



UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXXI Ciclo

A Deep Learning approach to Lane Detection

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Dott. Pietro Cerri

Dottorando: *Marco Allodi*

Anni 2015/2018

Abstract

Lane detection is a crucial element for advanced driver assistance systems (ADAS) and fully autonomous driving. In the last decades much progress has been made to realize systems that provide high reliability in every possible scenario but nowadays most of these systems still work mainly in highways or other highly predictable and structured environments.

In this thesis, the lane detection problem is studied using approaches based on Convolution Neural Network that represents an extremely powerful framework to understand the context of a scene which is a key requirement for detecting lanes in road images. In particular, this thesis focuses on the analysis of the lane detection problem in challenging environments like urban and rural or more generally scenes that present critical lighting, traffic, weather, and environmental conditions.

Another essential requirement of every perception task for autonomous driving is real-time processing. For this reason, the architectures proposed in the thesis are designed to provide the best trade-off between efficiency and accuracy. Regarding the learning procedures, the networks developed are trained to solve an instance segmentation problem to detect the main lane boundaries on the road: ego lane left and right boundaries, left and right lane boundaries. To evaluate the trained models two recently released dataset for lane detection have been used: the TuSimple Lane Detection benchmark, which is composed by images acquired on US highways at daytime, and the BDD100K dataset, which contains road images collected in a wide variety of different environments and conditions. The work performed shows that the implemented architectures and training procedures are able to provide results comparable to other state of the art approaches on the TuSimple Lane Detection Challenge. In the case of more complex and challenging scenarios, the presented network models offer very promising results and this is shown with a qualitative comparison with a classic computer vision based lane detection.

Contents

1	Introduction	1
1.1	Perception and lane detection for autonomous driving	2
1.2	Deep Learning for visual recognition	4
1.3	Thesis contribution and outline	5
2	Related Work	7
2.1	Convolutional neural network for segmentation	7
2.2	Lane Detection	13
2.2.1	Classical Computer Vision based approaches	15
2.2.1.1	Feature extraction	16
2.2.1.2	Lane model fitting	18
2.2.1.3	Lane tracking	19
2.2.2	Deep Learning based approaches	19
3	Lane detection and segmentation using convolutional neural network	25
3.1	Datasets	26
3.1.1	Cityscapes	26
3.1.2	TuSimple lane detection	27
3.1.3	BDD100K	29
3.1.4	Lanes annotation	30
3.2	Model architectures	33
3.2.1	Networks layers	34

3.2.1.1	Convolution	34
3.2.1.2	Pooling	35
3.2.1.3	Batch Normalization	35
3.2.1.4	Dropout	36
3.2.1.5	Activation function	37
3.2.2	Networks blocks	37
3.2.2.1	Residual bottleneck	38
3.2.2.2	Inverted residual	39
3.2.2.3	Residual dilated pyramid	40
3.2.3	Residual bottleneck based architecture	41
3.2.4	Inverted residual based architecture	42
3.2.5	Residual dilated pyramid based architecture	44
3.3	Loss functions	45
3.3.1	Cross Entropy loss	46
3.3.2	Dice loss	47
3.3.3	Instance segmentation for lane detection	48
3.3.4	Curve fitting loss for lane detection	49
3.4	Training	50
3.4.1	Adam optimizer	51
3.4.2	Data augmentation	52
3.4.3	Finetuning a pretrained model	54
3.5	Model optimization	55
3.5.1	Weight pruning	55
3.5.2	Layer cascade network	56
4	Results	61
4.1	Results for semantic segmentation on Cityscapes	61
4.1.1	Weight pruning results	66
4.1.2	Cascade network results	67
4.2	Lane detection results	67
4.2.1	Instance segmentation result	68

Contents	iii
4.2.2 Qualitative results and critical cases	71
4.2.3 Binary segmentation results	75
4.2.4 CNN based vs classic lane detection results	75
5 Discussion	81
5.1 Future works	84
Bibliography	85

List of Figures

1.1	Examples of lane detection scenes in highway environment.	3
1.2	Examples of lane detection scenes in challenging environment. The following situation are pictured: heavy traffic conditions in 1.2a, critical weather conditions in 1.2b, no markings road in 1.2b and difficult lighting conditions in 1.2d	4
2.1	Diagram of the Fully Convolutional network proposed by Long. Image from the paper [1]	8
2.2	Diagram of the SegNet encoder-decoder architecture. Image from the paper [2]	9
2.3	Diagram of Deeplab atrous spatial pyramid pooling (ASPP). Image from the paper [3]	11
2.4	Pyramid Scene Parsing network architecture with the <i>Pyramid Pooling</i> module. Image from the paper [4]	12
2.5	Diagram of the ERFNet architecture. Image from the paper [5] . . .	14
2.6	Example of bird's eye view images. Figure 2.6a shows the bird's eye view without motion compensation while 2.6b shows the bird's eye view computed using a calibration with compensated motion	17
2.7	Example of output produced by DeepLanes network. Image from the paper [6]	20
2.8	Example of output produced by VPGNet. Image from the paper [7] .	21

2.9	Example of the output produced by the network proposed by Kim. Image from the paper [8]	21
2.10	Example of output produced by SCNN in comparison with a standard CNN for segmentation. Image from the paper [9]	22
2.11	Overview of the <i>LaneNet</i> architecture. Image from the paper [10] . .	23
2.12	Overview of the system proposed by Neven. Image from the paper [10]	23
3.1	Example of an image (left figure) with relative fine (central figure) and coarse (right figure) annotations taken from the Cityscapes dataset	27
3.2	Example of images with annotations taken from the TuSimple lane detection dataset	28
3.3	Example of images from the BDD100K dataset with lane markings annotations.	29
3.4	Example of images from the BDD100K dataset with drivable area annotations.	30
3.5	Examples of lane detection with all four lanes boundaries: ego lane left (green), ego lane right (red), left lane (blue), right lane (yellow). On the left are displayed the original images while on the right are displayed the annotated images.	31
3.6	Examples of intersection scene where lanes are not annotated. . . .	31
3.7	Examples of road images with partial markings. On the left are dis- played the original images while on the right are displayed the anno- tated images.	32
3.8	Examples of road images without markings. On the left are displayed the original images while on the right are displayed the annotated images.	33
3.9	An example of dilated convolution using a kernel of size 3x3. The col- ored cells represent the receptive field while the red dots indicate the positions of the kernel elements with rate of 1,2,3 shown respectively in figure 3.9a, 3.9b and 3.9c	35
3.10	Residual block.	38

3.11	Graph of the residual bottleneck module. C indicates the number of input channels while refers t to the reduce factor.	39
3.12	Graph of the inverted residual module. C indicates the number of input channels while refers t to the expand ratio.	40
3.13	Graph of the residual dilated pyramid module.	42
3.14	Residual bottleneck based architecture (RBTNet). DS indicates the downsampler block and RBT the residual bottleneck module. In the connections between blocks C represents the size of the input channel while s indicates the downsample ratio of the feature maps.	43
3.15	Inverted residual based architecture (IRNet). In the connections between blocks C represents the size of the input channel while s indicates the downsample ratio of the feature maps.	44
3.16	Residual dilated pyramid based architecture (RDPNet). DS indicates the downsampler block, RDP defines the residual dilated pyramid block and RBT the residual bottleneck module. In the connections between blocks C represents the size of the input channel while s indicates the downsample ratio of the feature maps.	45
3.17	Comparison between the lanes fitted using a classic homography transformation from the camera calibration and the lanes fitted using the conditioned homography from <i>H-Net</i>	51
3.18	Examples of data augmentation. In 3.18a is shown the original image, in 3.18b is shown the horizontally flipped image, in 3.18c is shown the rotated image, in 3.18d is shown the translated image, in 3.18e is shown the image with color jitter transformation, in 3.18f is shown the image transformed using a composed data augmentation.	53
3.19	Diagram of the three steps weight pruning procedure	56
3.20	Layer cascade architecture	57

3.21	Segmentation of the easy pixels after the first stage using two different thresholds. In figure 3.21a is shown the original image, in 3.21b the segmentation ground truth, in 3.21c the segmentation output after stage 1 with a threshold of 0.95 and in 3.21d the segmentation output after stage 1 with a threshold of 0.995	59
4.1	Qualitative segmentation results on Cityscapes.	63
4.2	Qualitative segmentation results on Cityscapes.	63
4.3	Qualitative segmentation results on Cityscapes.	64
4.4	Qualitative segmentation results on Cityscapes.	64
4.5	Qualitative segmentation results on Cityscapes.	64
4.6	Qualitative segmentation results on Cityscapes.	65
4.7	Qualitative segmentation results on Cityscapes.	65
4.8	Qualitative segmentation results on Cityscapes.	65
4.9	Example of lane label image. The thickness used for the lanes line on the image is four pixels wide.	69
4.10	Qualitative instance segmentation results for lane detection. On the left column is shown the RGB image, on the center column is shown the predicted segmentation and on the right column is shown the ground truth segmentation.	72
4.11	Qualitative instance segmentation results for lane detection. On the left column is shown the RGB image and on the right column is shown predicted segmentation over the image.	73
4.12	Qualitative instance segmentation results for lane detection in case of challenging scenarios where the network fails.	73
4.13	Qualitative instance segmentation results for lane detection. Two cases where the network in subsequent frames obtains really different results.	74

-
- 4.14 Qualitative comparison between CNN based lane detection and classic Computer Vision based lane detection. In the figure urban scenes are displayed presenting clear road markings. On the left column are shown the original images, on the center column are shown the CNN predictions, on the right column are shown the classic lane detection features. 77
- 4.15 Qualitative comparison between CNN based lane detection and classic Computer Vision based lane detection. In the figure are displayed scenes presenting different weather and illumination conditions. On the left column are shown the original images, on the center column are shown the CNN predictions, on the right column are shown the classic lane detection features. 78
- 4.16 Qualitative comparison between CNN based lane detection and classic Computer Vision based lane detection. In the figure are displayed urban scenes characterized by challenging features (traffic, narrow turn and horizontal markings). On the left column are shown the original images, on the center column are shown the CNN predictions, on the right column are shown the classic lane detection features. . . 79

List of Tables

3.1	Level of sparsity and segmentation error obtained on ENet and ERFNet using different thresholds values in the first stage.	58
3.2	Intersection over union results on ERFNet with and without using dilated convolution.	60
3.3	Intersection over union results on ENet with and without using dropout.	60
4.1	Results on the Cityscapes benchmark. IoU for class and categories is reported along with the number of parameters of the networks. The first set of networks are the ones proposed in this work. The second set is composed by lightweight and efficient networks. The last set instead presents networks designed to be as accurate as possible.	62
4.2	IoU class score of ENet and ERFNet on the Cityscapes benchmark applying weight pruning with different sparsity level.	66
4.3	Comparison of IoU class score between ENet and ENet-LC on the Cityscapes benchmark.	67
4.4	IoU score using TuSimple (TS) dataset as training set and test set, TuSimple (TS) and BDD100K datasets as training set and test set, TuSimple (TS) as training set and TuSimple (TS) and BDD100K datasets as test set	69
4.5	Training set, validation set, test set sizes for the TuSimple and BDD100K datasets	70
4.6	Execution times (in milliseconds) for the forward pass.	71

4.7	TuSimple lane detection results. Accuracy, false positive (FP) and false negative (FN) are reported.	71
4.8	IoU score for binary segmentation for models trained on the TuSimple dataset and BDD100K dataset using cross entropy loss and dice loss	75

Chapter 1

Introduction

In recent years autonomous driving has become a topic of great interest and many important companies started to invest in it heavily. The increasing amount of car accidents and deaths on the road is one of the most concerning problems in modern society. The continuous technological progress is significantly contributing to the improvement of safety in transportation, and one of the major breakthroughs will be autonomous driving.

