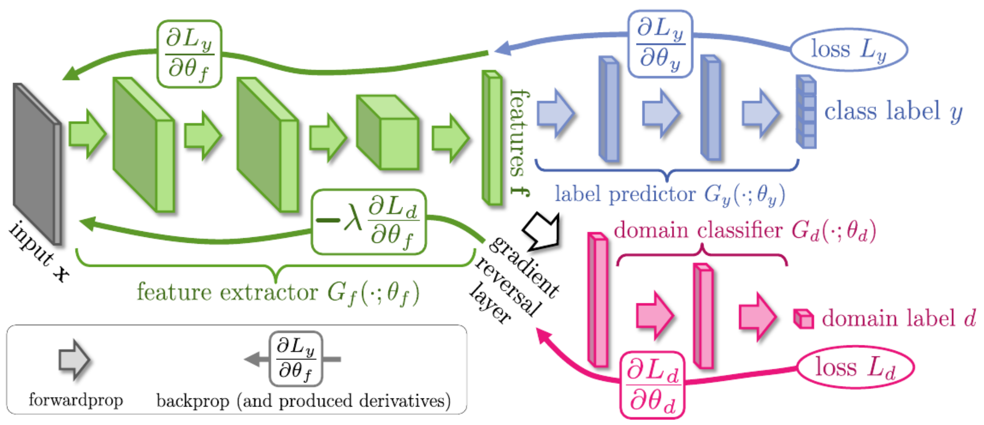
1. 4 blocks attached in series, each implemented by means of a convolutional layer with kernels of shape 3×3, $in\_channels = 256$, $out\_channels = 256$, with stride 1 and padding 1 (in order to maintain the same spatial shape across the 4 blocks), followed by a ReLU activation
2. Transposed convolution, with $in\_channels = 256$, $out\_channels = 256$, filters of shape 2×2 and stride 2
3. ReLU activation
4. convolutional layer with $in\_channels = 256$, $out\_channels = n\_classes$, filters of shape 1×1 as a means to reduce the channel dimension to the desired number of classes.

It is necessary to note that the mask predictions are generated without competition among the classes e.g. given the input feature maps associated with an input sample to the network, *n_classes* masks are generated.
Then, the mask associated with the predicted class for the given ROI is selected as final prediction of the instance segmentation branch.
The output mask predictions are each of spatial shape 28×28. At inference time, these are resized to the shape of the associated bounding box prediction obtained from the parallel bounding box regression branch.

4.5 Domain Adaptation branches

4.5.1 Adversarial Self-supervised Domain Adaptation

The work of [140] is the first to introduce the adversarial methods to tackle the problem of domain shift between source and target samples. It is based on the assumption that at training time a large amount of *unlabeled* target data is available. The adversarial pretext task consists in a discriminator trained to predict whether the feature representation produced by the feature extractor is associated with a source or a target example. To this end, they introduce a label called *domain label*. When a point has been sampled from the source distribution $d_i = 0$, otherwise $d_i = 1$ when it has been sampled from the target distribution.
When a sample $\mathbf{x}$ is fed to the deep architecture, depicted in Figure 4.5, the feature extractor network produces a lower-dimensional embedding $\mathbf{f} = G_f(\mathbf{x}, \theta_f)$. Then, the resulting embedding is sent simultaneously to two tasks: the main classification task $G_y(\cdot, \theta_y)$, which predicts the class of the input element, and the domain adversarial pretext task $G_d(\cdot, \theta_d)$, which determines whether the sample belongs to the source or target domain.
The objective to minimize is composed by the contribution of both tasks, which are trained at the same time, ensuring that the learned features $\mathbf{f}$ retain their discriminative potential useful for the classification task, while being domain invariant. In other words, we want the distributions $S(\mathbf{f} = \{G_f(\mathbf{x}, \theta_f) | \mathbf{x} \sim S(\mathbf{x})\})$ and $T(\mathbf{f} = \{G_f(\mathbf{x}, \theta_f) | \mathbf{x} \sim T(\mathbf{x})\})$ to be as similar as possible.



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input $\mathbf{x}$
gradient reversal layer
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Figure 4.5: Self-supervised adversarial domain adaptation through gradient reversal. Credits: [140]

More formally, the authors define the optimization model starting from the following functional:

$$E(\theta_f, \theta_y, \theta_d) = \sum_{i\in\{1,..N\} d_i=0} L_y(G_y(G_f(\mathbf{x_i}, \theta_f)); y_i) - \lambda \sum_{i\in\{1,..N\}} L_d(G_d(G_f(\mathbf{x_i}, \theta_f)); d_i) \tag{4.1}$$

where L_y is the loss for class label prediction, L_d is the loss for domain label prediction,$i \in \{1, ..., n\_samples\}$, y_i is the class label ground truth and d_i is the domain label of the i-th sample. This allows to define the following minimization problem

$$(\hat{\theta_y}, \hat{\theta_f}) = \arg\min_{\theta_y, \theta_f} E(\theta_f, \theta_y, \hat{\theta_d}) \tag{4.2}$$

$$\hat{\theta_d} = \arg\max_{\theta_d} E(\hat{\theta_y}, \hat{\theta_f}, \theta_d) \tag{4.3}$$

At the saddle point resulting from the solution of this optimization problem, the parameters θ_y minimize the loss of the classifier, whereas the parameters θ_d minimize the loss of the domain discriminator.
Interestingly, the parameters θ_f *minimize* the loss of the classifier, while *maximizing* the loss of the discriminator[140].
The modifiable hyperparameter λ balances the effect of the two tasks during the learning of the parameters.

4.5.2 Gradient Reversal Layer

The fundamental component that is implemented in Ganin's self-supervised adversarial domain adaption framework is the **gradient reversal layer**.
The deep model which is defined in [140] is set in a multi-task learning context, in which one task is the main classification problem, while the other is the self-supervised domain classification which enables its proposed domain adaptation framework.
Both branches are attached to a **common backbone** feature extractor.
In practice, the gradient reversal layer is represented by a pseudo function, which is defined by two incompatible equations: one for the forward pass and the other for the backpropagation.

$$1) \mathcal{R}_\lambda(\mathbf{x}) = \mathbf{x}$$
$$2) \frac{\partial \mathcal{R}_\lambda(\mathbf{x})}{\partial \mathbf{x}} = -\lambda \mathbf{I}$$

Following such formulation, the optimization objective becomes:

$$E(\theta_f, \theta_y, \theta_d) = \sum_{i\in\{1,..N\}, d_i=0} L_y(G_y(G_f(\mathbf{x_i}, \theta_f); \theta_y); y_i) + \sum_{i\in\{1,..N\}} L_d(G_d(\mathcal{R}_\lambda(G_f(\mathbf{x_i}, \theta_f)); \theta_d); d_i)$$

4.5.3 Gradient Reversal training schedule

Although λ has been presented as a fixed hyperparameter, the authors of [140] propose an update schedule which gradually modifies the value of λ from 0 to 1 during training, in order to suppress the noisy signal that is caused by the reversed gradients at the early stages of training.
Defining $p = \frac{curr\_iter}{max\_iter}$ as the fraction of training iterations that have been completed since the start of the optimization of the network parameters, the λ update schedule proposed in [140] is as follows:

$$\lambda_p = \frac{2}{1 + \exp(-\gamma \cdot p)} - 1$$

4.6 DA-PanopticFPN model implementation

As all components of the DA-PanopticFPN architecture have been presented, the architecture implementation can be described in detail.

The built model consists of an **FPN backbone**, which utilizes as bottom-up embeddings the feature map outputs of the $\{conv2, conv3, conv4, conv5\}$ ResNet50 stages, here named respectively as $\{C_2, C_3, C_4, C_5\}$. From these, the top-down pathway composed of $\{P_2, P_3, P_4, P_5, P_6\}$ is built by utilizing the mentioned ResNet50 embeddings. P_6 is only utilized by the RPN module to generate object proposals at the largest possible scale, but is not considered for any other task.
For all levels $\{P_2, P_3, P_4, P_5, P_6\}$, the number of channels is $n\_channels = 256$, in order to render all components compatible.

The **Mask-RCNN** instance segmentation, bounding box regression and classification branches are employed as a means to output instance mask predictions.
It predicts in parallel **instance masks, bounding boxes** and the respective **object classes** by employing object proposals generated by the RPN from the FPN top-down levels $\{P_2, P_3, P_4, P_5\}$
It is set up so as to predict $n\_thing\_classes = 2$, which is the number of things categories that are considered in the developed synthetic-to-real dataset: pedestrian and car.

The **semantic segmentation branch** is described in Figure 4.6.

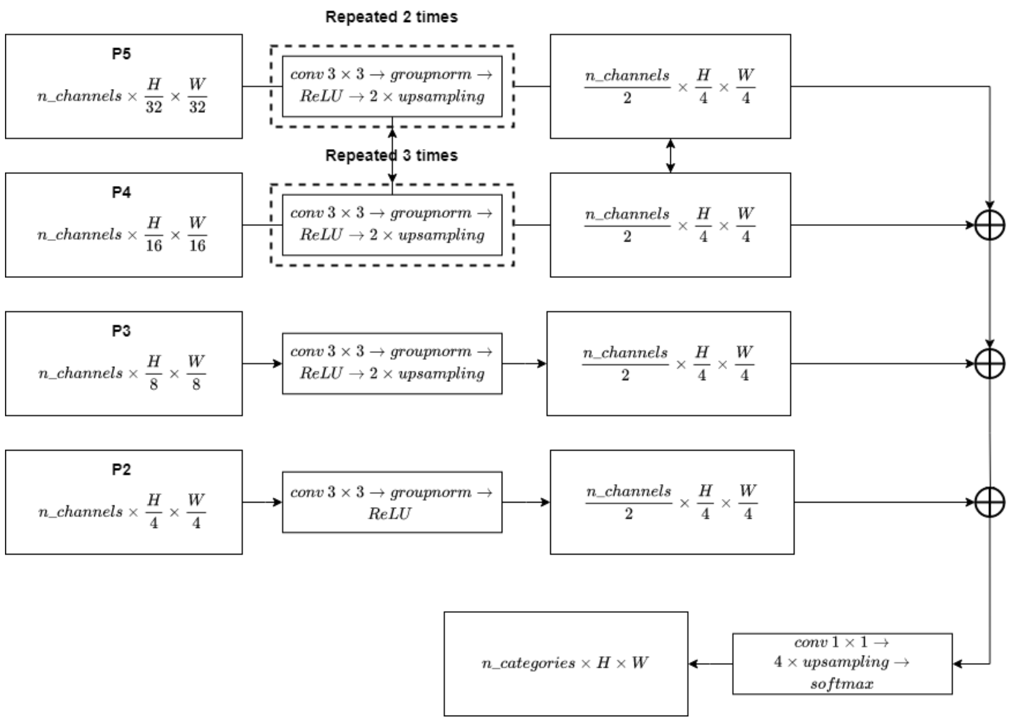

Repeated 2 times
P5
$n\_channels \times \frac{H}{32} \times \frac{W}{32}$
$conv\ 3 \times 3 \rightarrow groupnorm \rightarrow ReLU \rightarrow 2 \times upsampling$
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
Repeated 3 times
P4
$n\_channels \times \frac{H}{16} \times \frac{W}{16}$
$conv\ 3 \times 3 \rightarrow groupnorm \rightarrow ReLU \rightarrow 2 \times upsampling$
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
P3
$n\_channels \times \frac{H}{8} \times \frac{W}{8}$
$conv\ 3 \times 3 \rightarrow groupnorm \rightarrow ReLU \rightarrow 2 \times upsampling$
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
P2
$n\_channels \times \frac{H}{4} \times \frac{W}{4}$
$conv\ 3 \times 3 \rightarrow groupnorm \rightarrow ReLU$
$\frac{n\_channels}{2} \times \frac{H}{4} \times \frac{W}{4}$
$n\_categories \times H \times W$
$conv\ 1 \times 1 \rightarrow 4 \times upsampling \rightarrow softmax$

Figure 4.6: Semantic segmentation branch

It utilizes $\{P_2, P_3, P_4, P_5\}$ as inputs and predicts the output semantic segmentation mask by means of a merged feature representation. It segments the input image into $n\_stuff\_classes = 11 + 1$, which is the number of stuff categories that are considered in the developed synthetic-to-real dataset: road, sidewalk,building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky.
The additional category is used as in [65] to group all entities that are instead considered as things, and are predicted by the Mask R-CNN branch.

The **domain adaptive** component is the main object of study, within the PanopticFPN architecture.
As its FPN backbone contains the embeddings $\{C_2, C_3, C_4, C_5\}$ of the ResNet50 in the bottom-up pathway, from low to high level semantics[91] and the ones of the top-down pathway $\{P_2, P_3, P_4, P_5, P_6\}$, rich in semantics at all levels it becomes unclear to which of these the domain classifier proposed by the authors of [140] should be attached to.
The issue becomes even more relevant if one considers that the Mask R-CNN and the semantic segmentation branch operate with multiple embeddings instead of a single intermediate representation output of shared backbone as in [140]

Hence, the main **research question** which is addressed here regards the optimal placement of one or more domain classifiers, with the objective to achieve the highest possible panoptic quality metric[20].
The only work which explored domain adaptation in the context of multi-task architectures that operate with multi level embeddings is the domain adaptive RetinaNet of[155], implemented for domain adaptation for the detection and classification setting.
The authors show that attaching three different adversarial self-supervised domain classifiers to the $\{C_3, C_4, C_5\}$ levels of the ResNet101 stages yields notable results.
Hence, their proposed implementation for $\{C_3, C_4, C_5\}$ has been implemented, with a simpler variation attached to C_2.
A custom implementation has instead been developed for the $\{P_2, P_3, P_4, P_5, P_6\}$ embeddings.
The entire architecture is shown in Figure 4.7. The domain classifiers are named for brevity DA x, with the suffix which indicates to which of the FPN embeddings it is attached to.

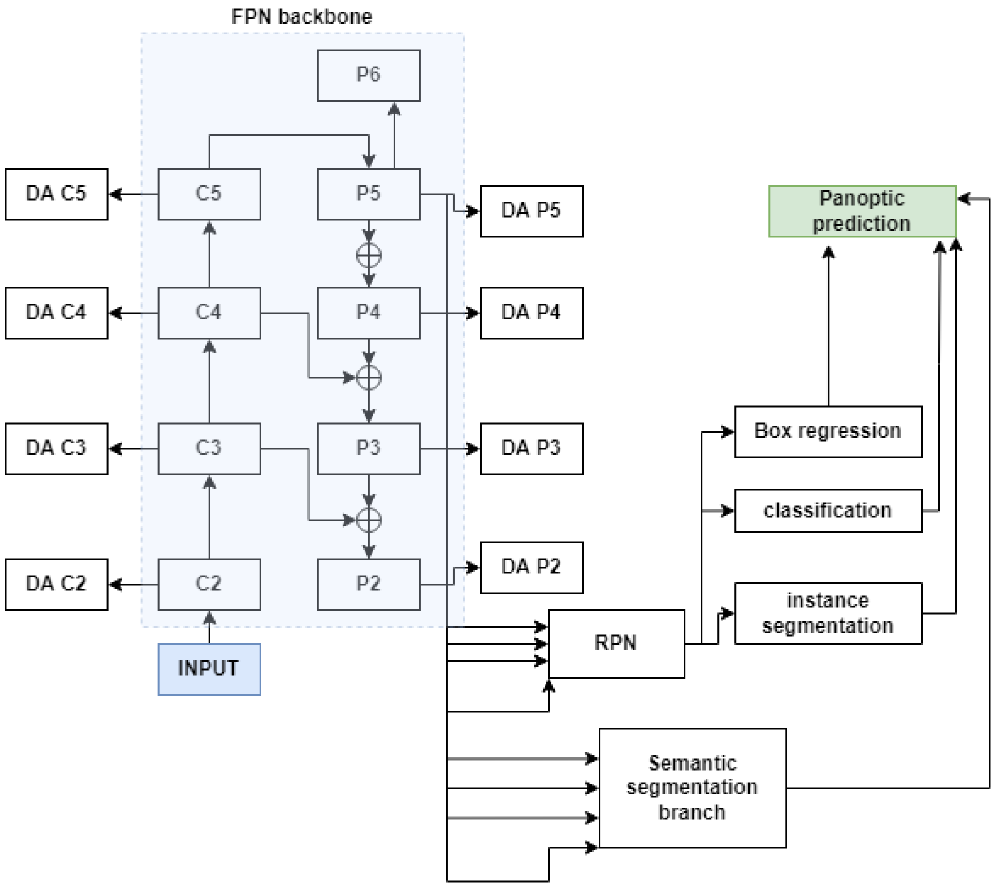


Figure 4.7: DA PanopticFPN architecture

The panoptic mask prediction can be computed by combining instance and semantic segmentation mask predictions.
To achieve this, predicted instances are sorted by decreasing confidence score, removing those under a defined threshold $inst\_score\_thresh = 0.5$. Then, based on such sorting, instance masks are copied into the final panoptic mask prediction. If one of the lower confidence score instance masks has an intersection over union (IoU) overlap ≥ 0.5 with the preliminary panoptic mask to which higher confidence scored instance have been added, it is removed.
As the overlaps are resolved, instances fill the final panoptic prediction mask.
The remaining regions are filled with the semantic segmentation predictions, unless the area covered by any of the semantic segments covers a free area of the final panoptic mask lower than a specified threshold $stuff\_area\_threshold = 4096$.
Thus, the panoptic prediction is computed externally to the model, as a post-processing step indicated in Figure 4.7 by the green block.