Nowadays the vast majority of produced vehicles already present different Advanced Driving Assistance System (ADAS). The most notable are Collision Avoidance System, Adaptive Cruise Control and Lane Keeping Assistance.

Collision Avoidance System detects approaching obstacles and automatically break the car helping to avoid or reduce the severity of crashes.

Adaptive Cruise Control (ACC) automatically detects a moving vehicle in front of the car and adapt its speed to keep a safe distance from them.

Lane Keeping Assistance automatically steers the vehicle in order to keep it within the lane boundaries. A simplified version of this system is called Lane Departure Warning which alerts the driver, usually using an acoustic signal, that the car is moving out of its lane. A similar system called Lane Change Assistance helps the driver during a lane change maneuver by signaling the presence of oncoming vehicles in the future lane.

While these systems can provide useful help for human drivers in really structured environments like highways, they cannot cope with the complexity of more dynamic and unpredictable environments like urban streets. That means that when these systems are enabled the human driver must always be concentrated on the driving and be ready to take control of the vehicle since every action the ADAS takes is entirely under his responsibility. The goal of autonomous driving is to deliver cars that have no need for human intervention in any situation at any time. This is an extremely challenging task which requires both high reliability since the system must be able to detect obstacles, other vehicles or pedestrians and recognize lanes and the structure of the road, and fast processing speed since the system must be able to take decisions in real-time.

A lot of IT (Google, Apple, Nvidia, Intel) and automotive (Daimler, Ford, Toyota, Tesla) companies are putting huge efforts in the development of their autonomous driving projects, and many of them are already testing their system on public roads. The problem of autonomous driving can be split into three major tasks:

- Perception: it consists of processing the data coming from sensors like cameras, LIDARs or radars to produce a consistent representation of the world including obstacles, lanes or traffic signs around the vehicle.
- Localization: it consists of defining the exact position of the vehicle on a map of the world using perception and other positioning sensors like GPS or odometry.
- Planning: given the world representation and the position of the vehicle in the world it aims to define a valid trajectory for the vehicle.

This thesis will focus on the perception problem.

1.1 Perception and lane detection for autonomous driving

In autonomous driving, perception is the problem of building and reproducing a consistent representation of the world surrounding the ego vehicle including lanes,

other vehicles, pedestrians and generic obstacles using the data coming from sensors like cameras, LIDAR or radars. Lane detection is one most actively researched problem since it is crucial for the realization of fully autonomous driving systems. Much progress has been made in the last decades, and nowadays many ADAS commercially available are capable of automatically keep the vehicle inside the ego lane avoiding dangerous consequences in the case the driver lack of concentration.

Regardless, these systems work mostly in highway environment, which is highly predictable and structured. A couple of examples of this type of environment are shown in figure 1.1. In these images the typical characteristic that made lane detection easy to solve in those cases are all present: clear markings, straight road, and uniform lighting conditions.



Figure 1.1: Examples of lane detection scenes in highway environment.

On the other hand, much progress is still needed to be made to enable these systems to provide high reliability in every possible scenario. Various examples of more challenging scenarios are represented in figure 1.2. In particular, a situation of heavy traffic with no visible markings is shown in 1.2a, a condition of rainy weather is shown in 1.2b, a road with no markings is shown in 1.2b and finally a scene with extreme lighting conditions is shown in 1.2d.

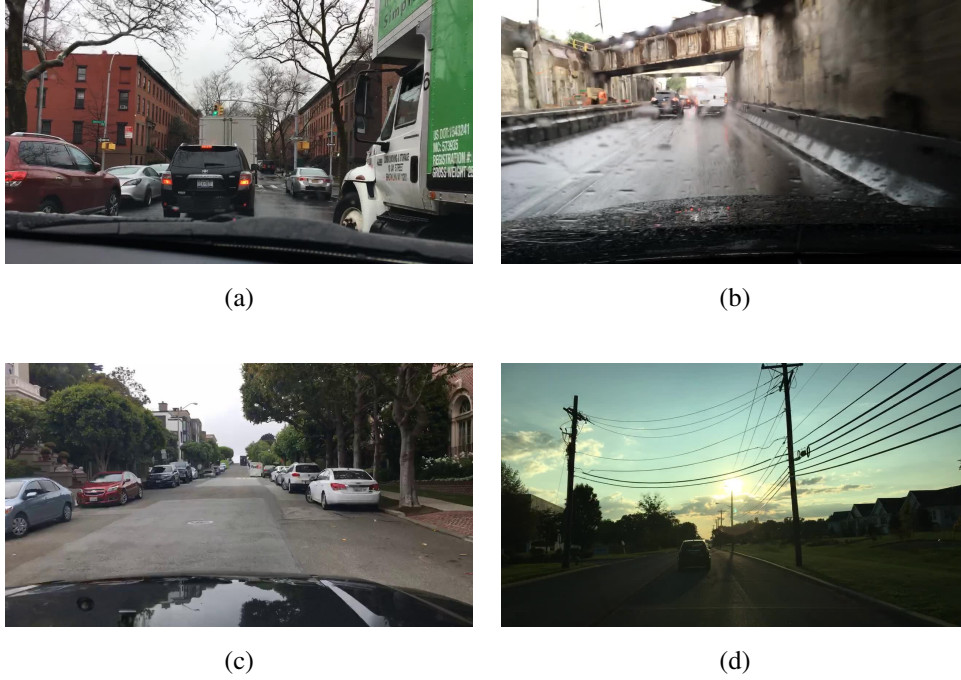


Figure 1.2: Examples of lane detection scenes in challenging environment. The following situation are pictured: heavy traffic conditions in 1.2a, critical weather conditions in 1.2b, no markings road in 1.2b and difficult lighting conditions in 1.2d

1.2 Deep Learning for visual recognition

In the last few years, deep learning was demonstrated to be a really powerful framework for solving visual perception tasks [11]. The most used deep neural network model is Convolutional Neural Network (CNN) which is specifically designed to work on images. In recent years CNN based approaches have outperformed every other previous method in the majority of Computer Vision common tasks like classification [12] [13] [14], object detection [15] [16] [17] and scene segmentation [3] [18] [19]. CNNs are designed to emulate the behavior of the visual cortex and automatically extract a hierarchy of features directly from the input images exploiting their local spatial correlation.

The main characteristics of this type of neural network are represented by local connectivity and weight sharing. Local connectivity means that the neurons of the network are arranged in 3D volumes (width, height, depth), and every neuron in a depth layer is connected only to other neurons in a local region. The connectivity represents the portion of the image seen by the neuron and is called receptive field. Weight sharing means that the parameters of the neuron's local connections are used over the entire input image enabling different neurons in the network to respond to different local input patterns. At the same time weight sharing allows to drastically reduce the number of parameters to learn for the model.

CNNs were initially proposed by Yann Le Cun [20] in the 1990s but became popular only in 2012 with the work of Alex Krizhevsky [14] where he developed a CNN, named AlexNet, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [21].

The key elements that permitted the rapid growth of popularity of Convolutional Neural Network among the Computer Vision community in the last few years were the technological advances of Graphic Processing Unit (GPU) and the availability of large scale datasets. GPU allowed obtaining a huge performance boost in parallel computation, dramatically reducing the total time required to train a network. Large scale datasets allowed the CNNs, which typically requires a massive amount of data to be appropriately trained, to achieve outstanding performance compared to more classical methods based on handcrafted features.

1.3 Thesis contribution and outline

As highlighted in the previous sections the lane detection problem is still an open topic of research despite it has already been studied for decades. Most of the lane detection systems realized both for academic and commercial purpose relies on the extraction of the markings using classical Computer Vision approaches.

The recent advances in the field of deep learning showed that Convolutional Neural Network is the most suited framework to solve visual perception tasks. For this reason, to study the lane detection problem, approaches that focus on the use of Convolution

Neural Network have been chosen. CNN are a powerful tool to exploits context and semantic meaning of the scene which is a key requirement in identifying the lanes in different environments. The contribution of the thesis is to study a method for the lane detection problem that can cope with the highway scenario to more challenging urban and rural cases. These include scenarios where lane markings are not present, and lanes have to be recognized by analyzing the global context and structure of the scene.

During the thesis different type of architectures, loss functions and training procedures have been studied to demonstrate how the problem could be solved even in most challenging scenarios which present critical lighting, traffic, weather, and environmental conditions.

The thesis is organized in the following way. First chapter 2 presents a review of the literature about two macro topics: image segmentation using convolution neural network and lane detection approaches based on both classical computer vision algorithms and deep learning based methods.

Second chapter 3 illustrates the work developed and implemented during the thesis. In particular describes the used datasets, the different implemented network architectures, and training procedures.

Then chapter 4 presents the obtained segmentation and lane detection results on the different datasets. Besides in this chapter is shown a qualitative comparison between a classical and CNN-based lane markings detector.

Finally, chapter 5 describes the conclusions of the work.

Chapter 2

Related Work

This chapter presents a review of the literature about deep learning for segmentation and lane detection. For lane, detection will be presented method based on classical Computer Vision algorithms and more recent approach based on Deep Learning.

2.1 Convolutional neural network for segmentation

A key element for autonomous navigation in an unknown environment is the capability of the system to recognize and reconstruct the semantic meaning of the scene. The goal of semantic segmentation is indeed to parse the image and assign a category to each pixels exploiting the structure of the scene. Nowadays almost every semantic segmentation algorithm is based on convolutional neural networks (CNN).

The first remarkable result was proposed in 2014 by Long et al. [1] whose work represents the first end-to-end fully convolutional network (FCN) for semantic segmentation. They took a common pretrained CNN architecture on a large scale dataset (i.e., ImageNet [22]) like VGGNet [23] or AlexNet [14], and they replaced the fully connected layers with convolutional layers and upsampling layers. In this way, they obtain as output a feature map of score instead of a single value score which represents a dense segmentation labeling of the input image. A scheme of the architecture is shown in figure 2.1.

The network outperformed state of the art algorithms of that time on the commonly used dataset PASCAL VOC [24].

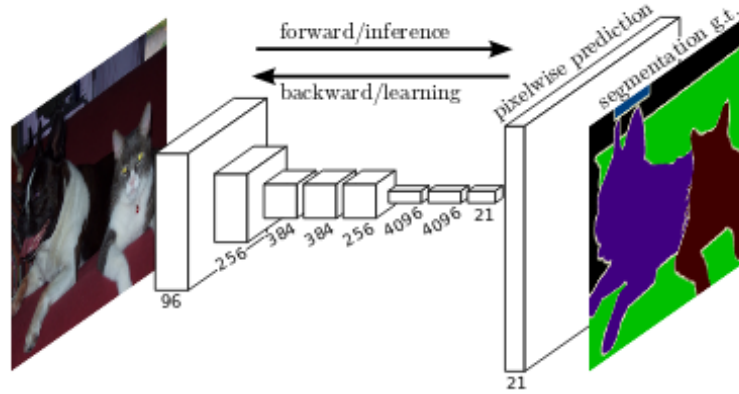


Figure 2.1: Diagram of the Fully Convolutional network proposed by Long. Image from the paper [1]

This work is considered a milestone and inspired every subsequent study in semantic segmentation using CNN. The principle followed for building a semantic segmentation model is to take a common deep neural network, like VGGNet [23], GoogleNet [13], ResNet [12] or DenseNet [25], which is used to extract and encode the information contained in the initial image into lower resolution feature maps. This network is commonly named Encoder and constitutes the first part of the semantic segmentation model.

The purpose of the second part of the model, called Decoder, is to preserve the context information from the lower resolution maps while recovering fine-grained details through upsampling layers.

Another popular network for segmentation was proposed in 2015 by Ronneberger [26]. The network which is called U-Net is fully convolutional and uses skip connections to link feature maps from the encoder to help the corresponding decoder layer to preserve the fine grain details. This work was presented as a network for the segmentation of medical images but later was widely adopted and integrated into

network for the segmentation of road scenes.

Another author to propose an Encoder-Decoder architecture was Badrinarayanan in 2015 with the network called SegNet [2]. In his work, he presents an encoder-decoder architecture in which the decoder uses pooling indices computed in the corresponding max-pooling layer in the encoder to upsample the low resolution feature maps. This type of layer is called unpooling.

SegNet is a fully convolutional network in which the encoder is composed of the first 13 convolutional layers of VGGNet. The decoder is a replica of the encoder for the convolutional layers while the encoder's max pooling layers are replaced by unpooling layers. The scheme of the architecture is shown in figure 2.2.

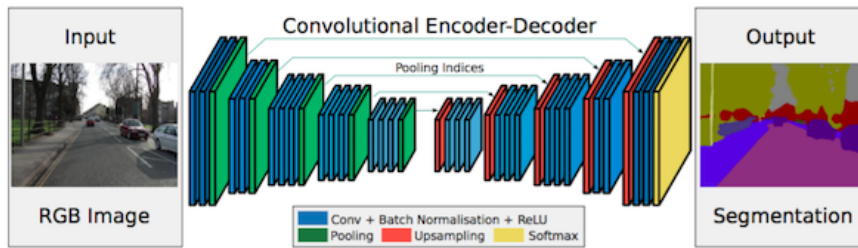


Figure 2.2: Diagram of the SegNet encoder-decoder architecture. Image from the paper [2]

The final network contains almost 30 million parameters and can be trained end-to-end. Besides the use of unpooling layers allows storing in memory only the max pooling indices instead of the whole encoder feature maps. In this way, unpooling layer permits to save more memory with respect to architecture like U-Net where feature maps from the encoder are concatenated to feature maps of the decoder. When released the network achieved state of the art performance on Cityscapes [27] and Camvid datasets.

The authors extend their network in the following work [28] by adding a measure of the model uncertainty for each pixel predicted class label. This has been accomplished at test time by generating a posterior distribution of pixel class labels using Dropout.

The works described so far extensively uses max pooling layers to downsample feature maps in order to reduce computational cost and increase receptive field. While this approach is particularly indicated for classification task, in the case of segmentation, lowering the resolution causes the loss of fine grain details.

To overcome this issue, Yu [29] proposed a module based on a cascade of dilated convolution to preserve spatial resolution and at the same time increase the receptive field. This also allows maintaining the same total number of parameters inside the network.

The architecture used by the author is a modified version of VGGNet where the last two pooling and convolutional layers are replaced by dilated convolution layers with a dilation factor up to 4. This defines the Front-End module of the network and its outputs is taken as inputs by the proposed *Context Module* which is made by five subsequent dilated convolutions layers with doubling dilation factor from 1 to 16. The network achieved state of the art result on the VOC-2012 segmentation dataset showing that keeping high resolution intermediate feature maps is both feasible and advantageous with the use of dilated convolution layers.

Another very popular network for segmentation is DeepLab [3] proposed by Chen in which dilated convolutions are used extensively to segment robustly at multiple scales. In the paper, they presented a module called atrous spatial pyramid pooling (ASPP) which apply convolution to the input feature maps at different dilation rates and then combines the results. More specifically ASPP contains four 3x3 convolutions executed in parallel with dilation rates of 6, 12, 18, 24. A diagram of ASPP is shown in figure 2.3. This allows to detect objects and capture long-range context at different scales and to improve further boundary localization they also added a fully connected Conditional Random Field at the end of the network.

The trend in the following years has been to continue to develop architecture capable of preserving boundaries and details while still capturing long range context.

In 2016 Lin presented RefineNet [30] a multi-path network that enables high-resolution prediction using long-range shortcut connections by exploiting all the intermediate feature maps at different resolutions. The architecture reaches its aim by fusing coarse high-level semantic features with fine-grain low level features to obtain high-resolution

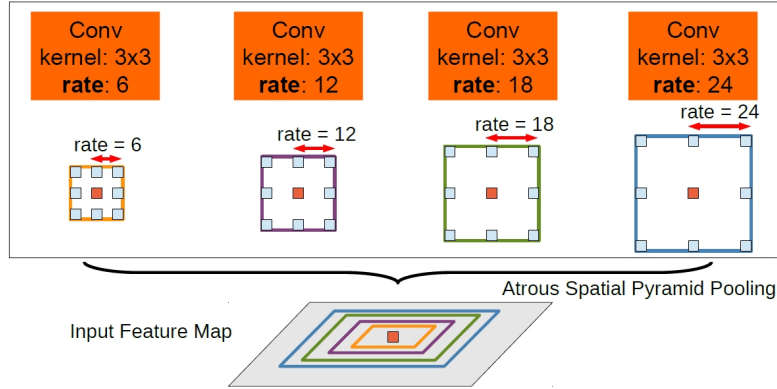


Figure 2.3: Diagram of Deeplab atrous spatial pyramid pooling (ASPP). Image from the paper [3]

semantically robust and detailed prediction. The network achieved an intersection-over-union score of 83.4 on the PASCAL VOC 2012 dataset and a score 87.9 on the Cityscapes which were among the best results at that time.

In 2017 Zhao presented [4] Pyramid Scene Parsing (PSPNet) whose aim is the parsing of complex scenes. This was achieved by aggregating the context of different regions to obtain an effective global prior representation. The author proposed a novel module called *Pyramid Pooling* to build a global scene prior at four different pyramid scales. The coarsest scale is represented by a single element. For the other scales, the feature maps are divided into sub-regions, and for each location, a global representation is obtained. A representation of the network architecture is shown in image 2.4. The network achieved state of the art performance on PASCAL VOC 2012 benchmark and Cityscapes benchmark.

The main downside of the previously cited works is that they employ models that present a huge number of parameters which made them not suitable for the execution in real-time applications.

ENet is a network architecture proposed in 2016 by Paske [31] specifically built to obtain fast inference time and low memory requirement for the segmentation task. The architecture follows the classical encoder-decoder scheme, where the encoder

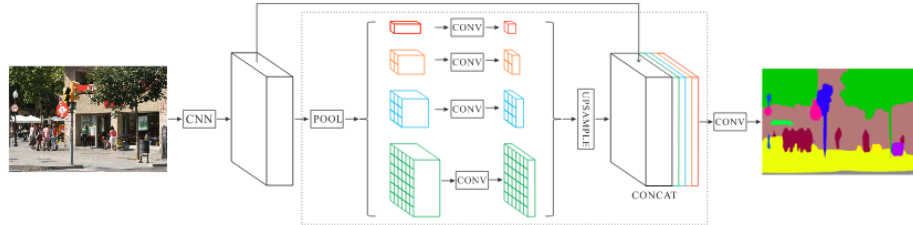


Figure 2.4: Pyramid Scene Parsing network architecture with the *Pyramid Pooling* module. Image from the paper [4]

uses dilated convolution and has a 16x downsample factor with respect to the initial feature maps and a lightweight decoder based on transposed convolution upsampling. The main building block of ENet is called *bottleneck*, a residual module that reduces the number of parameters and floating point required operation. This can be accomplished using three consecutive convolutions, the first is a 1x1 convolution used to reduce the dimensionality of the input, followed by a 3x3 convolution of different types (regular, dilated, transposed) and finally, another 1x1 convolution used to restore the original number of channels. To increase the receptive field and the learning capabilities of the network the 3x3 convolution is replaced by a factorized convolution 1x5, 5x1 in the last modules of the encoder.

To improve significantly inference time the network use a specific initial block that heavily downsamples the input image in an early stage. In opposition to SegNet where the encoder and the decoder have symmetric structure, in Enet the decoder is significantly smaller than the encoder because its task is just to upsample and refine the encoder output details. Overall the network present less than 400K parameters and requires 3.83 GFLOPs for a 640x360 input image.