4.6.1 Domain classifier implementation

The domain classifier associated with C_2 is built as follows:

1. gradient reversal layer
2. convolutional layer $n\_in\_channels = 256$, $n\_out\_channels = 256$, filters 1×1
3. ReLU
   item convolutional layer $n\_in\_channels = 256$, $n\_out\_channels = 128$, filters 1×1
4. ReLU
5. adapative average pooling, output spatial shape 1×1
6. convolutional layer $n\_in\_channels = 128$, $n\_out\_channels = 1$, filters 1x1

The domain classifier associated with C_3 is built as follows:

1. gradient reversal layer
2. convolutional layer $n\_in\_channels = 512$, $n\_out\_channels = 512$, filters 1×1
3. ReLU
   item convolutional layer $n\_in\_channels = 512$, $n\_out\_channels = 256$, filters 1×1
4. ReLU
5. adapative average pooling, output spatial shape 1×1
6. convolutional layer $n\_in\_channels = 256$, $n\_out\_channels = 1$, filters 1x1

The domain classifier associated with C_4 is built as follows:

1. gradient reversal layer
2. convolutional layer $n\_in\_channels = 1024$, $n\_out\_channels = 512$, filters 3×3, stride$= 2$
3. Batch normalization
4. ReLU
5. convolutional layer $n\_in\_channels = 512$, $n\_out\_channels = 128$, filters 3×3, stride$= 2$

6. Batch normalization
7. ReLU
8. Dropout with probability $p = 0.5$
9. convolutional layer $n\_in\_channels = 128$, $n\_out\_channels = 128$, filters 3×3, stride$= 2$
10. Batch normalization
11. ReLU
12. Dropout with probability $p = 0.5$
13. adapative average pooling, output spatial shape 1×1
14. Linear $n\_in\_channels = 128$, $n\_out\_channels = 1$

The domain classifier associated with C_5 is built as follows:

1. gradient reversal layer
2. convolutional layer $n\_in\_channels = 2048$, $n\_out\_channels = 1024$, filters 3×3, stride$= 2$
3. Batch normalization
4. ReLU
5. Dropout with probability $p = 0.5$
6. convolutional layer $n\_in\_channels = 1024$, $n\_out\_channels = 256$, filters 3×3, stride$= 2$
7. Batch normalization
8. ReLU
9. Dropout with probability $p = 0.5$
10. convolutional layer $n\_in\_channels = 256$, $n\_out\_channels = 256$, filters 3×3, stride$= 2$
11. Batch normalization
12. ReLU
13. Dropout with probability $p = 0.5$

14. adapative average pooling, output spatial shape 1×1
15. Linear $n\_in\_channels = 256$, $n\_out\_channels = 128$
16. Batch Normalization
17. ReLU
18. Dropout with probability $p = 0.5$
19. Linear $n\_in\_channels = 128$, $n\_out\_channels = 1$

The domain classifier associated with $\{P_2, P_3, P_4, P_5, P_6\}$ is built as follows:

1. gradient reversal layer
2. convolutional layer $n\_in\_channels = 256$, $n\_out\_channels = 128$, filters 1×1, stride$= 2$
3. Batch normalization
4. ReLU
5. Dropout with probability $p = 0.5$
6. convolutional layer $n\_in\_channels = 128$, $n\_out\_channels = 64$, filters 1×1, stride$= 2$
7. Batch normalization
8. ReLU
9. Dropout with probability $p = 0.5$
10. adapative average pooling, output spatial shape 1×1
11. Linear $n\_in\_channels = 64$, $n\_out\_channels = 128$
12. Batch normalization
13. ReLU
14. Dropout with probability $p = 0.5$
15. Linear $n\_in\_channels = 128$, $n\_out\_channels = 1$

4.7 Optimization of the model parameters

4.7.1 Introduction

As the DA-PanopticFPN architecture is defined, its computation graph and parameter dependencies are fixed.
Hence, it is possible to jointly optimize the network parameters according to the multiple task which have been assigned to the network.
This is referred to as the process of **training the network**.
By default the weights are arbitrarily initialized, but in order to best fit the objective of the optimization, (e.g. instance or semantic segmentation of the scene, bounding box regression and so on) an error signal is defined with respect to the network predictions and the ground truths, both associated to the selected input data.
As the error function computed between predicted and desired outputs, for the given inputs, it allows to determine the approximate way to modify the parameters of the network so as to obtain predictions which closely match the ground truths.
Such framework is formally defined as the minimization of a **loss function**, which allows to measure the aforementioned error, based on the task which is objective of the optimization.
In the general setting, such optimization problem can be written as[156]:

$$\min_{\theta} F(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_\theta(x_i))$$

In which:

- θ is the set (or a subset) of the optimizable neural network parameters
- $f_\theta(\cdot)$ is the parameterized composite function which represents the neural network
- (x_i, y_i) represents the data and the ground truth label of the i-th sample
- L is the loss function applied to the ground truth and the prediction for the $i-th$ sample

In practice, the minimization happens at sequential steps that allow progressive minimization of the objective, by means of estimates of the theoretical loss.
Such estimates, are computed over subsets of the whole dataset, named batches.
Each batch is used to compute estimates of the loss at each step of the optimization.
By computing the gradients of such estimate with respect to the network parameters, it is possible to optimize each according the the locally optimal minimization step.

This amounts to the modification of the parameters at the i-th step of P maximum iterates, based on the computed direction of descent, over the batch of samples:

$$\theta_{i+1} = \theta_i - \alpha_i \nabla_\theta F(\theta)$$

Here, the gradient is rescaled by the learning rate α_i, which is generally dependent on the iterate at which the gradient is computed.
Although described in brief this is the general procedure followed by batch stochastic gradient descent(SGD)[157], which allows to progressively optimize the parameters of arbitrary functions.
It is a general optimization technique applicable to a broad class of optimization problems, not just neural networks.
There exist more variants of the method which account also for the trajectory followed by subsequent gradients such as ADAM [158] or SGD with momentum[159], thus allowing for faster rates of convergence.

4.7.2 Semantic Segmentation loss

The semantic segmentation branch outputs a tensor of predictions$\mathbf{p}$ of shape $n\_stuff\_classes \times H \times W$, in which $n\_stuff\_classes = 12$, and $H \times W$ is the spatial shape of the input image.
This is done for each image in the input batch of size N.
As such, for each pixel location i, j a vector $v_{i,j}$ of $n\_stuff\_classes$ logits is predicted and normalized to become a vector of probabilities by means of the softmax function.

$$softmax(v_{i,j}) = \frac{\exp^{v_{i,j}}}{\sum_{k=1}^{n\_stuff\_classes} \exp^{\mathbf{p}_{k,i,j}}}$$

Hence, semantic segmentation is treated as a pixel-wise classification problem. Given the one hot encoded ground truth pixel-wise classes $\mathbf{y}_{i,j}$ of each input image in the training batch, the DA-PanopticFPN parameters are optimized by means of the following multi class cross-entropy loss:

$$Lsem\_seg = -\frac{1}{N \cdot H \cdot W} \sum_{b \in \{1,\dots,N\}} \sum_{\substack{i \in \{1,\dots,H\} \\ j \in \{1,\dots,W\}}} \sum_{k \in \{1,\dots,n\_stuff\_classes\}} y^b_{k,i,j} \log(p^b_{k,i,j})$$

Due to the fact that a class is predicted for each location of the image plane $H \times W$, for all N images in the training a batch, the loss is normalized by the constant $N \cdot H \cdot W$ as a means to obtain an estimate of the average classification loss.
It can also be noted, that in the practical implementation the batched images can be of different heights and width.

However, in order to work with fixed shape tensors within the same batch, the images are resized to a common $H \times W$ shape. This is also applied to their semantic segmentation mask.

4.7.3 Mask R-CNN loss

The Mask R-CNN branch involves three parallel tasks: instance segmentation, object proposal classification, bounding box regression from the object proposals to the ground truth boxes .
As training of each the subnetworks which implement the aforementioned tasks requires a common set of ROIs obtained upstream by the RPN module, a batch of a fixed number of ROIs by subsampling the set of proposals.
Considering a batch of $N = 4$ input images, $R_m = 512$ ROIs are sampled from each image, yielding a total of $N_{m\_ROIs} = R_m \cdot N = 2048$.
These are sampled so as to keep a proportion of 1:4 foreground to background ROIs.
If the proportions are maintained, $N_{foreground\_ROIs} \approx \frac{N_{m\_ROIs}}{4}$.
As for Fast R-CNN [106], a ROI is considered positive if its IoU with a ground truth is greater than 0.5 and negative if $0.1 \leq IoU \leq 0.5$ otherwise.
The instance segmentation and bounding box regression losses are **defined only on positive ROIs**, e.g. the respective loss is set to 0 if the ROI is negative.
As a result, as in [68], the mask and bounding box targets are defined by the intersection between an RoI and its associated ground-truth mask/box respectively.
Furthermore, the instance and bounding box regression losses are only considered for the predicted object class, while all others are not accounted for.
Let:

- $n\_things\_classes = 2 + 1$ be the number of considered thing classes plus an additional background category
- p_i be the i-th ROI $n\_things\_classes = 2 + 1$ vector of class predictions
- y_i be the i-th ROI $n\_things\_classes = 2 + 1$ vector which represents the one hot encoded ground truth class (i.e. if the ground truth class is the k-th $y_{i,k} = 1$, while all other positions are equal to 0)
- t_i be the 4-tuple $t_i = (t_x, t_y, t_w, t_h)_i$ (parameterized version of x,y, width and height[106]) that represents the predicted bounding box in transformed coordinates.
- t_i^* be the 4-tuple $t_i^* = (t_x^*, t_y^*, t_w^*, t_h^*)_i$ (parameterized version of x,y, width and height[106]) that represents the cropped ground truth bounding box in transformed coordinates.

- M_i be the i-th ROI $n\_thing\_classes \times 28 \times 28$ mask prediction of the instance segmentation branch.
  As each 28×28 plane represents a mask of logits, sigmoid is applied separately to each plane, in order to obtain at each spatial location the probability that it is a foreground or background pixel.
  A 28×28 mask is predicted for each class, but only the one at the index of the ground truth category is considered during training.
  At inference time instead, the class prediction of the parallel classification branch is considered for selection of the respective mask.

- $M_i^{y_{gt}}$ be the i-th ROI 28×28 mask prediction obtained by selecting from M_i the spatial plane associated with the ground truth label index $y_gt = argmax_{k \in \{1,\ldots,n\_thing\_classes} y_i$

- M_i^* be the i-th ROI 28×28 cropped mask ground truth target obtained as described above, resized to the $\times 28 \times 28$ shape of the prediction.
  Each pixel of the grid is either 1 if it belongs to the foreground object, or 0 if it is associated with the background.

Thus, each of the Mask R-CNN task-specific heads (and the FPN backbone) is optimized by means of the following joint optimization objective:

$$L_m = \frac{1}{N_{m\_ROIs}} L_{m\_reg} + \frac{1}{N_{m\_ROIs}} L_{m\_cls} + \frac{1}{N_{foreground\_ROIs}} L_{m\_mask}$$

$$\begin{aligned}
Lm\_mask &= -\frac{1}{H_{mask} \cdot W_{mask}} \sum_{i \in \{1,\ldots,N\}} \sum_{\substack{j \in \{1,\ldots,H_{mask}\} \\ k \in \{1,\ldots,W_{mask}\}}} (M^*_{i,j,k} \log(M^{y_{gt}}_{i,j,k}) + \\
&\quad (1 - M^*_{i,j,k}) \log(1 - M^{y_{gt}}_{i,j,k})) \\
L_{m\_cls} &= -\sum_{i \in \{1,\ldots,N\}} \sum_{k \in \{1,\ldots,n\_thing\_classes\}} y_{i,k} \log(p_{i,k}) \\
L_{m\_reg} &= \sum_{i \in \{1,\ldots,N\}} \sum_{j \in x,y,w,h} \mathbb{1}\left[y_i \geq 1\right] smooth_{L_1}(t_i^j - t_i^{j*}) \\
smooth_{L_1}(x) &= \begin{cases} 0.5x^2 & if |x| \leq 1 \\ |x| - 0.5 & otherwise \end{cases}
\end{aligned}$$

The contribution of the i-th ROI to the bounding box regression loss L_{m_reg} is only considered if the ground truth class label of the i-th ROI is not the background class(which is represented by class label = 0).

4.7.4 RPN loss

In order to perform object/non-object classification and box regression of the anchors with respect to a target bounding box, the respective ground truths have to be defined.
The procedure that is employed to define the ground truth targets for the RPN is described in [112].
The set of location independent anchors is defined by considering a set of anchor **aspect ratios** and **scales**.All combinations of aspect ratios and scales are considered to generate the set of available anchors i.e. $scales = \{32, 64, 128, 256, 512\}$, aspect ratios = $\{0.5, 1, 2\}$, thus $n\_anchors = card(scales) \cdot (aspect\_ratios)$.
Such set is then associated to all possible spatial locations of the feature maps that are fed to the RPN module.
In practice, if multi-level feature maps are input to the RPN, as is in the case of the FPN backbone, each scale is associated with the respective level of the FPN top-down pathway embeddings e.g. scale= 32 is associated with P_2, scale= 64 with P_3 and so on. Multiple aspect ratios are instead kept at all scales
As this yields a large number of ROIs, biased towards background ones[112], these are subsampled so as to obtain a fixed number of ROIs R_{RPN} from each image in a batch of input samples of size N. Hence, in this work the number of sampled ROIs that are batched together as inputs to the RPN is $N_{RPN\_ROIs} = R_{RPN} * N$.
As a batch of ROIs is defined, the ground truth labels associated with each of the anchors considered by the RPN.
Given an anchor, it is associated with an object(positive label) if:

- it is the anchor with the highest Intersection-overUnion (IoU) overlap with a ground-truth box
- it is an anchor that has an IoU overlap higher than $Acceptance\_IoU\_threshold =$ 0.7 with any ground-truth box

Non-positive anchors are associated with the negative category(background) if their intersection-over-union with all ground truth boxes is less than a defined threshold e.g. $Negative\_IoU\_threshold = 0.3$. All other anchors are instead discarded.
As such matching is completed, the RPN regression target is automatically defined for each positive anchor by the associated ground truth bounding box coordinates.
It is fundamental to note that a batch of ROIs is obtained by subsampling the set of all possible anchors associated to all possible locations of the feature maps which are fed to the RPN module.
As the sampling procedure defined in [106] is used to train the RPN as described in [112],

Hence the RPN loss is defined with the form shown in[112], with adjusted normalization constants :

$$
\begin{aligned}
L_{RPN} &= \frac{1}{N_{RPN\_ROIs}} \sum_{i \in \{1,\dots,N\}} L_{cls}(p_i, p_i^*) \\
&\quad + \frac{1}{N_{RPN\_ROIs}} \sum_{i \in \{1,\dots,N\}} p_i^* L_{reg}(t_i, t_i^*) \\
L_{cls}(p_i, p_i^*) &= -(p_i^* \log(p_i) + (1 - p_i^*) \log(1 - p_i)) \\
L_{reg}(t_i, t_i^*) &= \sum_{j \in x,y,w,h} smooth_{L_1}(t_i^j - t_i^{j*}) \\
smooth_{L_1}(x) &= \begin{cases} 0.5x^2 & if|x| \leq 1 \\ |x| - 0.5 & otherwise \end{cases}
\end{aligned}
$$

In which:

- i is the index of an anchor in a batch of training samples
- p_i is the predicted objectness probability for anchor i
- p_i^* is the ground truth object/not-object label, respectively encoded as 1 and 0
- t_i is a 4-tuple $t_i = (t_x, t_y, t_w, t_h)_i$ representing the predicted bounding box proposal transformed coordinates, respectively associated to the parameterized x,y, width and height[106].
- t_i^* is a 4-tuple $t_i^* = (t_x^*, t_y^*, t_w^*, t_h^*)_i$ representing the ground truth bounding box in transformed coordinates[106].
- L_{cls} is the negative log loss which penalizes wrongly predicted object/non-object anchors.
- L_{reg} is the bounding box regression loss from anchor box coordinates to ground truth box coordinates. It is only activated for samples whose ground truth label is positive e.g. anchors which enclose objects
- N_{RPN_ROIs} is the normalization factor which accounts for the number of sampled ROIs R_{RPN} from each image of given input batch of image samples of size N.

4.7.5 Domain classifiers loss

As the domain classifier performs a self-supervised classification task, the ground truth label is created by considering whether each sample originates from the source(synthetic) or the target(real world) dataset.
Hence it can be represented by a binary classification problem in which, given a

batch of $N = N_{source} + N_{target}$ and $N_{source} = N_{target}$ samples, the source and target domain samples $i \in \{1, ..., N\}$ become associated with a domain label ground truth $d_i^* = 0$ and $d_i^* = 1$, respectively.
Setting $d_{i,j}$ as the predicted probability output for the i-th sample, generated by the j-th $j \in \{C_2, C_3, C_4, C_5, P_2, P_3, P_4, P_5, P_6\}$ domain classifier, the optimized objective function can be described.
For the j-th domain classifier which is attached one of $\{C_2, C_3, C_4, C_5, P_2, P_3, P_4, P_5, P_6\}$, the following binary classification log loss is defined, following the Focal loss variant[160]:

$$L_{dom}^{j} = -\alpha \sum_{i\{1,...,N\}} (d_{i,j}^*(1 - d_{i,j})^\gamma \log(d_{i,j}) + \\ (1 - d_{i,j}^*)(d_{i,j})^\gamma \log(1 - d_{i,j}))$$

In which $\gamma = 2$ and $\alpha = 0.25$ are hyper-parameters of the focal loss, while λ_j is the coefficient that allows to balance the weight of the j-th domain classifier loss. As the neural network is defined, the weight coefficient λ_j of the j-th domain classifier described by the authors of the self-supervised adversarial domain adaptation [140] is not inherently visible in the loss.
This is because it downweights only the sign reversed gradients, computed with respect to the preceding layers parameters.
The domain classifier is optimized without downweighting the gradients with respect to its own parameters.

4.7.6 Joint loss

With the defined partial contributions to the overall loss function, the optimization objective is defined as follows:

$$\begin{aligned}\mathcal{L} = \lambda_i \left(L_{m\_mask} + L_{m\_reg} + L_{m\_cls}\right) + \\ \lambda_s L_{sem\_seg} + \\ L_{RPN} + \\ \sum_{\substack{j \in \{C_2, C_3, C_4, C_5 \\ P_2, P_3, P_4, P_5, P_6\}}} \mathbb{1}\left[\lambda_j > 0\right] L_{dom}^{j}\end{aligned}$$

The $\lambda_i = 1.0$ weighs the instance segmentation, bounding box regression and box classification losses as described in [65].
$\lambda_s = 0.5$ weighs the semantic segmentation loss within the overall objective.
This is the training protocol described by [65] for training the standard PanopticFPN model, as the losses should provide contributions of similar magnitude in order for each to not overwhelm the overall computation of the objective.
The contribution of each domain classifier to the loss is only considered if its respective gradient reversal coefficient λ_j is greater than 0.
RPN is trained jointly with the network by employing the joint approximate training mentioned by the authors of Faster R-CNN [112].

Chapter 5

Experiments

5.1 Experimental context

As the DA-PanopticFPN architecture has been defined, the context of the experiments can be presented.
In this work the objective is to attain an high performance panoptic segmentation model, in terms panoptic quality metric, which generates high quality panoptic mask predictions in a real world scene understanding context, while being trained on labeled synthetic driving images and unlabeled real world scenes.
The reasoning behind the design of such framework stems from the notable disparity in effort required to label real world data compared to the easily attainable synthetic version[62][49]. Simulated data comes by-design with the desired labels[6]and can be generated at a massive scale by means of parallel simulation. Hence no human effort is required.
However, there is a clear tradeoff between the use of synthetic and real data.
While the former is easily attainable, it originates from a different data distribution with respect to real world data.
The disparity between the appearance of simulated and real world data is known as **domain shift** and it hinders the capabilities of models trained purely on synthetic data to generalize on real world scenarios.
Nevertheless, domain adaptation techniques have been developed to mitigate the discrepancy that exists between the synthetic and real domain distributions.

To this end, these experiments have been designed with the objective to find the optimal domain adaptation setup in order to improve panoptic segmentation models.
The selection of possible domain adaptation techniques has been narrowed down to the renowned self-supervised adversarial learning framework proposed by Ganin

et al. [140], due to the remarkable improvements attained by the technique on domain adaptation for classification models.
As their techniques requires modifications in order to be adapted to the PanopticFPN architecture[65],these experiments explore the optimal attachment of one ore more of the domain classifiers proposed by the authors, to the components of the model developed in this dissertation.
More specifically, a total of 9 domain classifiers have been developed and attached to the PanopticFPN backbone embeddings $\{C_2, C_3, C_4, C_5 P_2, P_3, P_4, P_5, P_6\}$, one for each possible output.

In order to assess the performance of each of the domain adaptive models, a **set of PanopticFPN baselines** has been built.
One trained solely on synthetic data and tested on the Cityscapes and bdd100k datasets, required to assess the improvement of each of the domain adaptive models with respect to one that does not have access to real world data.
Two more baselines trained and tested on the real world labels of Cityscapes and bdd100k, in order to assess the maximal possible margin of improvement that is still attainable by completely removing the domain shift between synthetic and real data.
As is mentioned, neither of the baselines employ domain adaptation.

5.2 Metrics: panoptic quality PQ

The context of the panoptic segmentation task requires the joint evaluation of the predictions of thing and stuff classes. Hence, as proposed in [19] the panoptic quality metric(abbreviated as PQ) provides the means to perform such simultaneous assessment, while also accounting for the required interdependence of the predictions.
Before presenting the metric, it is necessary to note that in the context of panoptic segmentation each ground truth is represented by a set of segments, each of which represent the uncountable stuff or the thing instances, the predictions are generated following the same format.
Hence, the **per class panoptic quality metric** is computed following a two step procedure:

1. matching of predicted p and ground truth$g$ segments.
2. panoptic quality computation, given the found matches.

For each class, the matching procedure splits the found matches into three sets: true positives **TP**(all matched pairs), **false positives** (all unmatched predicted

segments), false negatives **FN** (all unmatched ground truths). True negatives are undefined, as unclassified background pixels, which are marked with a special value both in the prediction or in the ground truth, are considered or penalized (in case of wrong prediction) in the computation of the metric.
As a result, the **per class panoptic quality PQ** can be computed as:

$$PQ = \frac{\sum_{(p,g)\in TP} IoU(p,g)}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}$$

Then, the overall panoptic quality is computed as the average of all per class PQ metrics. This renders PQ insensitive to class imbalances[19].

In the above formula, $IoU(p, g)$ is the intersection over union of a given matched true positive pair of predicted and ground truth **binary masks**:

$$IoU(p,g) = \frac{|p \cap g|}{|p \cup g|}$$

In any given binary mask, each pixel associated with an entity of interest of the considered class(regardless of things or stuff), is associated with the value 1, while everything else is set to 0.
The terms $|p \cap g|$ and $|p \cup g|$ respectively represent the cardinality of the intersection and union set obtained from the marked white pixels in the predicted p and ground truth masks g.

5.2.1 Segmentation quality SQ and recognition quality RQ

As seen in [19], panoptic quality can provide further insights on the quality of the model by reformulating the PQ formula by means of multiplication and division by the number of true positives TP.

$$PQ = \underbrace{\frac{\sum_{(p,g)\in TP} IoU(p,g)}{|TP|}}_{\text{segmentation quality SQ}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{recognition quality RQ}}$$

This way, the panoptic quality can provide further insights on the quality of the segmentations(SQ) and its simultaneous capability to detect with a given level of accuracy the entities in the scene(RQ). The former can be interpreted as the average IoU of the matched segments, while the latter is exactly the f1-score

5.3 Experiment design

In order to assess the impact that one or a combination of domain classifiers[140] have on the improvement of the domain adaptive PanopticFPN model, a common hyperparameter setup, described in the Model design chapter, has been implemented for all trained model configurations.

Starting from this common setup, **three baseline** models have been defined, with the provided names:

1. domain adaptation baseline: a baseline model trained solely on the built labeled synthetic dataset and tested on panoptic Cityscapes and BDD100k.

2. target cityscapes baseline: a baseline model trained with label supervision on the Cityscapes training set and tested on the Cityscapes and BDD100k validation sets(no domain adaptation).

3. target bdd100k baseline: a baseline model trained with label supervision on the BDD100k training set and tested on the Cityscapes and BDD100k validation sets(no domain adaptation).

Baseline 1) is the most important one, as it serves as a **reference configuration** to assess the impact that the adversarial self-supervised domain classification task has on improving the panoptic quality metric associated to each of the considered domain classifier configurations.
Baseline 2) and **3)** are instead necessary to understand to which degree the domain shift has been **mitigated** by the domain adaptive model configurations.
As **2) and 3)** are trained with supervision on the target domain(real world) datasets, they allow to **measure the maximal possible panoptic quality** that could theoretically be attained by a domain adaptive model, under the circumstance that the latter completely learns the optimal mapping that allows to remove the domain shift between synthetic and real data.
This holds due to the fact that no other hidden effect would theoretically be limiting the capabilities of the baseline 1) from reaching those of baseline 2) and 3).
Of course, in practice improving the synthetic dataset by adding a greater degree of realism(thus requiring a newer graphic engine) or by hard example mining on the synthetic data might possibly allow to further reduce the domain gap, but this is left for further work.

With the defined baseline models, it is possible to define the domain adaptive classifier configurations objective of the experiment.
Each of such setups is considered as modifications to the template PanopticFPN model, thus each defines a variant of the Domain-Adaptive PanopticFPN.

For each of the domain classifier $i \in S = \{C_2, C_3, C_4, C_5, P_2, P_3, P_4, P_5, P_6\}$, two possible states are defined:

- $\lambda_i = 0$: no gradient is backpropagated to the layers that precede the i-th gradient reversal layer of the i-th domain classifier.
  Thus it does not influence the model parameter optimization and can be considered **detached from the model**

- $\lambda_i \geq 0$: gradients are backpropagated to the layers that precede the i-th gradient reversal layer of the i-th domain classifier, and are reversed and rescaled by the λ_i coefficient.
  In this case, the domain classifier is defined as **attached to the model**

As such setup would yield $2^{cardinality(S)} = 512$ possible configurations, with a necessary training time with the defined setup of $\approx$ 2 hours per configuration, training all such configurations would require $\approx$ 42 days to reach completion.
Hence to reduce the complexity and test in an heuristic manner also possibly more effective configurations, a subset of all such possible combinations is considered.
Domain adaptation is performed separately on the configurations that can be generated by the subsets of S:

- $C = \{C_2, C_3, C_4, C_5\}$, with $2^{cardinality(C)} = 16$ combinations

- $P = \{P_2, P_3, P_4, P_5, P_6\}$, with $2^{cardinality(S)} = 32$ combinations

The reasoning is behind such choice is, besides the reduction in complexity, to apply domain adaptation separately to the embeddings of the bottom-up(the set C) and top-down(the set P) pathway of the FPN backbone architecture.
Thus 48 configurations are possible by defining the possible values of $\lambda_i \in \{0{,}0.5\{$.
It is necessary to note that $\lambda_i = 0.5$ is set to be the maximal reachable value by the λ_i with the per-training iterate update schedule proposed in [140].
$\lambda_i = 0.5$ is chosen due to the fact the reversed gradients even in such configuration can make the training diverge, due to the fact that the PanopticFPN models entails a greater number of tasks, (hence an higher total loss if one considers also the multiple domain classifiers of the DA-PanopticFPN) compared to the classification and self-supervised classification tasks shown by the authors of the adversarial self supervised domain adaptation paper [140].
For such reasons, in the performed experiments the λ_i of each **attached classifiers** is reduced by a factor which depends on the cardinality $|CFG|$ of the subset named CFG of attached classifiers, as with preliminary experiments on the subset $\lambda_i \in \{0{,}0.5\{$ showed the divergence of some configurations in the case of too many attached domain classifiers.
Such preliminary configurations, including the divergent ones, are also considered

in the experiments.
Thus from this description, the following set of experiment configurations are defined:

- $i \in C = \{C_2, C_3, C_4, C_5\}$, with $2^{cardinality(C)} - 1 = 15$(to exclude the baseline with all detached domain classifiers), $\lambda_i \in \{0{,}0.5\{$
- $P = \{P_2, P_3, P_4, P_5, P_6\}$, with $2^{cardinality(S)} - 1 = 31$ (to exclude the baseline with all detached domain classifiers), $\lambda_i \in \{0, \frac{0.5}{|CFG|}\{$ in which $|CFG|$ is the cardinality of the subset of attached classifiers to the DA-PanopticFPN
- $i \in C = \{C_2, C_3, C_4, C_5\}$, with $2^{cardinality(C)} - 1 = 15$ (to exclude the baseline with all detached domain classifiers), $\lambda_i \in \{0, \frac{0.5}{|CFG|+1}\{$ in which $|CFG|$ is the cardinality of the subset of attached classifiers to the DA-PanopticFPN

Thus, a total of 61 experiments on the possible domain adaptive model configurations have been run, with 3 additional separate configurations which represent the aforementioned baselines.

All configurations are validated on a subset of Synthetic-CARLA, Cityscapes and BDD100k.

5.4 Model training setup

Model training has been carried out by considering the same PanopticFPN architecture hyperparameters described in the Model design chapter, for all configurations.

Each DA-PanopticFPN model starts with **pretrained Imagenet**[52] weights, for all parameters that are also present in the template PanopticFPN model.
All domain classifier convolutional layers are initialized by sampling them from a normal distribution with $mean = 0$, standard deviation$= 0.01$. Their linear layers are initialized by sampling from an uniform distribution $U\left[-\frac{1}{\sqrt{in\_features}}, \frac{1}{\sqrt{in\_features}}\right]$.
No biases are used for any of these layers.

In the baseline experiments the batch size has been set to $batch\_size = 4$.
In all other experiments with domain adaptive components the batch is composed by half of the samples from the labeled synthetic domain data and the remaining half from the unlabeled target domain(e.g. Cityscapes or bdd100k as representative of real world data). Thus $batch\_size = 8 = batch\_source + batch\_target$, $batch\_source = batch\_target$.