The network was benchmarked on the CamVid and Cityscapes dataset reporting superior segmentation accuracy with respect to SegNet. In particular, it achieved an IoU score of 58.3 on Cityscapes. Moreover, ENet was able to run at 14.6 fps on an NVIDIA TX1 GPU or 135.4 fps on an NVIDIA Titan X GPU using an input image with size 640x360.

ERFNet (Efficient Residual Factorized ConvNet) [5] proposed by Romera in 2017 is an architecture for real-time semantic segmentation whose aim is to improve the trade-off between accuracy and performance further. ERFNet is a residual based network and is build using a so-called *non-bottleneck-1d* module. Differently from ENet's *bottleneck* this module exploit factorized convolution to achieve higher learning capabilities while reducing the number of parameters. Specifically, the block comprises two consecutive 3x1-1x3 factorized convolution. Overall the network is an encoder-decoder architecture in which, similarly to ENet, the decoder is lightweight, and the encoder operates a 16x downsample on the input and make heavy use of dilated convolution to increase the receptive field.

The diagram of the ERFNet architecture is shown in figure 2.5. The results shown in the paper demonstrates that the *non-bottleneck-1d* module can achieve an IoU-Class score of 69.7 on Cityscapes. The network contains 2 million parameters and is only 2x slower than ENet. Specifically, it runs at 7.1 fps on an NVIDIA TX1 GPU or 83.3 fps on an NVIDIA Titan X GPU using an input image with size 640x360.

2.2 Lane Detection

Lane detection constitutes one of first tackled and most actively researched problem in the field of computer vision applied to autonomous vehicles. It is a crucial building block for every advanced driver assistance system, and much progress has been made in the last decades to realize systems that provide high reliability in every possible scenario.

Nowadays exists many commercial ADAS that are able to drive the vehicle inside the ego lane autonomously. Anyway, these systems can only be used safely in the highway environment which is highly predictable and highly structured.

On the other hand, the urban environment presents a lot more complexities for the lane detection task. The reason for this higher complexity is caused by a series of factors regarding the structure of the road. Indeed urban environment presents particular topology like intersections, roundabouts or u-turns. In general, road curvature is

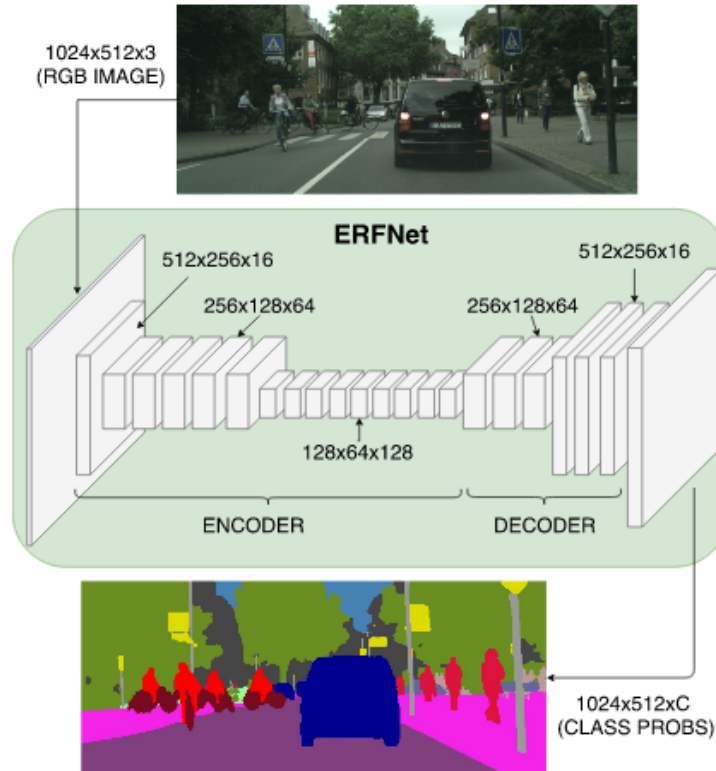


Figure 2.5: Diagram of the ERFNet architecture. Image from the paper [5]

a lot more variable than highway, with really sharp curves and straight elements that may alternate in quick succession. There are roads completely without lane markings, roads with lane markings just on one side and roads with faded or broken lane markings. The lane markings themselves may present different color (white, yellow), pattern (solid, dashed, double), width or even be circular reflector like bott's dots. Parking lot, pedestrian crossing, reserved lanes for special vehicles have also to be taken into account. The surface of the road may also change with different pavement color, roughness or a combination of them.

Besides, the appearance of urban roads is usually more affected by external elements like shadows, parked cars, different types of delimiters, puddles, and potholes. More-

over, all the road characteristics mentioned above may vary hugely not only between different countries but within few kilometers.

Depending on the use case lane detection system may have different aims. The following list summarizes the three main objectives for a lane detection system.

- **Lane Departure Warning Systems:** these systems warn the driver every time the vehicle crosses the ego lane boundary. This approach requires to correctly detect the lane boundary with respect to the vehicle trajectory. Overall this is a relatively easy task since lane boundaries need to be detected close to the vehicle. These systems are mainly used in highway environment.
- **Forward Collision Warning Systems:** these systems need to detect the distance of vehicle ahead. In this case, lane detection algorithms are used to identify vehicles that are driving in the ego-lane correctly. These systems are mainly used in highway environment.
- **Autonomous Driving:** these systems need to identify the whole topology of the road and lanes.

2.2.1 Classical Computer Vision based approaches

Classical computer vision methods have been used since the 1990s to solve the lane detection problem. In case of clearly defined lane marking on the road, this task can be solved quite easily and efficiently using classical computer vision algorithms like edge detection or Hough transform based approaches. Most of the approaches for lane detection present in the literature follow a similar sequence of steps:

- Extract feature using image processing techniques
- Lane model fitting
- Lane tracking

2.2.1.1 Feature extraction

The feature extraction step aims to detect road markings from the camera image. There are essentially two geometric domains in which this step can be realized:

- Perspective view: this is directly the image taken from the camera which preserve the perspective effect.
- Bird's eye view (BEV): this is also called Inverse Perspective Mapping (IPM) and represent a geometric transformation that renders the scene as if seen from above. In this space, lines parallelism is preserved. This transformation assumes that the road is perfectly planar, a hypothesis which is not always satisfied. The major downside of using this domain is the presence of high distortion in case the flat surface hypothesis is not satisfied, or the calibration between the camera and the road has not been correctly estimated. This means that pitch and roll angle variation with respect to the road surface while the vehicle moves have to be compensated in order to mitigate the distortion on the BEV image. An example of bird's eye view is illustrated in figure 2.6. In particular, in the left image (2.6a) is shown a bird's eye view without motion compensation that presents the typical distortion that worsens as the distance from the camera increases. On the other hand in the right image (2.6b) is shown the bird's eye view computed from the same perspective image but using a motion compensated calibration.

Although lane markings may appear in different pattern or color, they usually present a clear contrast with respect to the road pavement. This contrast can be exploited using a gradient-based edge detection approach like Sobel or Canny.

In [32] the author proposed a method that use Canny edge detector to highlight lane markings feature and then a Hough transform that operates at different image resolution has been applied to improve the accuracy of the algorithm.

In [33] the authors presented an algorithm that obtains lane markings from the camera image transformed with an Inverse Perspective Mapping and then a low level filtering is performed to extract Dark-Light-Dark (DLD) pattern. The resulting segments are then clustered together.

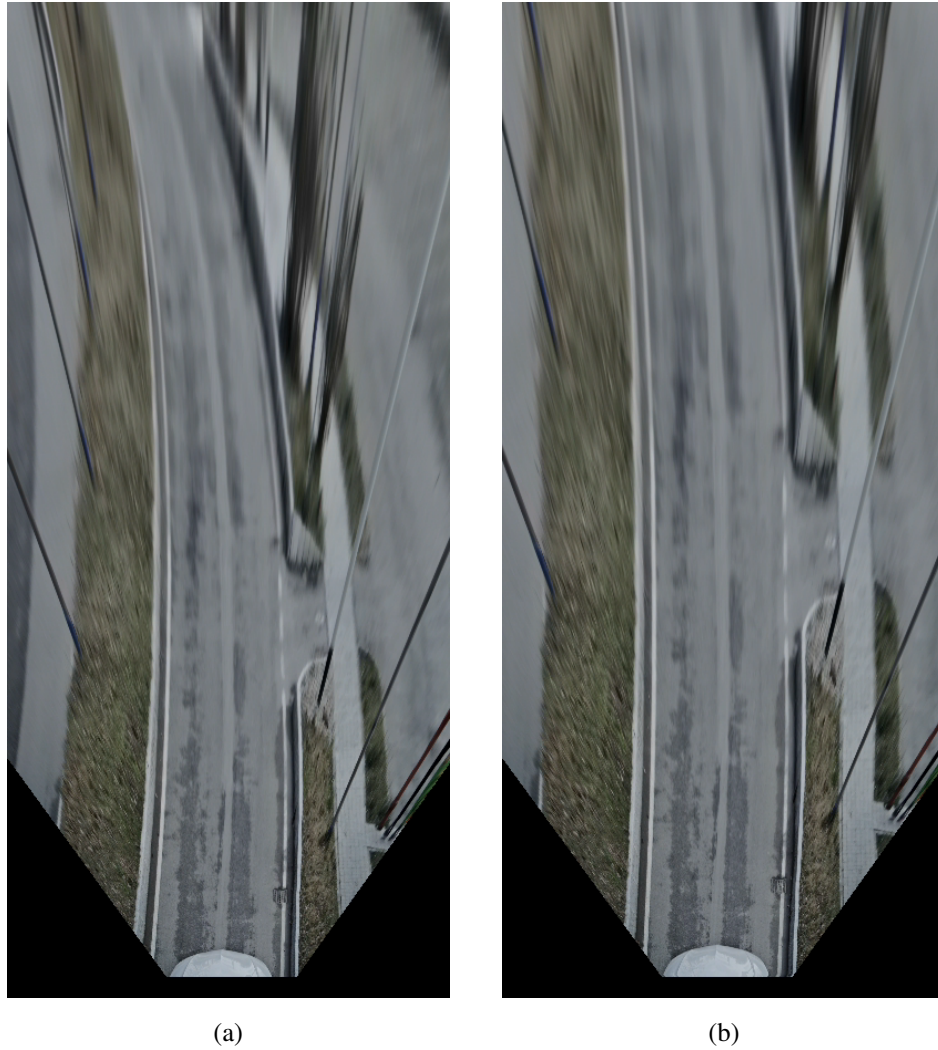


Figure 2.6: Example of bird's eye view images. Figure 2.6a shows the bird's eye view without motion compensation while 2.6b shows the bird's eye view computed using a calibration with compensated motion

The greatest challenge in the feature extraction step is to handle illumination changes and shadow effects. To overcome these issues many authors proposed to process the

image in the Hue-Saturation-Value (HSV) color space [34] [35].