Due to the fact that each batch of *batch_size* RGB images can contain images of different $H \times W$ height, width and aspect ratios, these are resized so that their shortest side is common and at least $\min(H, W) \geq 800$. If after resizing its longest side $\max(H, W) \geq 1333$, it is downscaled such that it $\max(H, W) = 1333$. Then, they are batched together based on their aspect ratio.
The images are preprocessed by mean subtraction and standardization of the RGB channels, considering the Imagenet[52] mean channel values $[103.530, 116.280, 123.675]$, and standard deviations= $[57.375, 57.120, 58.395]$.
As data augmentation, input images are randomly flipped horizontally with probability $p = 0.5$

Regarding the model parameter optimization, the ADAM[158] algorithm for stochastic optimization has been utilized.
The optimization procedure is run for $N\_iter = 10000$ iterations.
ADAM is configured as follows:

- $momentum = 0.9$
- $\gamma = 0.1$
- weight decay $= 1e - 4$

As a polynomial decay schedule, of which the behavior is shown in figure5.1 is used to modify the learning rate during the N_iter of optimization, such schedule is configured as follows:

- initial learning rate $lr_0 = 0.001$
- final learning rate $lr_{10000} = 0.0$
- linear warmup period of 1000 iterations
- warmup factor set to 0.001
- power= 0.9

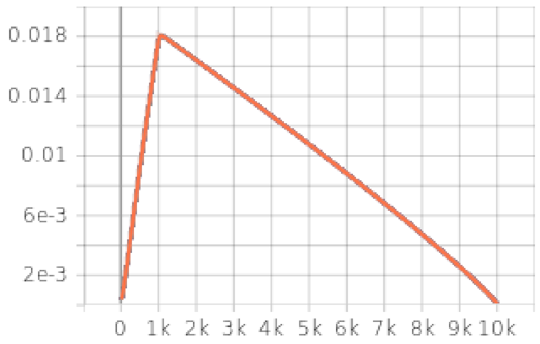


Figure 5.1: Polynomial decay schedule, obtained with the provided configuration

The model is trained on a distributed data parallel setup, on two 3090 GPUs, filling approximately $80 - 95\%$ of the memory of each.

Each configuration requires ≈ 2 hours to complete training and inference steps. Inference and sampling of the validation loss is carried out every 1000 training iterations.

5.5 Experiment results

Based on the described experiment setup, the training results have been gathered. While the total training and validation loss have been, considered, their interpretation is not trivial due to the effect of the domain classifier on the backpropagated reversed gradients[140] , which although allow the model to generalize to the target domain, affect the neural network parameter optimization process.

Aside from this preliminary note, for each model configuration and baseline, the following metrics have been collected at the best validation iterate of the the 10000 training iterations (the first 1000 iterates are not considered as it is the warmup period of the ADAM optimizer):

- panoptic quality on Cityscapes validation and BDD100k out of sample sets
- recognition quality on Cityscapes validation and BDD100k out of sample sets
- segmentation quality on Cityscapes validation and BDD100k out of sample sets
- instance segmentation loss on Cityscapes validation
- semantic segmentation loss on Cityscapes validation
- training iterate

The baseline metrics are reported below, **for brevity the prefix c** or **b** signify Cityscapes validation and bdd100k validation respectively(L stands for Loss):

baseline name	cPQ	cRQ	cSQ	bPQ	bRQ	bSQ	cL seg	cL mask	best iter
DA baseline 1)	26.6	19.6	33.9	26.0	52.6	68.0	0.61	0.48	6000
cityscapes 2)	50.8	62.7	76.6	32.7	40.4	78.3	0.12	0.34	10000
bdd100k 3)	39.9	49.7	73.0	36.3	44.3	81.2	0.15	0.28	10000

The best 16 domain adaptation experiment results are reported below, **sorted by decreasing cityscapes validation panoptic quality**.

For each configuration, the experiment names indicate with the convention Embed-dingName___ $\lambda_{embedding}$,
the FPN backbone embedding name $\{C_2, C_3, C_4, C_5, P_2, P_3, P_4, P_5, P_6\}$ and the maximum λ coefficient assigned to the respective **attached domain classifier**. The λ of all domain classifiers that are associated with an embedding that is not present in the provided naming convention are set $\lambda = 0$, thus are detached from the network. These do not provide any contribution to the loss.

experiment name	**cPQ**	bPQ	cRQ	bRQ	cSQ	bSQ	cL seg	cL mask	best iter
C2_0.17___C5_0.17	**30.1**	22.7	38.3	29.0	63.6	67.7	0.22	0.42	1000
C2_0.25	29.3	21.7	37.2	28.4	58.8	62.6	0.5	0.49	1000
P5_0.25___P6_0.25	28.8	23.3	36.6	29.3	63.8	68.4	0.23	0.44	1000
P3_0.17___P5_0.17 P6_0.17	28.7	23.0	36.7	29.0	59.1	69.1	0.31	0.33	1000
P2_0.17___P3_0.17 P5_0.17	28.7	22.1	36.2	28.1	61.8	68.9	0.25	0.48	2000
C4_0.17___C5_0.17	28.7	22.2	36.6	28.5	64.1	72.4	0.43	0.41	7000
P2_0.17___P4_0.17 P5_0.17	28.7	22.7	36.4	28.5	57.4	66.0	0.27	0.43	2000
P2_0.25___P3_0.25	28.6	23.2	36.6	29.4	63.8	71.3	0.38	0.42	1000
P4_0.25___P5_0.25	28.5	21.9	36.5	28.1	65.0	64.0	0.39	0.38	1000
P2_0.12___P3_0.12 P4_0.12___P6_0.12	28.5	21.0	36.1	27.6	62.8	67.5	0.39	0.42	10000
P2_0.5	28.5	22.6	36.2	28.3	59.1	60.5	0.58	0.55	1000
P2_0.25___P4_0.25	28.5	21.6	36.0	27.8	59.3	72.6	0.45	0.45	7000
C2_0.5	28.4	20.9	35.8	27.1	59.6	68.8	0.38	0.42	5000
C4_0.5	28.4	21.4	36.1	27.6	59.3	68.9	0.29	0.4	5000
P4_0.25___P6_0.25	28.3	21.9	36.3	28.2	62.9	68.3	0.3	0.47	6000
P2_0.12___P3_0.12 P5_0.12___P6_0.12	28.3	21.5	36.1	28.1	63.6	67.8	0.35	0.45	1000

Table 5.1: Domain adaptation experiment metrics, sorted by decreasing panoptic quality on Cityscapes validation set(cPQ)

experiment name	**cPQ**	bPQ	cRQ	bRQ	cSQ	bSQ	cL seg	cL mask	
C2_0.17 C5_0.17	13.2%	15.8%	13.0%	11.5%	20.9%	-0.4%	-63.9%	-12.5%	
C2_0.25	10.2%	10.7%	9.7%	9.2%	11.8%	-7.9%	-18.0%	2.1%	
P5_0.25 P6_0.25	8.3%	18.9%	8.0%	12.7%	21.3%	0.6%	-62.3%	-8.3%	
P3_0.17 P5_0.17 P6_0.17	7.9%	17.3%	8.3%	11.5%	12.4%	1.6%	-49.2%	-31.2%	
P2_0.17 P3_0.17 P5_0.17	7.9%	12.8%	6.8%	8.1%	17.5%	1.3%	-59.0%	0.0%	
C4_0.17 C5_0.17	7.9%	13.3%	8.0%	9.6%	21.9%	6.5%	-29.5%	-14.6%	
P2_0.17 P4_0.17 P5_0.17	7.9%	15.8%	7.4%	9.6%	9.1%	-2.9%	-55.7%	-10.4%	
P2_0.25 P3_0.25	7.5%	18.4%	8.0%	13.1%	21.3%	4.9%	-37.7%	-12.5%	
P4_0.25 P5_0.25	7.1%	11.7%	7.7%	8.1%	23.6%	-5.9%	-36.1%	-20.8%	
P2_0.12 P3_0.12 P4_0.12 P6_0.12	7.1%	7.1%	6.5%	6.2%	19.4%	-0.7%	-36.1%	-12.5%	
P2_0.5	7.1%	15.3%	6.8%	8.8%	12.4%	-11.0%	-4.9%	14.6%	
P2_0.25 P4_0.25	7.1%	10.2%	6.2%	6.9%	12.7%	6.8%	-26.2%	-6.2%	
C2_0.5	6.8%	6.6%	5.6%	4.2%	13.3%	1.2%	-37.7%	-12.5%	
C4_0.5	6.8%	9.2%	6.5%	6.2%	12.7%	1.3%	-52.5%	-16.7%	
P4_0.25 P6_0.25	6.4%	11.7%	7.1%	8.5%	19.6%	0.4%	-50.8%	-2.1%	
P2_0.12 P3_0.12 P5_0.12 P6_0.12	6.4%	9.7%	6.5%	8.1%	20.9%	-0.3%	-42.6%	-6.2%	

Table 5.2: Domain adaptation experiments reported in **percentage change of the metrics with respect to the domain adaptation baseline(baseline 1)**, sorted by decreasing panoptic quality on Cityscapes validation set(cPQ)

experiment name	**cPQ**	bPQ	cRQ	bRQ	cSQ	bSQ	cL seg	cL mask
C2_0.17 C5_0.17	-40.7%	-30.6%	-38.9%	-28.2%	-17.0%	-13.5%	83.3%	23.5%
C2_0.25	-42.3%	-33.6%	-40.7%	-29.7%	-23.2%	-20.1%	316.7%	44.1%
P5_0.25 P6_0.25	-43.3%	-28.7%	-41.6%	-27.5%	-16.7%	-12.6%	91.7%	29.4%
P3_0.17 P5_0.17 P6_0.17	-43.5%	-29.7%	-41.5%	-28.2%	-22.8%	-11.7%	158.3%	-2.9%
P2_0.17 P3_0.17 P5_0.17	-43.5%	-32.4%	-42.3%	-30.4%	-19.3%	-12.0%	108.3%	41.2%
C4_0.17 C5_0.17	-43.5%	-32.1%	-41.6%	-29.5%	-16.3%	-7.5%	258.3%	20.6%
P2_0.17 P4_0.17 P5_0.17	-43.5%	-30.6%	-41.9%	-29.5%	-25.1%	-15.7%	125.0%	26.5%
P2_0.25 P3_0.25	-43.7%	-29.1%	-41.6%	-27.2%	-16.7%	-8.9%	216.7%	23.5%
P4_0.25 P5_0.25	-43.9%	-33.0%	-41.8%	-30.4%	-15.1%	-18.3%	225.0%	11.8%
P2_0.12 P3_0.12 P4_0.12 P6_0.12	-43.9%	-35.8%	-42.4%	-31.7%	-18.0%	-13.8%	225.0%	23.5%
P2_0.5	-43.9%	-30.9%	-42.3%	-30.0%	-22.8%	-22.7%	383.3%	61.8%
P2_0.25 P4_0.25	-43.9%	-33.9%	-42.6%	-31.2%	-22.6%	-7.3%	275.0%	32.4%
C2_0.5	-44.1%	-36.1%	-42.9%	-32.9%	-22.2%	-12.1%	216.7%	23.5%
C4_0.5	-44.1%	-34.6%	-42.4%	-31.7%	-22.6%	-12.0%	141.7%	17.6%
P4_0.25 P6_0.25	-44.3%	-33.0%	-42.1%	-30.2%	-17.9%	-12.8%	150.0%	38.2%
P2_0.12 P3_0.12 P5_0.12 P6_0.12	-44.3%	-34.3%	-42.4%	-30.4%	-17.0%	-13.4%	191.7%	32.4%

Table 5.3: Domain adaptation experiments reported in **percentage change of the metrics with respect to the Cityscapes baseline**(baseline 2) , sorted by decreasing panoptic quality on Cityscapes validation set(cPQ)

5.5.1 Result summary

As can be seen from Table 5.2, the top domain adaptive configuration in the top row, attains a 13.2% and 15.8% panoptic quality(PQ) improvement respectively on Cityscapes and BDD100k validation over the baseline trained solely on synthetic and tested on real data.
Overall, all metrics improve with respect to baseline 1), except for the segmentation quality on BDD100k validation, which is only marginally increased with respect to the baseline metrics.
The losses are all much lower than the baseline ones, with a decrease of -63% and 12.5% in semantic and instance segmentation loss respectively, for the top performing model.
Nevertheless, there is a lot of room for improvement.
As the table 5.2 shows, the top performing domain adaptive model presents a panoptic quality metric which is 40.7% and 30.6% worse than the "oracle" model (baseline 2) metrics, respectively computed on predictions made on Cityscapes and BDD100k validation sets.
Thus, the final model can be defined as the DA-PanopticFPN model which is trained with the domain adaptive classifiers attached to C_2 and C_5(each implemented as described in the model design section), respectively with maximum $\lambda_{C_2} = 0.17$ and $\lambda_{C_5} = 0.17$. The architecture is shown in figure 5.2, with the considered domain adaptive components highlighted in yellow. The ones that must remain detached are greyed out.

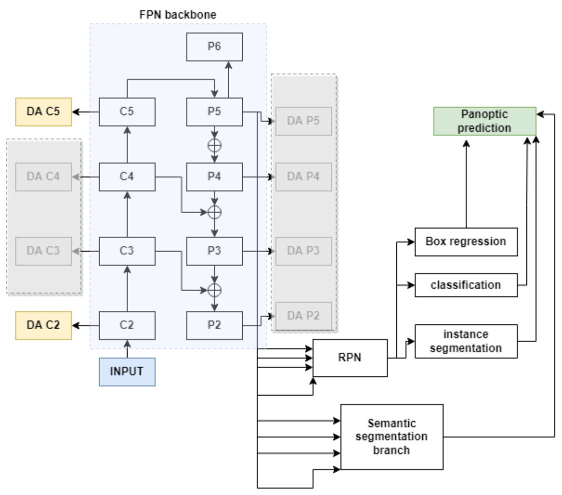


Figure 5.2: DA PanopticFPN architecture with the optimal domain classifiers attached respectively to C_2 and C_5

The PQ metrics obtained on the validation sets by the best configuration at its best training iteration are shown for Synthetic-CARLA, Cityscapes and BDD100K in figure 5.3.

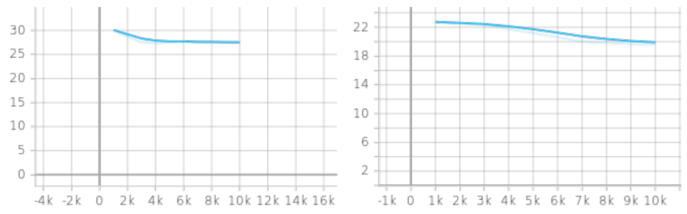


Figure 5.3: PQ metrics of the best model on validation Cityscapes(left) and BDD100k(right)

For comparison, all PQ metrics are plotted in an unique graph in figure 5.4. The grey configuration at the top is the baseline 2) model.