Another approach proposed in [34] assumes marking color and brightness known a priori and learned by a classifier. In this way, image pixels are classified, according to color and brightness probability, as lane markings or background. Positive pixels are grouped in segments and filtered to remove outliers. These are regions in the image that are not lane markings but present similar color and brightness and have to be removed using shape and size based criteria.

2.2.1.2 Lane model fitting

Lane model fitting allows recovering a model for the lane starting from the extracted road markings points. Expressing the lane utilizing a model allows providing a more robust and reliable representation for path planning and localization modules. Different types of model fitting have been proposed over the years ranging from parametric models (line, parabola) to splines. Model fitting is often performed in the bird's eye view domain where the perspective effect is removed, and thus the lane model can be expressed in a more straightforward way.

In case only short range lane detection is required a linear model could suffice to accomplish the task. Examples of lane detection method based on linear model have been proposed in [36] [37] [34].

In [38] the lane is modeled by a parabola, and the fitting is realized using a least square minimization with an outliers removal process based on RANSAC. In [32] a parabola model is also used, but the fitting is performed using Hough transform.

Another parametric model that can be used to represent the lane is the clothoid. This model allows avoiding a sharp change of curvature between straight and curve lane sequences. Examples of these methods can be found in the works of [39] and [40].

Spline models define a different way to express a lane. They do not make a global model assumption, but they parametrize the lane through a series of control points allowing to represent a wider range of curves. Examples of lane detection approach that uses spline models can be found in [41], which uses cubic spline, and in [42] which instead uses B-spline.

2.2.1.3 Lane tracking

Tracking is used to improve the final results in terms of stabilization by integrating information over time and predicting the lane position in the future. This step usually exploits also information from movement sensors like vehicle odometry [43], IMU and GPS [38].

Lane tracking and estimation can be achieved using Kalman filter like in [44] [45] and [36]. Kalman filter based approaches exploit model prediction to guide the detection of the lane that is afterward used to update the state of the filter. The main drawback of Kalman filter based methods is that they can only represent unimodal distribution; thus they cannot handle road discontinuities. In those cases, a track reinitialization is typically required. To overcome this issue in [46] the author proposed to use two instances of the lane model to solve the road discontinuity problem.

Overall particle filtering is a more suited framework for this task since can it handle multi modal distributions representation. This estimation approach is used for example in [41]. Here the author presented a lane detection model based on spline where particle filtering is used to sample and predict the spline control points. Vehicle motion model is also used to update the particles state.

Another work based on particle filtering is [47] where greyscale and stereo disparity images are used to weight particles in the measurement step. Road discontinuities are handled by uniform sampling during particle initialization.

2.2.2 Deep Learning based approaches

The main issues related to classical computer vision approaches is that they rely on handcrafted features that cannot capture all the complexity and unpredictability of a road scenario. On the other hand, deep learning based methods can learn the features automatically at training time. In this way, these methods could potentially handle every possible scenario, road topology, illumination and weather conditions if well represented in the dataset.

One of the first work that tried to solve the lane detection problem using deep learning is [6]. Here lane position estimation is performed using images acquired from

laterally-mounted down facing cameras. A deep neural network process these images and predict, in an end-to-end manner, if there is a lane marking or not and the position of the marking itself. An example of outputs produced by the network is shown in figure 2.7, where it is demonstrated that the method is able to detect lanes even in case of broken or faded markers and in the presence of shadows.

He in [48] instead proposed to feed a CNN with both front camera view and bird's



Figure 2.7: Example of output produced by DeepLanes network. Image from the paper [6]

eye view. The former view is needed to remove vehicles, curbs, and barriers while the latter is used to remove ground arrows and words. In contrast to the previous, this work is not end-to-end but requires preprocessing and post processing steps to output the final lane.

In [7] the author presented a multitask network that jointly predicts the lane, the marking types, the road sign markings and the vanishing point of the scene. The network has been trained on 20000 images that present various challenging conditions like night and rainy conditions. An example of output obtained by VPGNet is shown in figure 2.8.

A different approach is presented in [8] where the ego lane left and right boundaries are predicted in a semantic segmentation framework. The final network is obtained using a transfer learning procedure. Initially the network has been pretrained on a general purpose dataset like ImageNet, then on road scene datasets like CamVid and Cityscapes and finally on a dataset build by the authors specifically for their lane detection task. The main limit of this method is that it can detect only a fixed and pre-

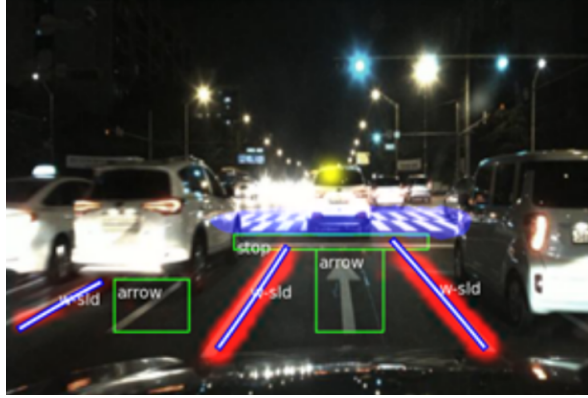


Figure 2.8: Example of output produced by VPGNet. Image from the paper [7]

defined number of lanes and is not able to handle lane change situations. An example of the network output is shown in figure 2.9.

Another approach based on segmentation has been proposed by Pan in [9]. Here the

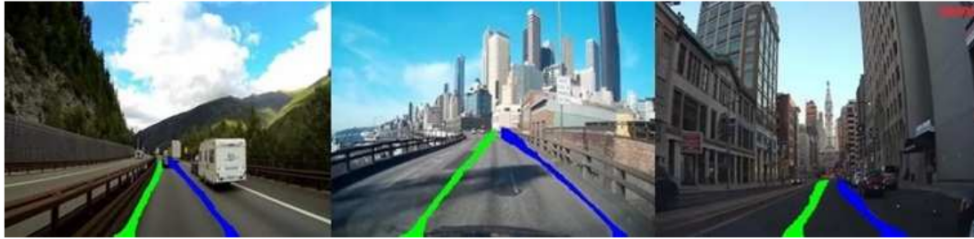


Figure 2.9: Example of the output produced by the network proposed by Kim. Image from the paper [8]

task is to predict left and right ego lane boundaries and left and right lane boundaries when present. The dataset used in this paper contains more than 100000 images collected in China in urban and highway environment in different illumination and weather conditions. In this work, a specific architecture, called Spatial CNN (SCNN), that uses slice-by-slice convolution in four directions within feature maps has been developed. This architecture proved to be particularly suited to detect long and continuous element like lanes. This feature is highlighted in figure 2.10 where is shown the

output of SCNN in comparison with a classical CNN architecture for segmentation. The downside of using semantic segmentation to distinguish between different lanes directly is that the network is constrained on the maximum number of lanes and can't handle lane changes scenarios well. Regardless the network achieved state of the art result on the TuSimple lane detection dataset.

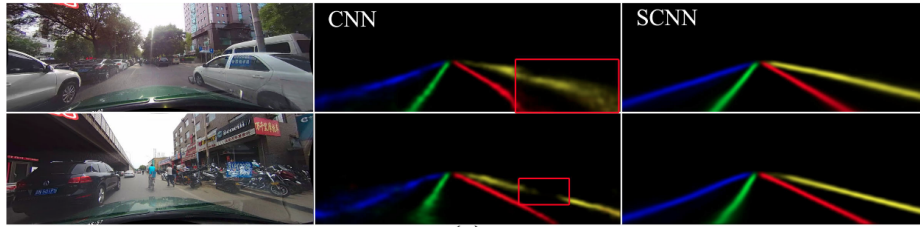


Figure 2.10: Example of output produced by SCNN in comparison with a standard CNN for segmentation. Image from the paper [9]

More recently in [10], Neven proposed an end-to-end approach for lane detection based on instance segmentation. This means that the network predicts each lane as a unique instance. Therefore the number of output lanes is not constrained, and the model can cope with lane changes. More specifically they built a network called *LaneNet* which after a shared encoder initial part is split into two main branches. A segmentation branch, which generates a binary mask for lane/background prediction, and a pixel embeddings branch, which produces an N-dimensional embedding for the lane pixels using a custom loss. The output of the network is then clusterized using an iterative algorithm to obtain the lane instances. *LaneNet* architecture is shown in figure 2.11.

The system also presents a second network, called *H-Net*, which is trained using a novel loss that allows to learning a conditioned perspective transformation that is optimal for a subsequent lane fitting. An overview of the final system is shown in figure 2.12.

The approach has been tested on the TuSimple lane detection dataset achieving competitive results.

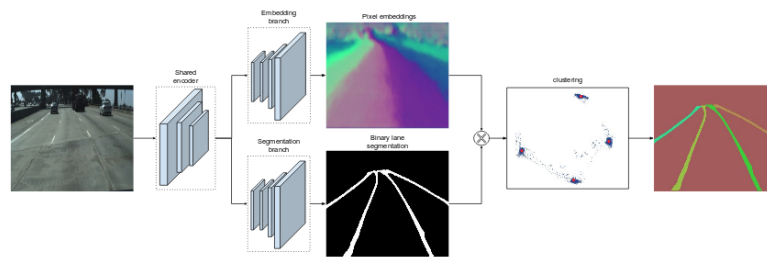


Figure 2.11: Overview of the *LaneNet* architecture. Image from the paper [10]

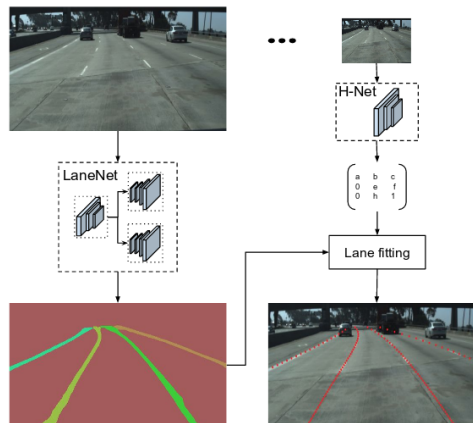


Figure 2.12: Overview of the system proposed by Neven. Image from the paper [10]

Chapter 3

Lane detection and segmentation using convolutional neural network

Reconstructing and recognizing the semantic meaning of the scene is an essential part of autonomous driving in order to allow the vehicle to drive safely in complex and unknown environments. The framework used to solve the segmentation problem is deep learning and more specifically Convolutional Neural Network (CNN).