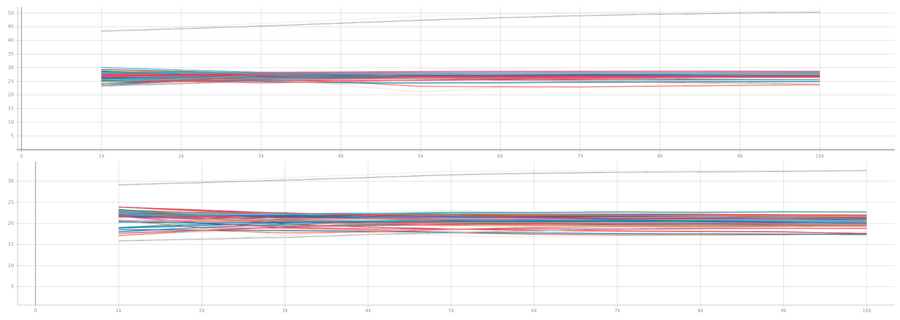


Figure 5.4: PQ metrics of the best model on validation Cityscapes(top) and BDD100k(bottom)

The train and validation total loss(sum of all contributions) of the best configuration on all datasets are plotted in figure 5.5

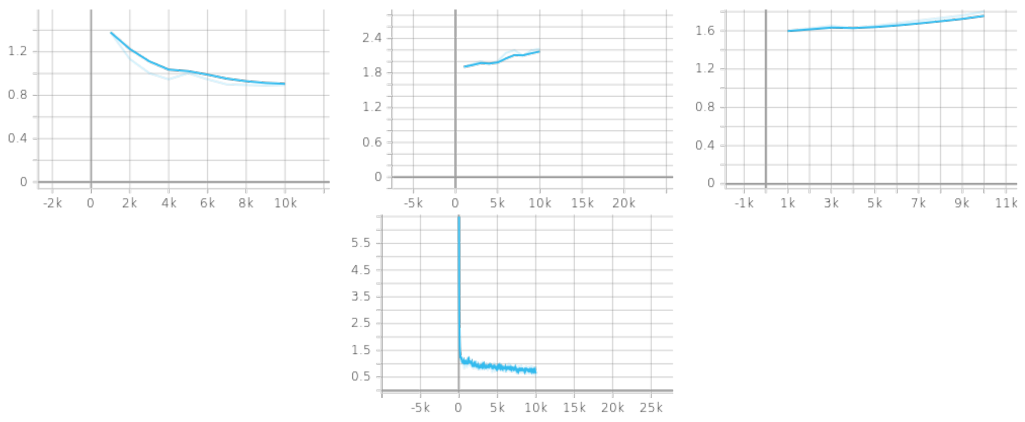


Figure 5.5: Train and validation losses: the train total loss on Synthetic-CARLA(bottom), validation total loss Synthetic-CARLA(top-left), Cityscapes (top-center), BDD100k(top-right)

The panoptic predictions obtained on the validation sets by the best configuration at its best training iteration are shown for Cityscapes in figure 5.6, BDD100K in figure 5.7 and Synthetic-CARLA in figure 5.8.

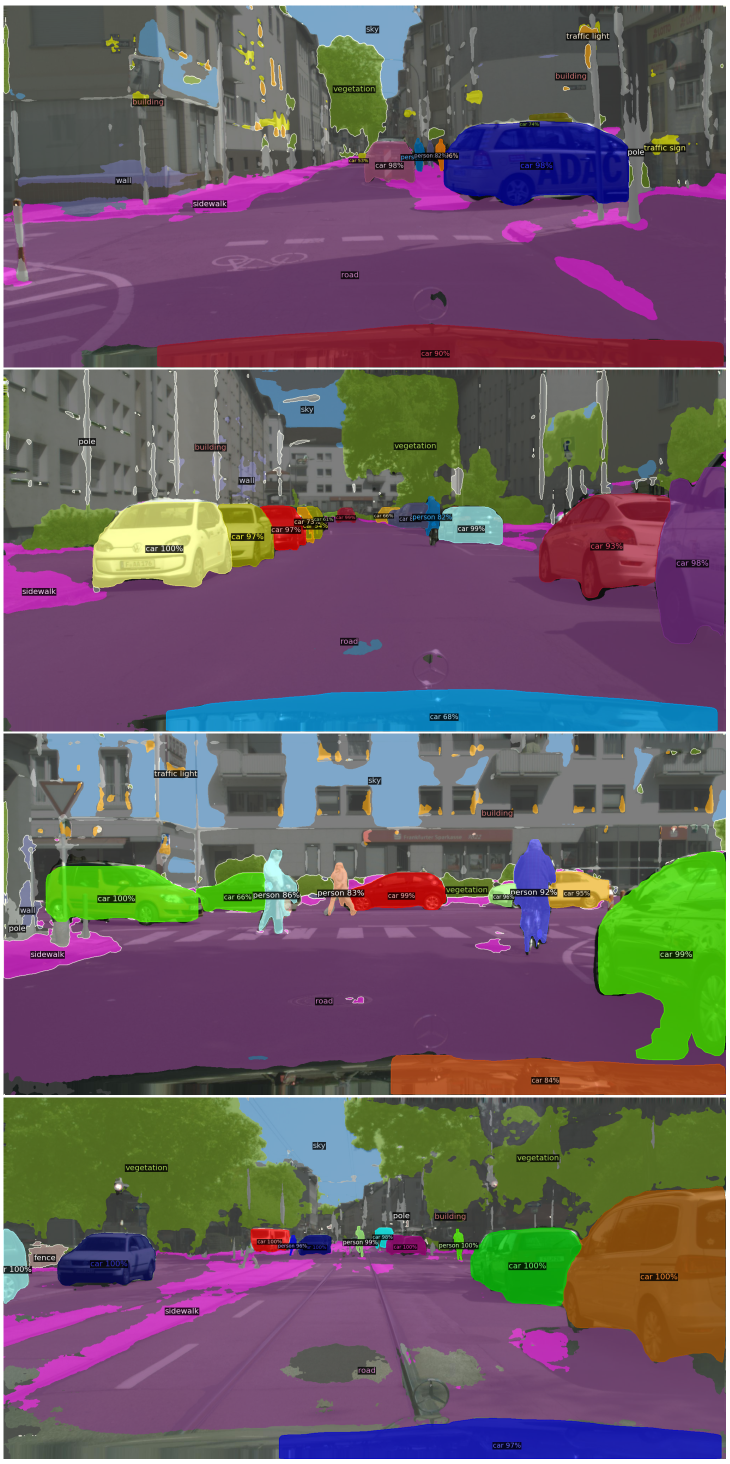


Figure 5.6: validation Cityscapes panoptic predictions of the best configuration

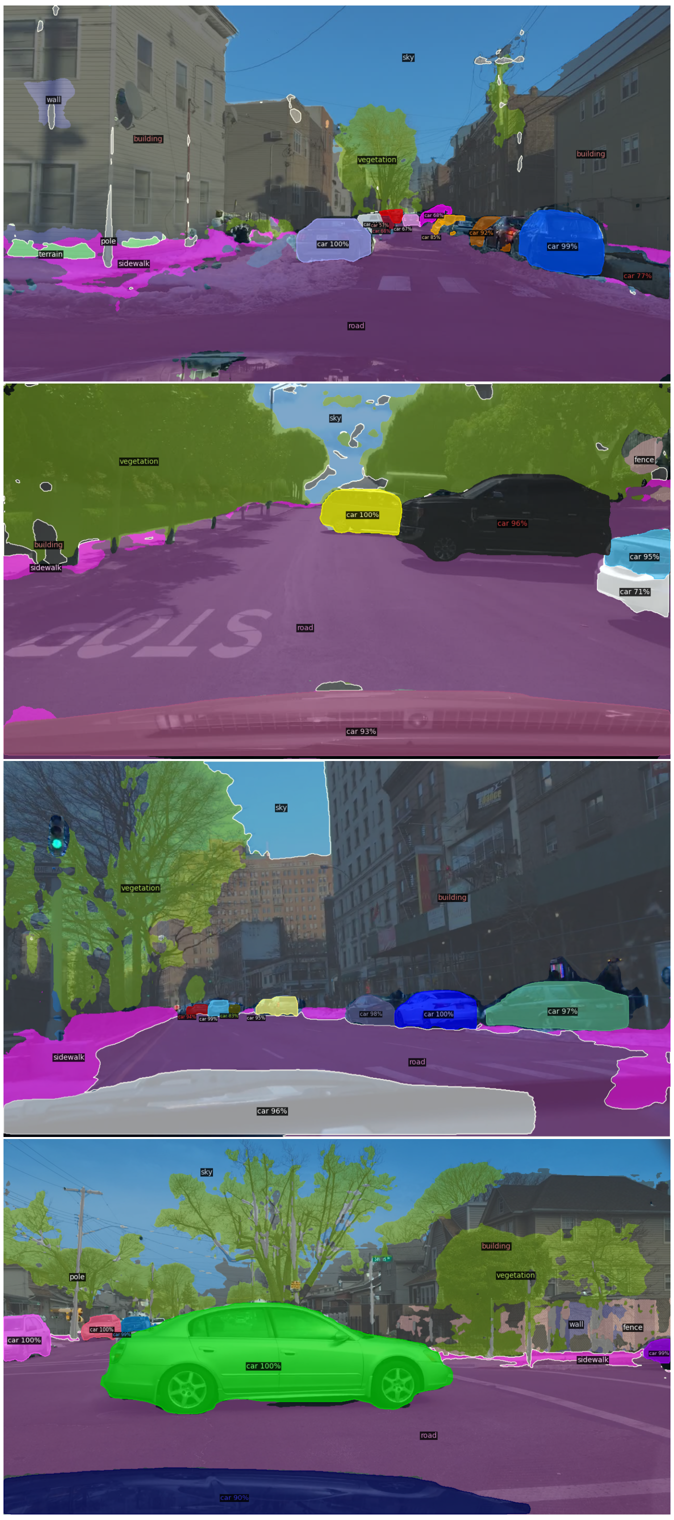


Figure 5.7: validation BDD100k panoptic predictions of the best configuration

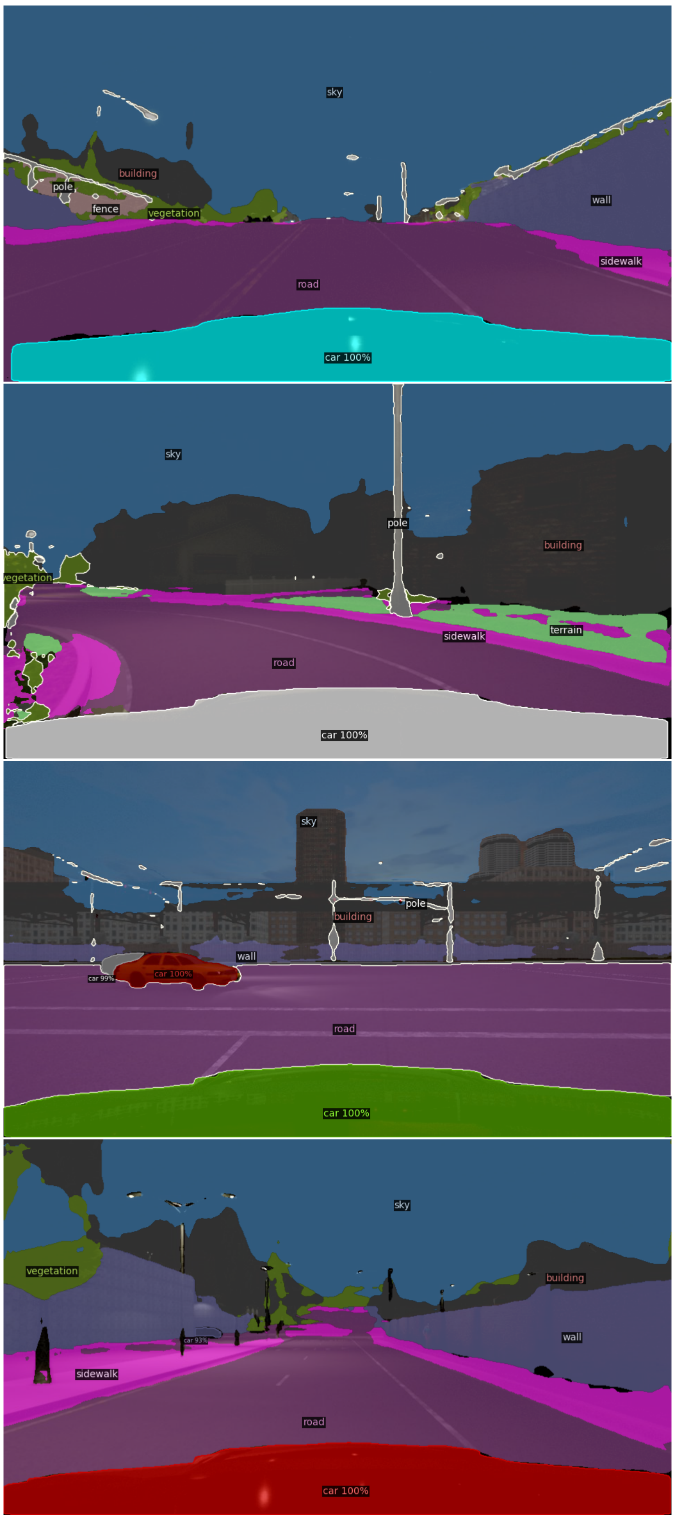


Figure 5.8: validation Synthetic-CARLA panoptic predictions of the best configuration

Chapter 6

Conclusions and further work

The development of SAE level5 autonomous vehicles entails a set of complex problems each of which has to be solved with the most robust algorithms, refined to the utmost perfection.
As the authors of [2] prove, to demonstrate safety and reliability of autonomous vehicles with respect to robustness requirements of the SAE[3] metrics, 100 to 600 years of cumulative driving are needed to collect the needed data, for each separate issue e.g. fatality rate, failure rate, crash rate.
Hence, the authors set the case for the aided development of autonomous vehicles by means of simulation.
However, up to now simulation can only provide approximate models of the real world.
As the discrepancy between simulation and real world data distributions, defined as domain shift, is the major obstacle that hinders the widespread use of simulation for autonomous vehicles, this dissertation focuses its attention on such issue.
The study has been developed from the perspective of the implementation of scene understanding models for autonomous driving, as it presents one of the best use-cases both for deep learning models and domain adaptation techniques that directly address the domain shift problem.
This issue has been addressed in regards to the novel panoptic segmentation task, which allows to capture fine details of the scene as well as detect all actors within it.
As deep models are known to be data hungry, the theoretical complete removal of the domain shift would allow them to be trained on simulation, which by design automatically annotates data, and achieve the same performance of such model trained on costly annotated real world data.

As such, this work developed the context of such use-case, by employing the state of the art CARLA[6] simulator as a means to generate auto-annotated synthetic data. Then, real world data has been obtained by means of the Cityscapes[60] and BDD100k[62] datasets.
With the creation of a label matched dataset for panoptic segmentation, the PanopticFPN model has been trained with proper modifications so as to be usable in a domain adaptation framework.
To this end, the self-supervised adversarial domain adaptation technique described by Ganin et al.[140] has been utilized to adapt the model for supervised and self-supervised training on synthetic and real data respectively.
As self-supervision does not require a human annotator in order to generate ground truths for the data, it enables the use of real world images, although as a restricted version(panoptic annotations on real world data would still provide much better signals for training deep models).
From the experiments that have been run, it has been possible to observe that it is indeed possible to improve models trained on labeled simulated data and self-supervised real world data, by means of domain adaptation. Hence, the optimal self-supervised adversarial domain adaptation setup has been determined and the developed DA-Panoptic FPN has been implemented according to such configuration, shown in Figure 5.2.
Domain adaptation employed as described above, allowed to improve the panoptic quality metric of the chosen model by 13.8% and 15.8% over the same baseline PanopticFPN architecture, named baseline 1, but trained solely on synthetic data and tested on the real world Cityscapes and BDD100k datasets.
Nevertheless, the margin for further improvements are large, as the ideal PanopticFPN model, named baseline 2, trained with label supervision directly on the Cityscapes real world dataset attains much higher panoptic quality.
The former should however be considered only as an ideal condition, as it implies that all real world data would be provided with annotations.
Nevertheless, a theoretical optimal domain adaptive model should be able to mitigate or completely remove the domain shift, so as to attain the same performance as the above "oracle" model.
For comparison, the optimal DA-PanopticFPN architecture attains 40.7% worse panoptic quality compared to the aforementioned oracle model.

Therefore, the avenues for further research on domain adaptation for the panoptic segmentation task are many.
Firstly, as this dissertation focused on the PanopticFPN architecture, the wide variety of panoptic segmentation models that are available in the literature should be benchmarked against the developed DA-PanopticFPN, with their own custom

domain adaptive implementation.
Many of such works have been mentioned in the related works chapter, such as the PanopticDeepLab[21], EfficientPS[22], CVRN[24] convolutional architectures.
Novel research on the panoptic task should also expand to the novel Vision Transformer deep learning models, which approach computer vision problems from the perspective of architectures originally developed to work on sequence based data.

Another variation stems from the use of the SYNTHIA [25] or GTA5[5] synthetic datasets, paired with either BDD100k or Cityscapes as a means to enrich the literature on domain adaptation for the panoptic segmentation task by means of easily accessible benchmarks.