In particular, this chapter will focus on the methodology used to solve two problems related to perception for autonomous driving namely segmentation of road scenes and lane detection. In the first case, semantic segmentation has been applied to understand the structure and composition of road scenes. The dataset chosen for this task is Cityscapes [27] which provide dense high-quality segmentation for almost 3000 images collected in urban streets.

For the lane detection problem, an approach based on instance segmentation has been preferred. Instance segmentation, which is a more general problem that combines semantic segmentation and object detection, provides not only dense pixel labeling of the scene but also a unique label to different instances of the same class.

The dataset used to train the model for lane detection are the TuSimple Lane Detection

Benchmark [49], which contains almost 3000 images acquired on US highways, and BDD100K [50], which contains 100000 road video clips collected in a wide variety of conditions.

The chapter is organized as follows: initially, a brief description of the used dataset will be given, after that the network modules and architectural choices will be explained. Following the loss functions and training procedures will be described, and at the end, the final section is devoted to present the network optimizations.

3.1 Datasets

The datasets used to evaluate the work developed in this thesis are popular benchmarks for the task of semantic segmentation of urban road scenes (Cityscapes) and lane detection (TuSimple lane detection benchmark and BDD100K).

A brief description of these two datasets is given in the following paragraphs.

3.1.1 Cityscapes

Cityscapes is a freely available dataset that focuses on the semantic understanding of urban road scenarios. The data has been collected in 50 different cities in Germany at daytime in clear and cloudy weather conditions. The dataset is a benchmark for pixel and instance semantic segmentation model and contains 5000 images with fine pixel-level annotations and 20000 additional images with coarse annotations. An example of fine and coarse annotated images is shown in figure 3.1.

The dataset presents a total of 30 different annotated classes grouped in eight macro categories which are flat, construction, nature, vehicle, sky, object, human and void. Instance-level segmentation is annotated for eight different classes including cars, motorcycles, and pedestrians. The 5000 fine annotated images are split in 2975 for training, 500 for validation and 1525 for testing. The split is done at city level meaning that all images collected in a city are placed in a single set.

The authors of the datasets also proposed a metric for the evaluation and benchmark of the pixel-level segmentation and instance-level segmentation tasks. For the first

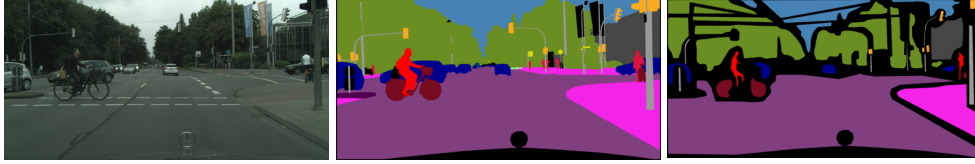


Figure 3.1: Example of an image (left figure) with relative fine (central figure) and coarse (right figure) annotations taken from the Cityscapes dataset

task to assess labeling results is used an intersection-over-union (IoU) score (equation 3.1) [24].

$$IoU = \frac{TP}{TP + FP + FN} \quad (3.1)$$

Where TP, FP, FN are respectively the number of true positive, false positive and false negative pixels. In the benchmark, the score is computed both for classes and categories.

In the case of instance segmentation, the metrics proposed is the average precision on the region level (AP) for each class. AP is computed as the area under the precision-recall curve [51].

3.1.2 TuSimple lane detection

The TuSimple lane detection dataset contains 3626 video clips for training, composed by 20 frames of which the last one is annotated, and 2782 video clips for testing.

On each training image is annotated the ego lane right and left boundaries, the left and right lanes and an additional one in case of lane change maneuver, so the lane number may vary from 2 to 5. The annotation is expressed as a polyline that indicates for each lane the horizontal positions in image coordinates at a fixed number of discretized vertical positions.

The image sequences have been acquired on highway roads in the United States with different traffic conditions at different daytimes in good and medium weather conditions. An example of the dataset images with relative annotations is given in figure 3.2.

The dataset was released for the CVPR 2017 Workshop on Autonomous Driving

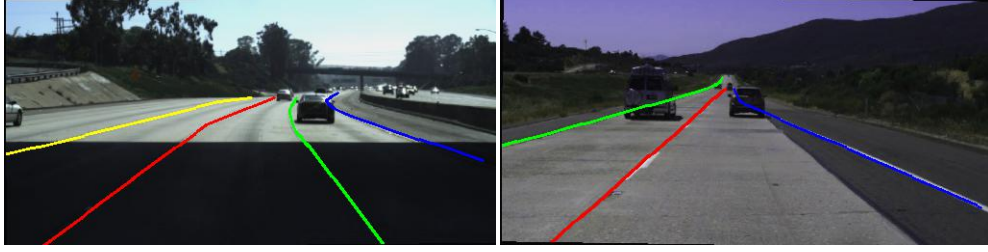


Figure 3.2: Example of images with annotations taken from the TuSimple lane detection dataset

Challenge. For the evaluation is used an accuracy metric acc that is calculated as the average ratio between the number of correct points C_p and number of ground-truth points S_p 3.2. A point is considered correctly identified when the difference between the horizontal coordinates of the predicted and the ground-truth points is below a certain threshold.

$$acc = \sum_p \frac{C_p}{S_p} \quad (3.2)$$

In addition to the accuracy metric a false positive FP and false negative FN rate is also provided. False positive rate is the ratio between the number of lane predicted but not matched with any ground-truth lane F_{pred} and the total number of predicted lanes N_{pred} 3.3. False negative rate instead is the ratio between the number of lane in the ground-truth but not matched with any predicted lane M_{pred} and the total number of predicted lanes N_{gt} 3.4.

$$FP = \frac{F_{pred}}{N_{pred}} \quad (3.3)$$

$$FN = \frac{M_{pred}}{N_{gt}} \quad (3.4)$$

3.1.3 BDD100K

The BDD100K, released by the Berkeley DeepDrive Industry Consortium, is the largest and most diverse dataset with annotations for autonomous driving. The dataset presents 100000 video clips of about 40 seconds length recorded at 30 fps with a resolution of 720p. For each video, the frame at the 10th second is selected and annotated. The videos have been acquired in different locations of the United States like New York, San Francisco, and Silicon Valley and present different scenarios like highways, urban streets, rural streets, and residential areas. Besides the videos have been recorded in a wide variety of weather conditions (sunny, rainy, cloudy, snowy) and illumination conditions (daytime, nighttime, dusk, dawn).

The dataset provides annotations for the training and testing of many different tasks like full-frame dense semantic segmentation, road object detection, lane marking detection and drivable area segmentation.

Annotated lane markings are divided into two type: vertical lane markings, which indicate markings of the lanes along the driving direction, and parallel lane markings, which defines stop lines. The pattern (single or double, solid or dashed) for the markings is also provided. An example of images with lane markings annotations is shown in figure 3.3.

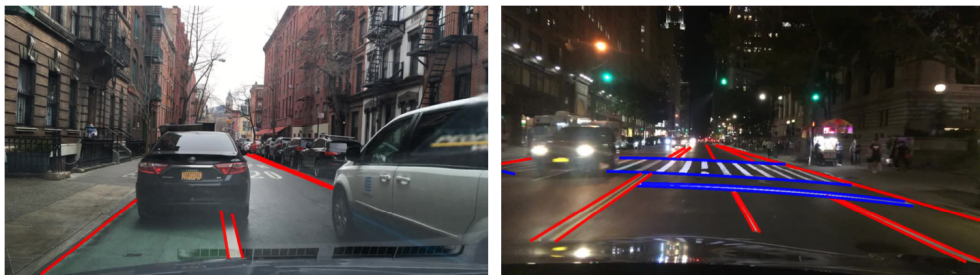


Figure 3.3: Example of images from the BDD100K dataset with lane markings annotations.

Drivable areas annotations are divided into two different categories based on the position of the ego vehicle. The first one has been called *direct drivable* and represents the area that can be driven safely by the ego vehicle. The second one, called *alternative drivable*, is the area where the ego vehicle could drive but other vehicles could be driving on it. An example of images with drivable area annotations is shown in figure 3.4.

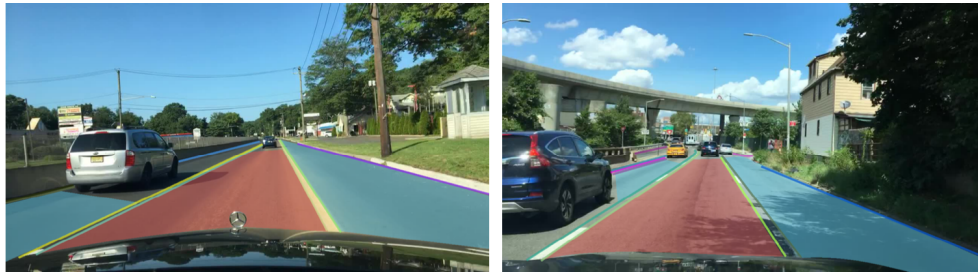


Figure 3.4: Example of images from the BDD100K dataset with drivable area annotations.

Using drivable areas and lane markings annotations a semiautomatic tool has been developed to obtain the annotation in the same format as the TuSimple Lane detection dataset, which is the boundaries of the main lanes on the road. The number of images obtained was around 8000 for training and 2000 for testing.

3.1.4 Lanes annotation

In this section is described in detail the way the lanes are annotated with particular focus on more complex cases. When all present the annotated lanes are four: ego lane left boundary, ego lane right boundary, left lane boundary, right lane boundary. These are depicted in figure 3.5.

In the case of road intersection, no lanes are annotated. The reason behind this choice is that in those cases, given a single frame for the detection, the direction of the ego vehicle is unknown. Some example of intersection scenario are shown in figure 3.6.



Figure 3.5: Examples of lane detection with all four lanes boundaries: ego lane left (green), ego lane right (red), left lane (blue), right lane (yellow). On the left are displayed the original images while on the right are displayed the annotated images.

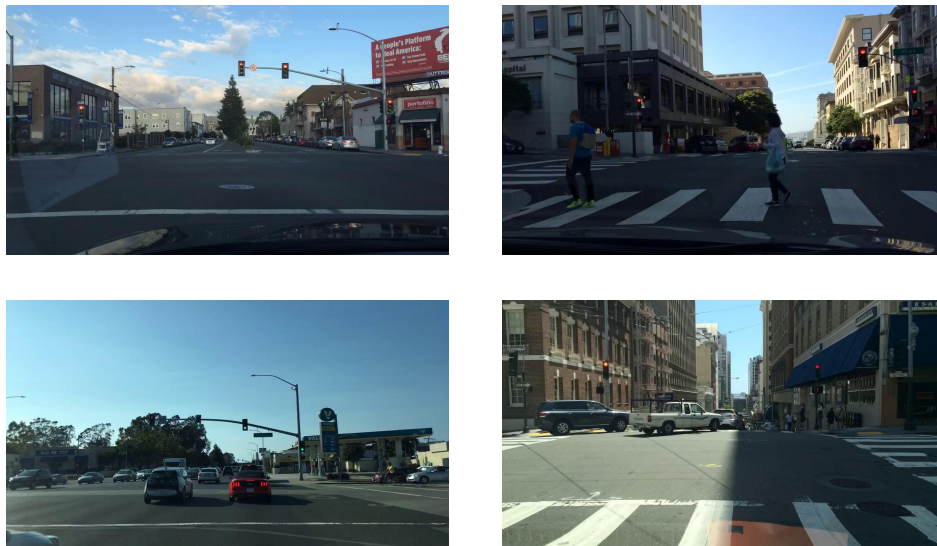


Figure 3.6: Examples of intersection scene where lanes are not annotated.

More challenging cases are represented by roads completely without markings or partially without markings on certain sides of the street. In those cases, the annotation of the lanes is based on road boundary delimiters like curbs or parked cars. For some roads also the width of the lane has to be considered in order to annotate the road as single lane or double lanes. In most of these situations defining the lanes can be really hard even for a human. In figure 3.7 are shown some images of roads with partial markings while in figure 3.8 are shown some images of roads without markings.



Figure 3.7: Examples of road images with partial markings. On the left are displayed the original images while on the right are displayed the annotated images.



Figure 3.8: Examples of road images without markings. On the left are displayed the original images while on the right are displayed the annotated images.

3.2 Model architectures

Convolutional Neural Networks are able to discover complex semantic and structure in the input data without requiring hand-engineered features. One of the tasks left to the researchers is the design of the network architecture. In particular for segmentation networks in recent years almost every proposed architecture followed the encoder-decoder scheme.

The encoder is used to extract and build a hierarchy of features from the input image into lower resolution feature maps. The decoder instead is used to restore the context information from the lower resolution maps and recover fine-grained details with a learnable upsample. In this section, the model architecture will be described starting from the basic layers to the whole network structure.

3.2.1 Networks layers

This section will briefly describe the layers that are present in the final models.

3.2.1.1 Convolution

This is the basic layer in every CNN and it implements the convolution operation. This convolution is usually performed over a 3D input tensor of dimension $D_i \times H_i \times W_i$ by convolving it with a kernel that is a 4D tensor of dimensions $D_o \times D_i \times H_k \times W_k$ to obtain an output tensor of dimension $D_o \times H_o \times W_o$.

A transposed convolution layer performs a regular convolution but reverts its spatial transformation. This layer is sometimes referred also as deconvolution layer or fractionally strided convolution layer. This type of convolution is used as learnable upsample layer opposed to non-learnable upsample layers that use some form of interpolation. For this reason, this convolution is widely used in the decoder module of segmentation networks.

Dilated convolution is a particular type of convolution that presents an additional parameter called the dilation rate. This parameter defines the spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 has an equal receptive field of a 5x5 kernel, while still using nine parameters. This dilation provides a larger receptive field without increasing the computational cost. Dilated convolutions are particularly popular in semantic segmentation or dense prediction networks. The concept of dilated convolution, using a kernel of size 3x3 and different dilation rates, is illustrated in figure 3.9.

Another common type of convolution is the depthwise convolution which is widely used in the popular MobileNet architecture [52]. With depthwise convolution each

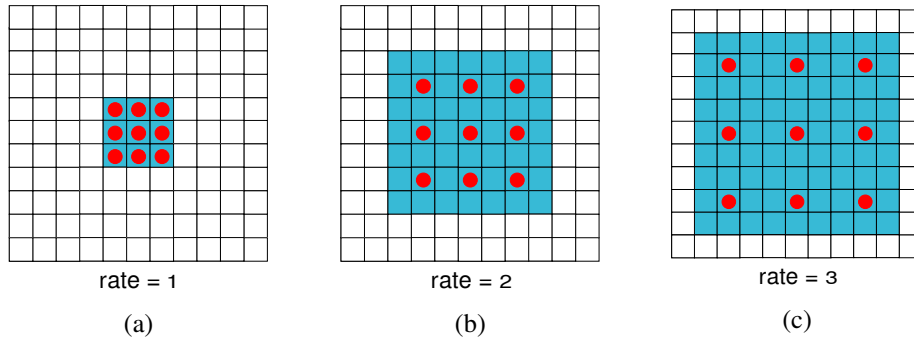


Figure 3.9: An example of dilated convolution using a kernel of size 3x3. The colored cells represent the receptive field while the red dots indicate the positions of the kernel elements with rate of 1,2,3 shown respectively in figure 3.9a, 3.9b and 3.9c

output channel is the result of the convolution of only one input channel.

3.2.1.2 Pooling

The aim of a pooling layer is to reduce the size of the feature map and consequently reduce the number of parameters and the amount of computational operation. This works independently for every depth layer applying a sliding window over the height and width dimensions of the feature map. Different kind of operation can be performed over the window values and this operation defines the type of the pooling layer. The most common pooling layers types are max pooling and average pooling. As the name suggests, max pooling keeps the maximum value inside the sliding window, while average pooling keeps the average value inside the window.

3.2.1.3 Batch Normalization

Batch Normalization was introduced in [53] by Ioffe and Szegedy to help the convergence of deep neural network by reducing the internal covariate shift phenomenon. During training, the data distribution of the input of one layer is affected by the parameters update of all the previous layers requiring the network layers to adapt to different

distributions continuously. This requires the use of lower learning rate values especially in very deep networks to avoid the problem of vanishing or exploding gradients. To improve the stability of a neural network, at every iteration batch normalization normalizes the output of a layer before the activation function using the mean μ_B (equation 3.5) and variance σ_B^2 (equation 3.6) of a mini-batch of size m .

$$\mu_B = \frac{1}{m} \sum_i x_i \quad (3.5)$$

$$\sigma_B^2 = \frac{1}{m} \sum_i x_i - \mu_B^2 \quad (3.6)$$

Using this procedure every output layers $\hat{x}_i$ will have zero mean and unitary variance.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (3.7)$$

However these shift plus rescale operation does not ensure optimality so Batch Normalization introduces two additional learnable parameters γ and β that allows respectively to rescale and shift again the output map y_i (equation 3.8).

$$y_i = \gamma \hat{x}_i + \beta \quad (3.8)$$

Batch Normalization allows the use of higher learning rate values and makes the model less sensitive to weight initialization. Also, it acts as a regularizer helping the network to generalize better and to avoid overfitting.

3.2.1.4 Dropout

Dropout [54] is another type of layer that helps regularization and prevent overfitting during training. At each iteration steps it randomly drops (sets to zero) with probability p neuron connections forcing the network to learn the same concept using different subsets of the network. In this way, the features learned by the network are more robust giving better results at test time.

In the special case of convolutional neural network the dropout module has a probability to disables each different features maps in a network layer.

3.2.1.5 Activation function

Activation functions are non-linear mapping responsible for the activation of neurons in the network given a certain input signal.

The most popular activation function for deep neural networks in recent years is the Rectified Linear Unit (ReLU). Proposed in [14] this function allows to obtain better performance, helps to avoid the vanishing gradient effect during training and speed up the model convergence. The ReLU activation function $f(x)$ is shown in equation 3.9.

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (3.9)$$

Other possible choices for the activation function could be the sigmoid function, the arctangent function and other ReLU variants which model the interval $x < 0$ in different ways instead of fixing the values to 0.

3.2.2 Networks blocks

The architecture of a network is a modular structure in which certain modules tend to be repeated through the network. This blocks typically present the same layer composition and connectivity but are configured in slightly different ways, like the number of channels. The blocks developed in this thesis are based on the residual module introduced by He with the ResNet architecture [12] which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2015. The concept of residual representation has been further studied and extended in [55] [56].

In the paper the author demonstrates that it was possible to successfully train very deep networks (up to 152 layers) using residual modules. Until then the main issues related to the train of very deep networks were caused by the vanishing gradient phenomenon. The fundamental insight of residual networks is that for a stack of layers is easier to fit a residual mapping rather than an unreferenced mapping. The original problem of learning a mapping $y = f(x)$ has been re-casted to learning a mapping $y = f(x) + x$. In this way if the identity mapping is optimal it would

be easier to push the residual part $f(x)$ directly to zero than to fit an identity function through a series of non-linearity. The residual block graph is shown in image 3.10.

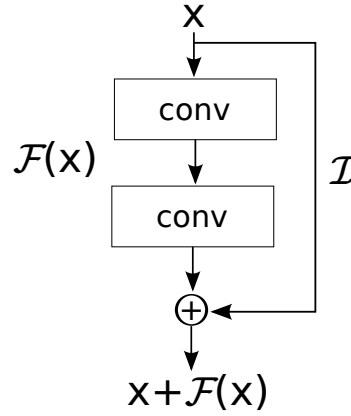


Figure 3.10: Residual block.

The aim of the residual blocks implemented in this thesis is to reduce the required computation by using specific structures like bottlenecks, dilated convolutions, and depthwise convolutions.

3.2.2.1 Residual bottleneck

The main branch of the residual bottleneck is composed of three consecutive convolutions. The first is a 1×1 convolution that reduces the number of channels by a factor t . In this way, the second convolution which is a 3×3 operates on a lower dimensional space hence reducing both parameters and computation times. This 3×3 convolution can be regular or dilated. The third convolution uses a 1×1 kernel again and it is used to restore the original number of channels.