Further benchmarks can also be developed by testing the mentioned variety of panoptic segmentation models, properly modified in order to use different domain adaptation techniques, such as discrepancy based approaches (e.g. maximum mean discrepancy), pretext-task based methods (which are similar in nature to the self-supervised task employed in this work) or other adversarial domain adaptation techniques (e.g. GAN based synthetic-to-real image translation methods).

One final development which stays on the same research track as this work, is the enrichment of the CARLA semantic labels so as to provide a set of stuff and thing classes which directly match with the ones of the renowned driving datasets in the literature: Cityscapes, BDD100k,Mapillary Vistas, SYNTHIA, GTA5 to name a few.

Bibliography

[1] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. «Digital Twin: Enabling Technologies, Challenges and Open Research». In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 108952–108971. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2020.2998358` (cit. on pp. 1, 3).

[2] Nidhi Kalra and Susan M. Paddock. «Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?» In: *Transportation Research Part A: Policy and Practice* 94.C (2016). Publisher: Elsevier, pp. 182–193. ISSN: 0965-8564. URL: `https://econpapers.repec.org/article/eeetransa/v_3a94_3ay_3a2016_3ai_3ac_3ap_3a182-193.htm` (visited on 03/26/2022) (cit. on pp. 1, 2, 47, 116).

[3] *J3016C: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International*. URL: `https://www.sae.org/standards/content/j3016_202104/` (visited on 03/28/2022) (cit. on pp. 2, 116).

[4] *Deepdrive*. en. URL: `https://deepdrive.io/` (visited on 02/17/2022) (cit. on p. 3).

[5] *GTA V + Universe*. Jan. 2017. URL: `http://web.archive.org/web/20170111195314/https://openai.com/blog/GTA-V-plus-Universe/` (visited on 02/17/2022) (cit. on pp. 3, 118).

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. «CARLA: An Open Urban Driving Simulator». In: *arXiv:1711.03938 [cs]* (Nov. 2017). arXiv: 1711.03938. URL: `http://arxiv.org/abs/1711.03938` (visited on 02/17/2022) (cit. on pp. 3, 13, 50, 54, 99, 117).

[7] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. «AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles». In: *arXiv:1705.05065 [cs]* (July 2017). arXiv: 1705.05065. URL: `http://arxiv.org/abs/1705.05065` (visited on 02/17/2022) (cit. on p. 3).

[8] *Gazebo : Blog : Vehicle and city simulation*. URL: `https://gazebosim.org/blog/car_sim` (visited on 02/17/2022) (cit. on p. 3).

[9] NVIDIA. *NVIDIA Automotive Simulation.* Jan. 2018. URL: `https://www.youtube.com/watch?v=booEg6iGNyo` (visited on 02/17/2022) (cit. on p. 3).

[10] NVIDIA. *NVIDIA DRIVE Sim.* Jan. 2019. URL: `https://www.youtube.com/watch?v=DXsLDyiONV4` (visited on 02/17/2022) (cit. on p. 3).

[11] Gerard Andrews. *What is synthetic data?* en-US. June 2021. URL: `https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/` (visited on 02/17/2022) (cit. on p. 3).

[12] *Artificial Intelligence & Autopilot.* en-us. URL: `https://www.tesla.com/AI` (visited on 02/07/2022) (cit. on p. 3).

[13] Tesla. *Tesla AI Day.* Aug. 2021. URL: `https://www.youtube.com/watch?v=j0z4FweCy4M&t=5715shttps://www.youtube.com/watch?v=j0z4FweCy4M` (visited on 02/17/2022) (cit. on p. 3).

[14] *VI-grade provides DiM150 dynamic driving simulator to NIO as key solution for the development of new electric cars.* en. URL: `https://www.vi-grade.com/en/about/news/vi-grade-provides-dim150-dynamic-driving-simulator-to-nio-as-key-solution-for-the-development-of-new-electric-cars_37/` (visited on 02/17/2022) (cit. on p. 3).

[15] Alexis C. Madrigal. *Waymo Built a Secret World for Self-Driving Cars.* en. Section: Technology. Aug. 2017. URL: `https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/` (visited on 02/17/2022) (cit. on p. 3).

[16] *Waypoint - The official Waymo blog: Off road, but not offline: How simulation helps advance our Waymo Driver.* URL: `https://blog.waymo.com/2020/04/off-road-but-not-offline--simulation27.html` (visited on 02/17/2022) (cit. on p. 3).

[17] *Waypoint - The official Waymo blog: Simulation City: Introducing Waymo's most advanced simulation system yet for autonomous driving.* URL: `https://blog.waymo.com/2021/06/SimulationCity.html` (visited on 02/17/2022) (cit. on p. 3).

[18] *Self-Driving Simulation | Uber ATG.* en. URL: `https://www.uber.com/us/en/atg/research-and-development/simulation/` (visited on 02/17/2022) (cit. on p. 3).

[19] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. «Panoptic Feature Pyramid Networks». In: *arXiv:1901.02446 [cs]* (Apr. 2019). arXiv: 1901.02446. URL: `http://arxiv.org/abs/1901.02446` (visited on 01/28/2022) (cit. on pp. 3, 100, 101).

[20] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. «Panoptic Segmentation». In: *arXiv:1801.00868 [cs]* (Apr. 2019). arXiv: 1801.00868. URL: `http://arxiv.org/abs/1801.00868` (visited on 01/28/2022) (cit. on pp. 4, 9–11, 15, 45, 59, 86).

[21] Bowen Cheng, Maxwell D. Collins, Yukun Zhu, Ting Liu, Thomas S. Huang, Hartwig Adam, and Liang-Chieh Chen. «Panoptic-DeepLab: A Simple, Strong, and Fast Baseline for Bottom-Up Panoptic Segmentation». In: *arXiv:1911.10194 [cs]* (Mar. 2020). arXiv: 1911.10194. URL: `http://arxiv.org/abs/1911.10194` (visited on 02/22/2022) (cit. on pp. 4, 16, 40, 43, 45, 46, 69, 118).

[22] Rohit Mohan and Abhinav Valada. «EfficientPS: Efficient Panoptic Segmentation». In: *arXiv:2004.02307 [cs]* (Feb. 2021). arXiv: 2004.02307. URL: `http://arxiv.org/abs/2004.02307` (visited on 02/22/2022) (cit. on pp. 4, 16, 40, 45, 69, 118).

[23] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. «UPSNet: A Unified Panoptic Segmentation Network». en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 8810–8818. ISBN: 978-1-72813-293-8. DOI: `10.1109/CVPR.2019.00902`. URL: `https://ieeexplore.ieee.org/document/8953750/` (visited on 01/28/2022) (cit. on pp. 4, 45).

[24] Jiaxing Huang, Dayan Guan, Aoran Xiao, and Shijian Lu. «Cross-View Regularization for Domain Adaptive Panoptic Segmentation». In: *arXiv:2103.02584 [cs]* (Mar. 2021). arXiv: 2103.02584. URL: `http://arxiv.org/abs/2103.02584` (visited on 03/23/2022) (cit. on pp. 4, 46, 118).

[25] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. «The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. June 2016, pp. 3234–3243. DOI: `10.1109/CVPR.2016.352` (cit. on pp. 5, 12, 13, 118).

[26] Qi Wang, Junyu Gao, and Xuelong Li. «Weakly Supervised Adversarial Domain Adaptation for Semantic Segmentation in Urban Scenes». In: *IEEE Transactions on Image Processing* 28.9 (Sept. 2019). arXiv: 1904.09092, pp. 4376–4386. ISSN: 1057-7149, 1941-0042. DOI: `10.1109/TIP.2019.2910667`. URL: `http://arxiv.org/abs/1904.09092` (visited on 03/12/2022) (cit. on p. 5).

[27] Matteo Biasetton, Umberto Michieli, Gianluca Agresti, and Pietro Zanuttigh. «Unsupervised Domain Adaptation for Semantic Segmentation of Urban Scenes». In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ISSN: 2160-7516. June 2019, pp. 1211–1220. DOI: `10.1109/CVPRW.2019.00160` (cit. on p. 5).

[28] Jiaxing Huang, Dayan Guan, Aoran Xiao, and Shijian Lu. «Semi-Supervised Domain Adaptation via Adaptive and Progressive Feature Alignment». In: *arXiv:2106.02845 [cs]* (June 2021). arXiv: 2106.02845. URL: `http://arxiv.org/abs/2106.02845` (visited on 03/11/2022) (cit. on p. 5).

[29] Hui Zhang, Yonglin Tian, Kunfeng Wang, Haibo He, and Fei-Yue Wang. «Synthetic-to-Real Domain Adaptation for Object Instance Segmentation». In: *2019 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. July 2019, pp. 1–7. DOI: `10.1109/IJCNN.2019.8851791` (cit. on p. 5).

[30] Manuel Diaz-Zapata, Özgür Erkent, and Christian Laugier. «Instance Segmentation with Unsupervised Adaptation to Different Domains for Autonomous Vehicles». In: *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. Dec. 2020, pp. 421–427. DOI: `10.1109/ICARCV50220.2020.9305452` (cit. on p. 5).

[31] Zhenwei He and Lei Zhang. «Multi-adversarial Faster-RCNN for Unrestricted Object Detection». In: *arXiv:1907.10343 [cs]* (Sept. 2019). arXiv: 1907.10343. URL: `http://arxiv.org/abs/1907.10343` (visited on 03/12/2022) (cit. on p. 5).

[32] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. «Domain Adaptive Faster R-CNN for Object Detection in the Wild». In: *arXiv:1803.03243 [cs]* (Mar. 2018). arXiv: 1803.03243. URL: `http://arxiv.org/abs/1803.03243` (visited on 03/12/2022) (cit. on p. 5).

[33] Dongnan Liu, Donghao Zhang, Yang Song, Fan Zhang, Lauren O'Donnell, Heng Huang, Mei Chen, and Weidong Cai. «Unsupervised Instance Segmentation in Microscopy Images via Panoptic Domain Adaptation and Task Re-weighting». In: *arXiv:2005.02066 [cs]* (May 2020). arXiv: 2005.02066. URL: `http://arxiv.org/abs/2005.02066` (visited on 03/11/2022) (cit. on p. 5).

[34] Borna Bešić, Nikhil Gosala, Daniele Cattaneo, and Abhinav Valada. «Unsupervised Domain Adaptation for LiDAR Panoptic Segmentation». In: *IEEE Robotics and Automation Letters* 7.2 (Apr. 2022). arXiv: 2109.15286, pp. 3404–3411. ISSN: 2377-3766, 2377-3774. DOI: `10.1109/LRA.2022.3147326`. URL: `http://arxiv.org/abs/2109.15286` (visited on 03/11/2022) (cit. on p. 5).

[35] Lingdong Kong, Niamul Quader, and Venice Erin Liong. «ConDA: Unsupervised Domain Adaptation for LiDAR Segmentation via Regularized Domain Concatenation». In: *arXiv:2111.15242 [cs]* (Nov. 2021). arXiv: 2111.15242. URL: `http://arxiv.org/abs/2111.15242` (visited on 03/12/2022) (cit. on p. 5).

[36] Qiangeng Xu, Yin Zhou, Weiyue Wang, Charles R. Qi, and Dragomir Anguelov. «SPG: Unsupervised Domain Adaptation for 3D Object Detection via Semantic Point Generation». In: *arXiv:2108.06709 [cs]* (Aug. 2021). arXiv: 2108.06709. URL: `http://arxiv.org/abs/2108.06709` (visited on 03/12/2022) (cit. on p. 5).

[37] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Jürgen Gall, and Cyrill Stachniss. «Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset». en. In: *The International Journal of Robotics Research* 40.8-9 (Aug. 2021). Publisher: SAGE Publications Ltd STM, pp. 959–967. ISSN: 0278-3649. DOI: `10.1177/02783649211006735`. URL: `https://doi.org/10.1177/02783649211006735` (visited on 03/01/2022) (cit. on p. 5).

[38] Ze Wang, Weiqiang Ren, and Qiang Qiu. «LaneNet: Real-Time Lane Detection Networks for Autonomous Driving». In: *arXiv:1807.01726 [cs]* (July 2018). arXiv: 1807.01726. URL: `http://arxiv.org/abs/1807.01726` (visited on 02/18/2022) (cit. on p. 8).

[39] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. «MOTS: Multi-Object Tracking and Segmentation». en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 7934–7943. ISBN: 978-1-72813-293-8. DOI: `10.1109/CVPR.2019.00813`. URL: `https://ieeexplore.ieee.org/document/8953401/` (visited on 02/18/2022) (cit. on p. 8).

[40] Tien-Wen Yeh, Huei-Yung Lin, and Chin-Chen Chang. «Traffic Light and Arrow Signal Recognition Based on a Unified Network». en. In: *Applied Sciences* 11.17 (Jan. 2021). Number: 17 Publisher: Multidisciplinary Digital Publishing Institute, p. 8066. ISSN: 2076-3417. DOI: `10.3390/app11178066`. URL: `https://www.mdpi.com/2076-3417/11/17/8066` (visited on 02/18/2022) (cit. on p. 8).

[41] Julian Müller and Klaus Dietmayer. «Detecting Traffic Lights by Single Shot Detection». In: *arXiv:1805.02523 [cs]* (Oct. 2018). arXiv: 1805.02523. URL: `http://arxiv.org/abs/1805.02523` (visited on 02/18/2022) (cit. on p. 8).

[42] Phuc Manh Nguyen, Vu Cong Nguyen, Son Ngoc Nguyen, Linh My Thi Dang, Ha Xuan Nguyen, and Vinh Dinh Nguyen. «Robust Traffic Light Detection and Classification Under Day and Night Conditions». In: *2020 20th International Conference on Control, Automation and Systems (ICCAS).* ISSN: 2642-3901. Oct. 2020, pp. 565–570. DOI: 10.23919/ICCAS50221.2020.9268343 (cit. on p. 8).

[43] Pavly Salah Zaki, Marco Magdy William, Bolis Karam Soliman, Kerolos Gamal Alexsan, Keroles Khalil, and Magdy El-Moursy. «Traffic Signs Detection and Recognition System using Deep Learning». In: *arXiv:2003.03256 [cs, eess]* (Mar. 2020). arXiv: 2003.03256. URL: http://arxiv.org/abs/2003.03256 (visited on 02/19/2022) (cit. on p. 8).

[44] Aashrith Vennelakanti, Smriti Shreya, Resmi Rajendran, Debasis Sarkar, Deepak Muddegowda, and Phanish Hanagal. «Traffic Sign Detection and Recognition using a CNN Ensemble». In: *2019 IEEE International Conference on Consumer Electronics (ICCE).* ISSN: 2158-4001. Jan. 2019, pp. 1–4. DOI: 10.1109/ICCE.2019.8662019 (cit. on p. 8).

[45] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. «SSD: Single Shot MultiBox Detector». In: *arXiv:1512.02325 [cs]* (Dec. 2016). arXiv: 1512.02325. DOI: 10.1007/978-3-319-46448-0_2. URL: http://arxiv.org/abs/1512.02325 (visited on 03/21/2022) (cit. on pp. 8, 35, 81).

[46] Qingquan Li, Long Chen, Quanwen Zhu, Ming Li, Qun Zhang, and Shuzhi Sam Ge. «Intersection detection and recognition for autonomous urban driving using a virtual cylindrical scanner». en. In: *IET Intelligent Transport Systems* 8.3 (2014). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-its.2012.0202, pp. 244–254. ISSN: 1751-9578. DOI: 10.1049/iet-its.2012.0202. URL: https://onlinelibrary.wiley.com/doi/abs/10.1049/iet-its.2012.0202 (visited on 02/19/2022) (cit. on p. 8).

[47] Dhaivat Bhatt, Danish Sodhi, Arghya Pal, Vineeth Balasubramanian, and Madhava Krishna. *Have i reached the intersection: A deep learning-based approach for intersection detection from monocular cameras.* Pages: 4500. 2017. DOI: 10.1109/IROS.2017.8206317 (cit. on p. 8).

[48] *DRIVE Labs: Pursuing Perfection for Intersection Detection - NVIDIA Blog.* en-US. May 2019. URL: https://blogs.nvidia.com/blog/2019/05/10/drive-labs-intersection-detection/ (visited on 02/19/2022) (cit. on p. 8).

[49] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. «COCO-Stuff: Thing and Stuff Classes in Context». In: *arXiv:1612.03716 [cs]* (Mar. 2018). arXiv: 1612.03716. URL: `http://arxiv.org/abs/1612.03716` (visited on 03/22/2022) (cit. on pp. 8–10, 39, 40, 99).

[50] Omar Elharrouss, Somaya Al-Maadeed, Nandhini Subramanian, and Najmath Ottakath. «Panoptic Segmentation: A Review». en. In: (), p. 29 (cit. on p. 11).

[51] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». In: *arXiv:1405.0312 [cs]* (Feb. 2015). arXiv: 1405.0312. URL: `http://arxiv.org/abs/1405.0312` (visited on 03/01/2022) (cit. on p. 11).

[52] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* ISSN: 1063-6919. June 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848` (cit. on pp. 11, 27, 104, 105).

[53] *COCO - Common Objects in Context.* URL: `https://cocodataset.org/#format-data` (visited on 03/28/2022) (cit. on pp. 12, 59).

[54] Edward H. Adelson. «On seeing stuff: the perception of materials by humans and machines». In: ed. by Bernice E. Rogowitz and Thrasyvoulos N. Pappas. San Jose, CA, June 2001, pp. 1–12. DOI: `10.1117/12.429489`. URL: `http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=903694` (visited on 03/12/2022) (cit. on p. 12).

[55] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. «Playing for Data: Ground Truth from Computer Games». In: *arXiv:1608.02192 [cs]* (Aug. 2016). arXiv: 1608.02192. URL: `http://arxiv.org/abs/1608.02192` (visited on 03/22/2022) (cit. on pp. 12, 13).

[56] Daniel Hernandez-Juarez, Lukas Schneider, Antonio Espinosa, David Vázquez, Antonio M. López, Uwe Franke, Marc Pollefeys, and Juan C. Moure. «Slanted Stixels: Representing San Francisco's Steepest Streets». In: *arXiv:1707.05397 [cs]* (July 2017). arXiv: 1707.05397. URL: `http://arxiv.org/abs/1707.05397` (visited on 03/22/2022) (cit. on p. 13).

[57] Emanuele Alberti, Antonio Tavera, Carlo Masone, and Barbara Caputo. «IDDA: a large-scale multi-domain dataset for autonomous driving». In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020). arXiv: 2004.08298, pp. 5526–5533. ISSN: 2377-3766, 2377-3774. DOI: `10.1109/LRA.2020.3009075`. URL: `http://arxiv.org/abs/2004.08298` (visited on 01/28/2022) (cit. on p. 13).

[58] Jean-Emmanuel Deschaud. «KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator». In: *arXiv:2109.00892 [cs]* (Aug. 2021). arXiv: 2109.00892. URL: `http://arxiv.org/abs/2109.00892` (visited on 03/22/2022) (cit. on p. 13).

[59] A Geiger, P Lenz, C Stiller, and R Urtasun. «Vision meets robotics: The KITTI dataset». en. In: *The International Journal of Robotics Research* 32.11 (Sept. 2013). Publisher: SAGE Publications Ltd STM, pp. 1231–1237. ISSN: 0278-3649. DOI: `10.1177/0278364913491297`. URL: `https://doi.org/10.1177/0278364913491297` (visited on 03/01/2022) (cit. on pp. 13, 55).

[60] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. «The Cityscapes Dataset for Semantic Urban Scene Understanding». In: *arXiv:1604.01685 [cs]* (Apr. 2016). arXiv: 1604.01685. URL: `http://arxiv.org/abs/1604.01685` (visited on 02/05/2022) (cit. on pp. 14, 50, 58, 117).

[61] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. «The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes». In: *2017 IEEE International Conference on Computer Vision (ICCV)*. ISSN: 2380-7504. Oct. 2017, pp. 5000–5009. DOI: `10.1109/ICCV.2017.534` (cit. on p. 14).

[62] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. «BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning». In: *arXiv:1805.04687 [cs]* (Apr. 2020). arXiv: 1805.04687. URL: `http://arxiv.org/abs/1805.04687` (visited on 01/28/2022) (cit. on pp. 14, 50, 58, 99, 117).

[63] Ming-Fang Chang et al. «Argoverse: 3D Tracking and Forecasting with Rich Maps». In: *arXiv:1911.02620 [cs]* (Nov. 2019). arXiv: 1911.02620. URL: `http://arxiv.org/abs/1911.02620` (visited on 01/28/2022) (cit. on p. 15).

[64] Holger Caesar et al. «nuScenes: A multimodal dataset for autonomous driving». In: *arXiv:1903.11027 [cs, stat]* (May 2020). arXiv: 1903.11027. URL: `http://arxiv.org/abs/1903.11027` (visited on 01/28/2022) (cit. on p. 15).

[65] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. «Panoptic Feature Pyramid Networks». In: *arXiv:1901.02446 [cs]* (Apr. 2019). arXiv: 1901.02446. URL: `http://arxiv.org/abs/1901.02446` (visited on 01/28/2022) (cit. on pp. 15, 16, 40, 45, 69, 76, 85, 98, 100).

[66] Sukjun Hwang, Seoung Wug Oh, and Seon Joo Kim. «Single-shot Path Integrated Panoptic Segmentation». In: *arXiv:2012.01632 [cs]* (Dec. 2020). arXiv: 2012.01632. URL: `http://arxiv.org/abs/2012.01632` (visited on 02/22/2022) (cit. on p. 16).

[67] Yanwei Li, Hengshuang Zhao, Xiaojuan Qi, Liwei Wang, Zeming Li, Jian Sun, and Jiaya Jia. «Fully Convolutional Networks for Panoptic Segmentation». en. In: (Dec. 2020). URL: `https://arxiv.org/abs/2012.00720v2` (visited on 02/22/2022) (cit. on p. 16).

[68] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. «Mask R-CNN». In: *arXiv:1703.06870 [cs]* (Jan. 2018). arXiv: 1703.06870. URL: `http://arxiv.org/abs/1703.06870` (visited on 01/28/2022) (cit. on pp. 16, 41, 45, 69, 77, 93).

[69] Liang-Chieh Chen, Huiyu Wang, and Siyuan Qiao. «Scaling Wide Residual Networks for Panoptic Segmentation». In: *arXiv:2011.11675 [cs]* (Feb. 2021). arXiv: 2011.11675. URL: `http://arxiv.org/abs/2011.11675` (visited on 02/22/2022) (cit. on p. 16).

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». en. In: (June 2017). URL: `https://arxiv.org/abs/1706.03762v5` (visited on 02/22/2022) (cit. on pp. 16, 27).

[71] Zhuofan Xia, Xuran Pan, Shiji Song, Li Erran Li, and Gao Huang. «Vision Transformer with Deformable Attention». en. In: (Jan. 2022). URL: `https://arxiv.org/abs/2201.00520v2` (visited on 02/22/2022) (cit. on p. 16).

[72] Alexey Dosovitskiy et al. «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». en. In: (Oct. 2020). URL: `https://arxiv.org/abs/2010.11929v2` (visited on 02/22/2022) (cit. on p. 16).

[73] Zhiqi Li, Wenhai Wang, Enze Xie, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, Tong Lu, and Ping Luo. «Panoptic SegFormer: Delving Deeper into Panoptic Segmentation with Transformers». en. In: (Sept. 2021). URL: `https://arxiv.org/abs/2109.03814v3` (visited on 02/22/2022) (cit. on p. 16).

[74] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning representations by back-propagating errors». en. In: *Nature* 323.6088 (Oct. 1986). Number: 6088 Publisher: Nature Publishing Group, pp. 533–536. ISSN: 1476-4687. DOI: `10.1038/323533a0`. URL: `https://www.nature.com/articles/323533a0` (visited on 03/14/2022) (cit. on pp. 17, 27).

[75] Xavier Glorot and Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks». en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics.* ISSN: 1938-7228. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256. URL: `https://proceedings.mlr.press/v9/glorot10a.html` (visited on 03/16/2022) (cit. on p. 18).

[76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification». In: *arXiv:1502.01852 [cs]* (Feb. 2015). arXiv: 1502.01852. URL: `http://arxiv.org/abs/1502.01852` (visited on 03/16/2022) (cit. on p. 18).

[77] Wadii Boulila, Maha Driss, Eman Alshanqiti, Mohamed Al-Sarem, Faisal Saeed, and Moez Krichen. «Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives». en. In: *Advances on Smart and Soft Computing.* Ed. by Faisal Saeed, Tawfik Al-Hadhrami, Errais Mohammed, and Mohammed Al-Sarem. Vol. 1399. Series Title: Advances in Intelligent Systems and Computing. Singapore: Springer Singapore, 2022, pp. 477–484. ISBN: 9789811655586 9789811655593. DOI: `10.1007/978-981-16-5559-3_39`. URL: `https://link.springer.com/10.1007/978-981-16-5559-3_39` (visited on 03/16/2022) (cit. on p. 18).

[78] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. «Data-dependent Initializations of Convolutional Neural Networks». In: *arXiv:1511.06856 [cs]* (Sept. 2016). arXiv: 1511.06856. URL: `http://arxiv.org/abs/1511.06856` (visited on 03/16/2022) (cit. on p. 18).

[79] Vincent Dumoulin and Francesco Visin. «A guide to convolution arithmetic for deep learning». In: *arXiv:1603.07285 [cs, stat]* (Mar. 2016). arXiv: 1603.07285 version: 1. URL: `http://arxiv.org/abs/1603.07285` (visited on 02/28/2022) (cit. on pp. 20, 21, 24).

[80] Fisher Yu and Vladlen Koltun. «Multi-Scale Context Aggregation by Dilated Convolutions». In: *arXiv:1511.07122 [cs]* (Apr. 2016). arXiv: 1511.07122. URL: `http://arxiv.org/abs/1511.07122` (visited on 02/28/2022) (cit. on p. 21).

[81] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928. URL: `http://jmlr.org/papers/v15/srivastava14a.html` (visited on 03/19/2022) (cit. on p. 22).

[82] Sergey Ioffe and Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». In: *arXiv:1502.03167 [cs]* (Mar. 2015). arXiv: 1502.03167. URL: `http://arxiv.org/abs/1502.03167` (visited on 03/19/2022) (cit. on pp. 22, 23).

[83] Vinod Nair and Geoffrey E. Hinton. «Rectified linear units improve restricted boltzmann machines». In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7. (Visited on 03/20/2022) (cit. on pp. 25, 26).

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html` (visited on 03/13/2022) (cit. on pp. 27–29).

[85] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (Nov. 1998). Conference Name: Proceedings of the IEEE, pp. 2278–2324. ISSN: 1558-2256. DOI: `10.1109/5.726791` (cit. on pp. 27, 28).

[86] Jeffrey L. Elman. «Finding Structure in Time». en. In: *Cognitive Science* 14.2 (1990). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1, pp. 179–211. ISSN: 1551-6709. DOI: `10.1207/s15516709cog1402_1`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1` (visited on 03/14/2022) (cit. on p. 27).

[87] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. URL: `https://doi.org/10.1162/neco.1997.9.8.1735` (visited on 03/14/2022) (cit. on p. 27).

[88] *[1609.02907] Semi-Supervised Classification with Graph Convolutional Networks*. URL: `https://arxiv.org/abs/1609.02907` (visited on 03/14/2022) (cit. on p. 27).

[89] Yann LeCun. «Generalization and network design strategies». en. In: *undefined* (1989). URL: `https://www.semanticscholar.org/paper/Generalization-and-network-design-strategies-LeCun/01b6affe3ea4eae1978aec54e87087feb76d9215` (visited on 03/14/2022) (cit. on p. 27).

[90] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. «Backpropagation applied to handwritten zip code recognition». In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667. DOI: `10.1162/neco.1989.1.4.541`. URL: `https://doi.org/10.1162/neco.1989.1.4.541` (visited on 03/14/2022) (cit. on p. 27).

[91] Matthew D. Zeiler and Rob Fergus. «Visualizing and Understanding Convolutional Networks». In: *arXiv:1311.2901 [cs]* (Nov. 2013). arXiv: 1311.2901. URL: `http://arxiv.org/abs/1311.2901` (visited on 03/14/2022) (cit. on pp. 29, 35, 36, 75, 85).

[92] Karen Simonyan and Andrew Zisserman. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: *arXiv:1409.1556 [cs]* (Apr. 2015). arXiv: 1409.1556. URL: `http://arxiv.org/abs/1409.1556` (visited on 03/13/2022) (cit. on pp. 29, 35).

[93] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385. URL: `http://arxiv.org/abs/1512.03385` (visited on 01/28/2022) (cit. on pp. 29–31, 69, 71–73).

[94] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. «Aggregated Residual Transformations for Deep Neural Networks». In: *arXiv:1611.05431 [cs]* (Apr. 2017). arXiv: 1611.05431. URL: `http://arxiv.org/abs/1611.05431` (visited on 03/13/2022) (cit. on pp. 31, 32).

[95] Sergey Zagoruyko and Nikos Komodakis. «Wide Residual Networks». In: *arXiv:1605.07146 [cs]* (June 2017). arXiv: 1605.07146. URL: `http://arxiv.org/abs/1605.07146` (visited on 03/13/2022) (cit. on p. 31).

[96] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. «Squeeze-and-Excitation Networks». In: *arXiv:1709.01507 [cs]* (May 2019). arXiv: 1709.01507. URL: `http://arxiv.org/abs/1709.01507` (visited on 03/15/2022) (cit. on p. 32).

[97] Mingxing Tan and Quoc V. Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». In: *arXiv:1905.11946 [cs, stat]* (Sept. 2020). arXiv: 1905.11946. URL: `http://arxiv.org/abs/1905.11946` (visited on 03/13/2022) (cit. on p. 32).

[98] Edward Adelson, Charles Anderson, James Bergen, Peter Burt, and Joan Ogden. «Pyramid Methods in Image Processing». In: *RCA Eng.* 29 (Nov. 1983) (cit. on p. 33).

[99] D.G. Lowe. «Object recognition from local scale-invariant features». In: *Proceedings of the Seventh IEEE International Conference on Computer Vision.* Vol. 2. Sept. 1999, 1150–1157 vol.2. DOI: `10.1109/ICCV.1999.790410` (cit. on p. 33).

[100] Lara Younes, Barbara Romaniuk, and Eric Bittar. *A COMPREHENSIVE AND COMPARATIVE SURVEY OF THE SIFT ALGORITHM (Feature detection, description, and characterization.* Feb. 2012 (cit. on p. 33).

[101] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition». In: *arXiv:1406.4729 [cs]* 8691 (2014). arXiv: 1406.4729, pp. 346–361. DOI: 10.1007/978-3-319-10578-9_23. URL: http://arxiv.org/abs/1406.4729 (visited on 03/21/2022) (cit. on p. 33).

[102] Jonathan Long, Evan Shelhamer, and Trevor Darrell. «Fully Convolutional Networks for Semantic Segmentation». In: *arXiv:1411.4038 [cs]* (Mar. 2015). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038 (visited on 03/21/2022) (cit. on pp. 33, 35, 36, 43, 44).

[103] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. «U-Net: Convolutional Networks for Biomedical Image Segmentation». In: *arXiv:1505.04597 [cs]* (May 2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597 (visited on 02/28/2022) (cit. on pp. 34–36, 43, 44).

[104] Zhaowei Cai, Quanfu Fan, Rogerio S. Feris, and Nuno Vasconcelos. «A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection». In: *arXiv:1607.07155 [cs]* (July 2016). arXiv: 1607.07155. URL: http://arxiv.org/abs/1607.07155 (visited on 03/21/2022) (cit. on p. 34).

[105] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. «Rich feature hierarchies for accurate object detection and semantic segmentation». In: *arXiv:1311.2524 [cs]* (Oct. 2014). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524 (visited on 02/27/2022) (cit. on pp. 34, 42, 78).

[106] Ross Girshick. «Fast R-CNN». In: *arXiv:1504.08083 [cs]* (Sept. 2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083 (visited on 01/28/2022) (cit. on pp. 34, 39, 78, 80, 93, 95, 96).

[107] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. «Feature Pyramid Networks for Object Detection». In: *arXiv:1612.03144 [cs]* (Apr. 2017). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144 (visited on 02/22/2022) (cit. on pp. 35, 36, 69, 74, 79).

[108] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. «Deconvolutional networks». In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* ISSN: 1063-6919. June 2010, pp. 2528–2535. DOI: 10.1109/CVPR.2010.5539957 (cit. on p. 36).

[109] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. «Path Aggregation Network for Instance Segmentation». en. In: *arXiv:1803.01534 [cs]* (Sept. 2018). arXiv: 1803.01534. URL: `http://arxiv.org/abs/1803.01534` (visited on 03/22/2022) (cit. on p. 37).

[110] Mingxing Tan, Ruoming Pang, and Quoc V. Le. «EfficientDet: Scalable and Efficient Object Detection». en. In: *arXiv:1911.09070 [cs, eess]* (July 2020). arXiv: 1911.09070. URL: `http://arxiv.org/abs/1911.09070` (visited on 03/22/2022) (cit. on pp. 37–39).

[111] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. «NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection». en. In: *arXiv:1904.07392 [cs]* (Apr. 2019). arXiv: 1904.07392. URL: `http://arxiv.org/abs/1904.07392` (visited on 03/22/2022) (cit. on p. 37).

[112] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». In: *arXiv:1506.01497 [cs]* (Jan. 2016). arXiv: 1506.01497. URL: `http://arxiv.org/abs/1506.01497` (visited on 01/28/2022) (cit. on pp. 39, 41, 78, 80, 95, 98).

[113] Nikita Dvornik, Konstantin Shmelkov, Julien Mairal, and Cordelia Schmid. «BlitzNet: A Real-Time Deep Network for Scene Understanding». In: *arXiv:1708.02813 [cs]* (Aug. 2017). arXiv: 1708.02813. URL: `http://arxiv.org/abs/1708.02813` (visited on 03/22/2022) (cit. on p. 39).

[114] Jiale Cao, Yanwei Pang, and Xuelong Li. «Triply Supervised Decoder Networks for Joint Detection and Segmentation». In: *arXiv:1809.09299 [cs]* (Sept. 2018). arXiv: 1809.09299. URL: `http://arxiv.org/abs/1809.09299` (visited on 03/22/2022) (cit. on p. 39).

[115] Jiayuan Mao, Tete Xiao, Yuning Jiang, and Zhimin Cao. «What Can Help Pedestrian Detection?» In: *arXiv:1705.02757 [cs]* (May 2017). arXiv: 1705.02757. URL: `http://arxiv.org/abs/1705.02757` (visited on 03/22/2022) (cit. on p. 39).

[116] Iasonas Kokkinos. «UberNet: Training a 'Universal' Convolutional Neural Network for Low-, Mid-, and High-Level Vision using Diverse Datasets and Limited Memory». In: *arXiv:1609.02132 [cs]* (Sept. 2016). arXiv: 1609.02132. URL: `http://arxiv.org/abs/1609.02132` (visited on 03/22/2022) (cit. on p. 39).

[117] Alex Kendall, Yarin Gal, and Roberto Cipolla. «Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics». In: *arXiv:1705.07115 [cs]* (Apr. 2018). arXiv: 1705.07115. URL: `http://arxiv.org/abs/1705.07115` (visited on 03/22/2022) (cit. on pp. 39, 40).

[118] Jian Yao, Sanja Fidler, and Raquel Urtasun. «Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation». In: *2012 IEEE Conference on Computer Vision and Pattern Recognition.* ISSN: 1063-6919. June 2012, pp. 702–709. DOI: 10.1109/CVPR.2012.6247739 (cit. on p. 39).

[119] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. «Cross-stitch Networks for Multi-task Learning». In: *arXiv:1604.03539 [cs]* (Apr. 2016). arXiv: 1604.03539. URL: http://arxiv.org/abs/1604.03539 (visited on 03/22/2022) (cit. on p. 40).

[120] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. «Fully Convolutional Instance-aware Semantic Segmentation». In: *arXiv:1611.07709 [cs]* (Apr. 2017). arXiv: 1611.07709. URL: http://arxiv.org/abs/1611.07709 (visited on 03/24/2022) (cit. on p. 41).

[121] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. «Instance-sensitive Fully Convolutional Networks». In: *arXiv:1603.08678 [cs]* (Mar. 2016). arXiv: 1603.08678. URL: http://arxiv.org/abs/1603.08678 (visited on 03/24/2022) (cit. on p. 41).

[122] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. «Hypercolumns for Object Segmentation and Fine-grained Localization». In: *arXiv:1411.5752 [cs]* (Apr. 2015). arXiv: 1411.5752. URL: http://arxiv.org/abs/1411.5752 (visited on 03/24/2022) (cit. on p. 42).

[123] Jordi Pont-Tuset, Pablo Arbelaez, Jonathan T. Barron, Ferran Marques, and Jitendra Malik. «Multiscale Combinatorial Grouping for Image Segmentation and Object Proposal Generation». In: *arXiv:1503.00848 [cs]* (Mar. 2016). arXiv: 1503.00848. DOI: 10.1109/TPAMI.2016.2537320. URL: http://arxiv.org/abs/1503.00848 (visited on 03/24/2022) (cit. on p. 42).

[124] Min Bai and Raquel Urtasun. «Deep Watershed Transform for Instance Segmentation». In: *arXiv:1611.08303 [cs]* (May 2017). arXiv: 1611.08303. URL: http://arxiv.org/abs/1611.08303 (visited on 03/24/2022) (cit. on p. 42).

[125] Naiyu Gao, Yanhu Shan, Yupei Wang, Xin Zhao, Yinan Yu, Ming Yang, and Kaiqi Huang. «SSAP: Single-Shot Instance Segmentation With Affinity Pyramid». In: *arXiv:1909.01616 [cs]* (Sept. 2019). arXiv: 1909.01616. URL: http://arxiv.org/abs/1909.01616 (visited on 03/23/2022) (cit. on pp. 42, 47).

[126] Pedro O. Pinheiro, Ronan Collobert, and Piotr Dollar. «Learning to Segment Object Candidates». In: *arXiv:1506.06204 [cs]* (Sept. 2015). arXiv: 1506.06204. URL: http://arxiv.org/abs/1506.06204 (visited on 03/24/2022) (cit. on p. 43).

[127] Xinlei Chen, Ross Girshick, Kaiming He, and Piotr Dollár. «TensorMask: A Foundation for Dense Object Segmentation». In: *arXiv:1903.12174 [cs]* (Aug. 2019). arXiv: 1903.12174. URL: `http://arxiv.org/abs/1903.12174` (visited on 03/24/2022) (cit. on p. 43).

[128] Dor Bank, Noam Koenigstein, and Raja Giryes. «Autoencoders». In: *arXiv:2003.05991 [cs, stat]* (Apr. 2021). arXiv: 2003.05991. URL: `http://arxiv.org/abs/2003.05991` (visited on 02/28/2022) (cit. on p. 43).

[129] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. «Extracting and composing robust features with denoising autoencoders». In: *Proceedings of the 25th international conference on Machine learning*. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 1096–1103. ISBN: 978-1-60558-205-4. DOI: `10.1145/1390156.1390294`. URL: `https://doi.org/10.1145/1390156.1390294` (visited on 02/28/2022) (cit. on p. 43).

[130] Marc'Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. «Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition». In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2007, pp. 1–8. DOI: `10.1109/CVPR.2007.383157` (cit. on p. 43).

[131] Yifei Zhang. «A Better Autoencoder for Image: Convolutional Autoencoder». en. In: (), p. 7 (cit. on p. 43).

[132] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. «Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction». en. In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski. Vol. 6791. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 52–59. ISBN: 978-3-642-21734-0 978-3-642-21735-7. DOI: `10.1007/978-3-642-21735-7_7`. URL: `http://link.springer.com/10.1007/978-3-642-21735-7_7` (visited on 02/28/2022) (cit. on p. 43).

[133] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. «DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs». In: *arXiv:1606.00915 [cs]* (May 2017). arXiv: 1606.00915. URL: `http://arxiv.org/abs/1606.00915` (visited on 03/24/2022) (cit. on pp. 43, 44).

[134] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. «Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs». In: *arXiv:1412.7062 [cs]* (June 2016).

arXiv: 1412.7062. URL: `http://arxiv.org/abs/1412.7062` (visited on 03/24/2022) (cit. on pp. 43, 44).

[135] Wei Liu, Andrew Rabinovich, and Alexander C. Berg. «ParseNet: Looking Wider to See Better». In: *arXiv:1506.04579 [cs]* (Nov. 2015). arXiv: 1506.04579. URL: `http://arxiv.org/abs/1506.04579` (visited on 03/24/2022) (cit. on pp. 43, 44).

[136] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. «Learning Deconvolution Network for Semantic Segmentation». In: *arXiv:1505.04366 [cs]* (May 2015). arXiv: 1505.04366. URL: `http://arxiv.org/abs/1505.04366` (visited on 03/24/2022) (cit. on p. 44).

[137] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation». In: *arXiv:1511.00561 [cs]* (Oct. 2016). arXiv: 1511.00561. URL: `http://arxiv.org/abs/1511.00561` (visited on 03/24/2022) (cit. on p. 44).

[138] Z. Tu, Xiangrong Chen, Alan Yuille, and Song Zhu. «Image Parsing: Unifying Segmentation, Detection, and Recognition». In: *International Journal of Computer Vision* 63 (Jan. 2005), pp. 113–140. DOI: `10.1007/s11263-005-6642-x` (cit. on p. 45).

[139] Tien-Ju Yang, Maxwell D. Collins, Yukun Zhu, Jyh-Jing Hwang, Ting Liu, Xiao Zhang, Vivienne Sze, George Papandreou, and Liang-Chieh Chen. «DeeperLab: Single-Shot Image Parser». In: *arXiv:1902.05093 [cs]* (Mar. 2019). arXiv: 1902.05093. URL: `http://arxiv.org/abs/1902.05093` (visited on 03/23/2022) (cit. on p. 46).

[140] Yaroslav Ganin and Victor Lempitsky. «Unsupervised Domain Adaptation by Backpropagation». In: *arXiv:1409.7495 [cs, stat]* (Feb. 2015). arXiv: 1409.7495. URL: `http://arxiv.org/abs/1409.7495` (visited on 03/26/2022) (cit. on pp. 48, 82–85, 97, 100, 101, 103, 106, 117).

[141] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. «Learning Transferable Features with Deep Adaptation Networks». In: *arXiv:1502.02791 [cs]* (May 2015). arXiv: 1502.02791. URL: `http://arxiv.org/abs/1502.02791` (visited on 03/26/2022) (cit. on p. 48).

[142] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. «Domain Separation Networks». In: *arXiv:1608.06019 [cs]* (Aug. 2016). arXiv: 1608.06019. URL: `http://arxiv.org/abs/1608.06019` (visited on 03/26/2022) (cit. on p. 48).

[143] Baochen Sun, Jiashi Feng, and Kate Saenko. «Return of Frustratingly Easy Domain Adaptation». In: *arXiv:1511.05547 [cs]* (Dec. 2015). arXiv: 1511.05547. URL: `http://arxiv.org/abs/1511.05547` (visited on 03/26/2022) (cit. on p. 48).

[144] Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. «Larger Norm More Transferable: An Adaptive Feature Norm Approach for Unsupervised Domain Adaptation». In: *arXiv:1811.07456 [cs]* (Aug. 2019). arXiv: 1811.07456. URL: `http://arxiv.org/abs/1811.07456` (visited on 03/26/2022) (cit. on p. 48).

[145] Mohammad Reza Loghmani, Luca Robbiano, Mirco Planamente, Kiru Park, Barbara Caputo, and Markus Vincze. «Unsupervised Domain Adaptation through Inter-modal Rotation for RGB-D Object Recognition». In: *arXiv:2004.10016 [cs]* (Apr. 2020). arXiv: 2004.10016. URL: `http://arxiv.org/abs/2004.10016` (visited on 03/26/2022) (cit. on p. 48).

[146] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A. Efros. «Unsupervised Domain Adaptation through Self-Supervision». In: *arXiv:1909.11825 [cs, stat]* (Sept. 2019). arXiv: 1909.11825. URL: `http://arxiv.org/abs/1909.11825` (visited on 03/26/2022) (cit. on p. 48).

[147] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge: Cambridge University Press, 2004. ISBN: 978-0-521-54051-3. DOI: `10.1017/CBO9780511811685`. URL: `https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A` (visited on 03/01/2022) (cit. on p. 52).

[148] Pei Sun et al. «Scalability in Perception for Autonomous Driving: Waymo Open Dataset». In: *arXiv:1912.04838 [cs, stat]* (May 2020). arXiv: 1912.04838. URL: `http://arxiv.org/abs/1912.04838` (visited on 01/28/2022) (cit. on p. 55).

[149] *Autopilot*. en-us. URL: `https://www.tesla.com/autopilot` (visited on 02/23/2022) (cit. on p. 56).

[150] *facebookresearch/detectron2*. original-date: 2019-09-05T21:30:20Z. Mar. 2022. URL: `https://github.com/facebookresearch/detectron2/blob/6886f85baee349556749680ae8c85cdba1782d8e/datasets/prepare_panoptic_fpn.py` (visited on 03/28/2022) (cit. on pp. 60, 61).

[151] *COCO 2018 Panoptic Segmentation Task API (Beta version)*. original-date: 2018-06-27T13:00:46Z. Mar. 2022. URL: `https://github.com/cocodataset/panopticapi/blob/7bb4655548f98f3fedc07bf37e9040a992b054b0/converters/detection2panoptic_coco_format.py` (visited on 03/28/2022) (cit. on p. 60).

[152] Marius Cordts. *The Cityscapes Dataset.* original-date: 2016-02-20T13:00:11Z. Mar. 2022. URL: https://github.com/mcordts/cityscapesScripts/blob/aeb7b82531f86185ce287705be28f452ba3ddbb8/cityscapesscripts/preparation/createPanopticImgs.py (visited on 03/28/2022) (cit. on p. 61).

[153] *bdd100k/to_coco_panseg.py at master · bdd100k/bdd100k.* en. URL: https://github.com/bdd100k/bdd100k (visited on 03/28/2022) (cit. on p. 61).

[154] Yuxin Wu and Kaiming He. «Group Normalization». In: *arXiv:1803.08494 [cs]* (June 2018). arXiv: 1803.08494. URL: http://arxiv.org/abs/1803.08494 (visited on 03/26/2022) (cit. on p. 77).

[155] Giovanni Pasqualino, Antonino Furnari, Giovanni Signorello, and Giovanni Maria Farinella. «An Unsupervised Domain Adaptation Scheme for Single-Stage Artwork Recognition in Cultural Sites». In: *arXiv:2008.01882 [cs]* (Dec. 2020). arXiv: 2008.01882. URL: http://arxiv.org/abs/2008.01882 (visited on 03/26/2022) (cit. on p. 86).

[156] Ruoyu Sun. «Optimization for deep learning: theory and algorithms». In: *arXiv:1912.08957 [cs, math, stat]* (Dec. 2019). arXiv: 1912.08957. URL: http://arxiv.org/abs/1912.08957 (visited on 03/27/2022) (cit. on p. 91).

[157] H. Robbins. «A Stochastic Approximation Method». In: (2007). DOI: 10.1214/AOMS/1177729586 (cit. on p. 92).

[158] Diederik P. Kingma and Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 03/27/2022) (cit. on pp. 92, 105).

[159] Sebastian Ruder. «An overview of gradient descent optimization algorithms». In: *arXiv:1609.04747 [cs]* (June 2017). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (visited on 03/27/2022) (cit. on p. 92).

[160] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. «Focal Loss for Dense Object Detection». In: *arXiv:1708.02002 [cs]* (Feb. 2018). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002 (visited on 03/27/2022) (cit. on p. 97).