Each of those three convolutions is followed by a batch normalization layer and the first two also by a ReLU nonlinearity. The resultant feature maps are summed with the input and in the end a ReLU activation function is applied to the output.

The graph of the residual bottleneck is shown in figure 3.11.

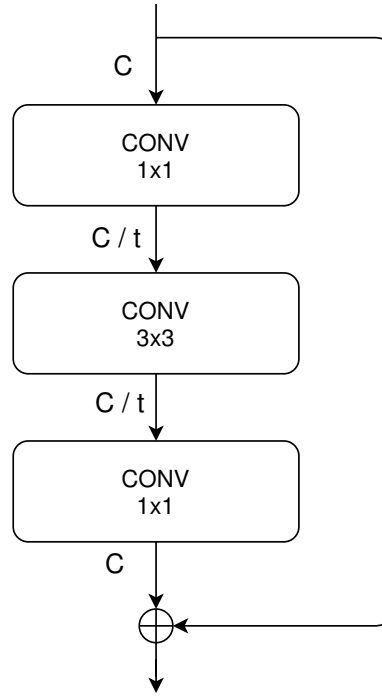


Figure 3.11: Graph of the residual bottleneck module. C indicates the number of input channels while refers t to the reduce factor.

3.2.2.2 Inverted residual

The inverted residual block has been proposed for the MobileNetv2 architecture in [57]. This module is composed of three consecutive convolutions. The first one is a 1×1 convolution that expands the number of channels by a factor t . The second is a depthwise convolution which operates on a higher dimensional space since applies convolution only on one input channel for each output channel. This convolution uses a 3×3 kernel and can be dilated.

Finally, another 1×1 convolution is used to reduce the number of channels to the initial value. These two convolutions are followed by a batch normalization layer and the first one also by ReLU nonlinearity. The final result is added to the input and a

ReLU activation function is applied to the output. The graph of the inverted residual is shown in figure 3.12.

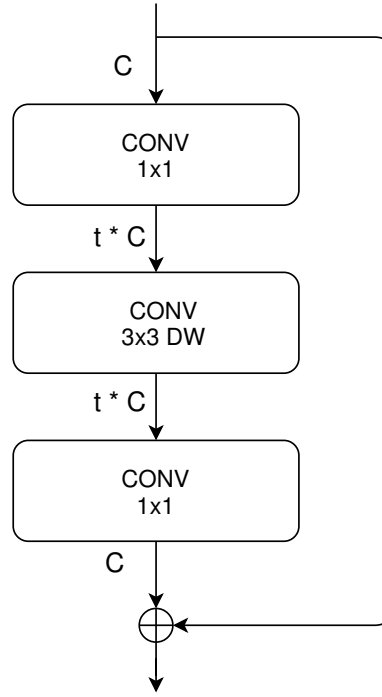


Figure 3.12: Graph of the inverted residual module. C indicates the number of input channels while refers t to the expand ratio.

3.2.2.3 Residual dilated pyramid

The residual dilated pyramid block build a pyramid of feature maps using different receptive field sizes (i.e., different dilation rates) to capture the spatial context at multiple scales. The structure of the dilated pyramid is inspired by the Efficient Spatial Pyramid module presented in [58].

The module is composed of two parts. The first part is a factorized convolution (1x3, 3x1) followed by a batch normalization layer and a ReLU activation function.

The second part is the pyramid of dilated convolution and is composed of three factorized convolution (1×3 , 3×1) where each one of these process one different third of the feature maps channels. The convolutions use different dilation rates. Given a base dilation value d the rates in the pyramid are based on the following scheme: d , $2d$, $3d$. The values tried for d have been 1,2,3,4 in different encoder architectures. The three feature maps that result from these convolution operations are summed hierarchically, before being concatenated, to avoid gridding artifact caused by different receptive fields. The feature maps obtained are then concatenated and passed through a batch normalization, summed with the input and finally a ReLU nonlinearity is applied. The graph of the residual dilated pyramid is shown in figure 3.13.

3.2.3 Residual bottleneck based architecture

The architecture based on residual bottleneck (RBTNet) modules presents an encoder-decoder design where the decoder fuse the encoder feature extracted at different resolutions.

The encoder presents three main blocks that are composed by a stack of residual bottleneck modules. Each encoder block is preceded by a downsampler block which halves the current feature maps resolutions but double the number of channels. The downsampler block is a residual block where the identity branch uses max pool layer to downsample the feature map while the residual branch, to reduce the resolution, uses a regular 3×3 convolution with stride 2. The total downsample ratio at the end of the encoder is 8x.

The encoder blocks present the following configuration:

- Block 1: 5 residual bottleneck all with a dilation rate of 1
- Block 2: 3 residual bottleneck with a dilation rate respectively of 1,2,3
- Block 3: 3 residual bottleneck with a dilation rate respectively of 1,2,3

The architecture uses skip connections to fuse in the decoder feature maps coming from different encoder intermediate layers. The network has 0.4 million param-

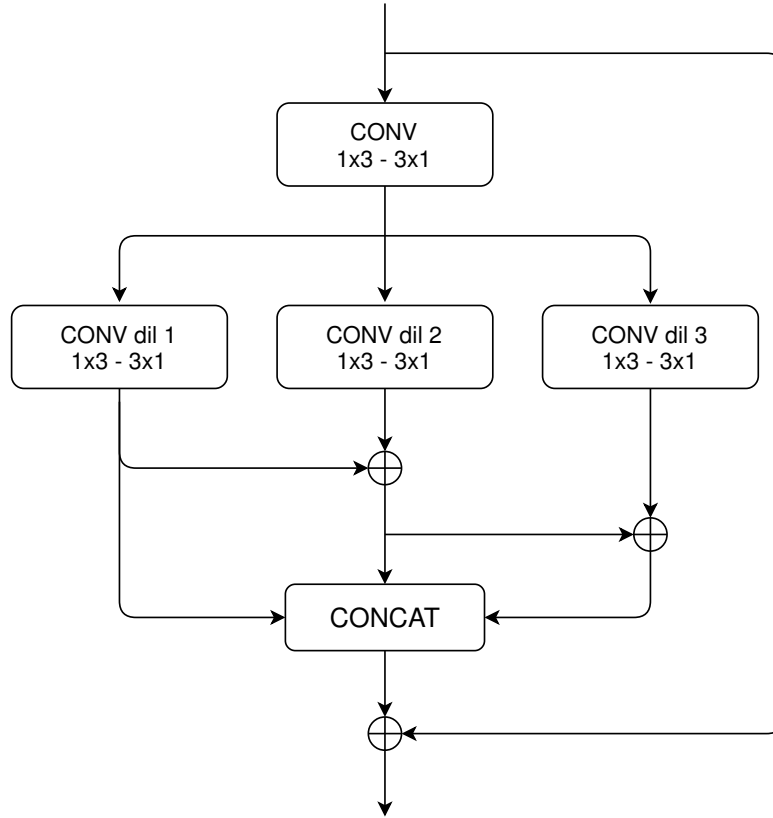


Figure 3.13: Graph of the residual dilated pyramid module.

ters and requires 1.2 billion multiply-add (MADD) operations at inference time. A representation of the full architecture is shown in figure 3.14.

3.2.4 Inverted residual based architecture

The network based on inverted residual modules (IRNet) presents an architecture similar to the previous one with a decoder that fuses feature maps extracted from the encoder at different resolutions.

The encoder is made of six inverted residual blocks with the first and the second

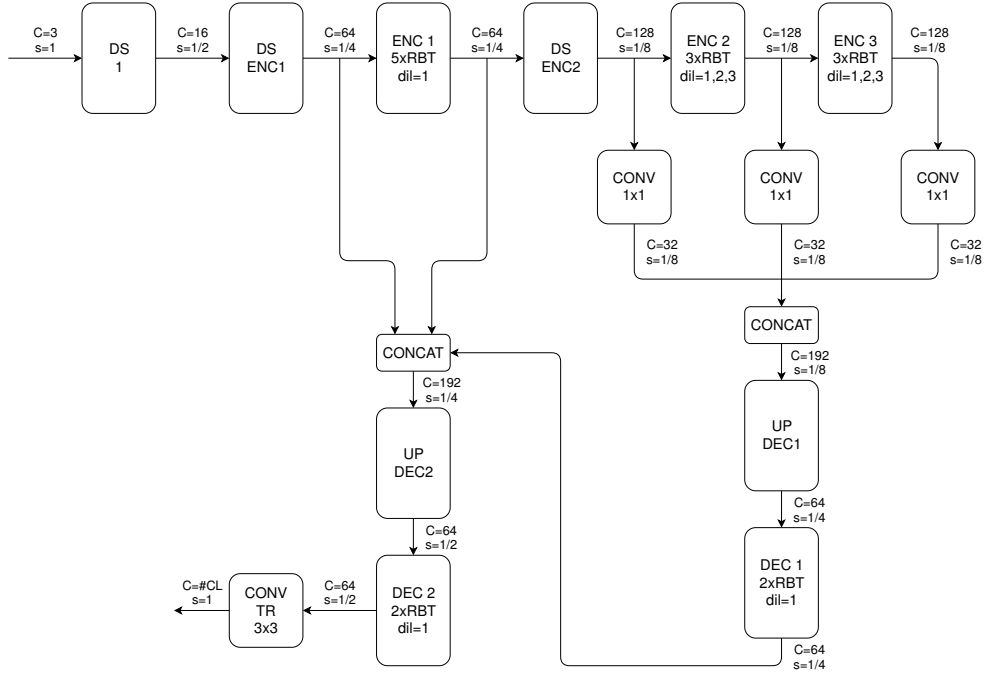


Figure 3.14: Residual bottleneck based architecture (RBTNet). DS indicates the down-sampler block and RBT the residual bottleneck module. In the connections between blocks C represents the size of the input channel while s indicates the downsample ratio of the feature maps.

block that operates a downsample in the feature maps. Before the stack of inverted residual blocks the first downsample is performed using a regular 3x3 convolution with stride 2. The total downsample ratio at the end of the encoder is 8x.

The encoder uses dilated convolution to increase the receptive field. More specifically inverted residual block 1 and 2 have no dilation while inverted residual block 3,4,5 and 6 have a dilation rate of respectively 2,4,8 and 16.

The architecture uses skip connections to fuse in the decoder feature maps coming from different encoder intermediate layers. The network has 1.2 million parameters and requires 4.5 billion multiply-add (MADD) operations at inference time. A representation of the full architecture is shown in figure 3.15.

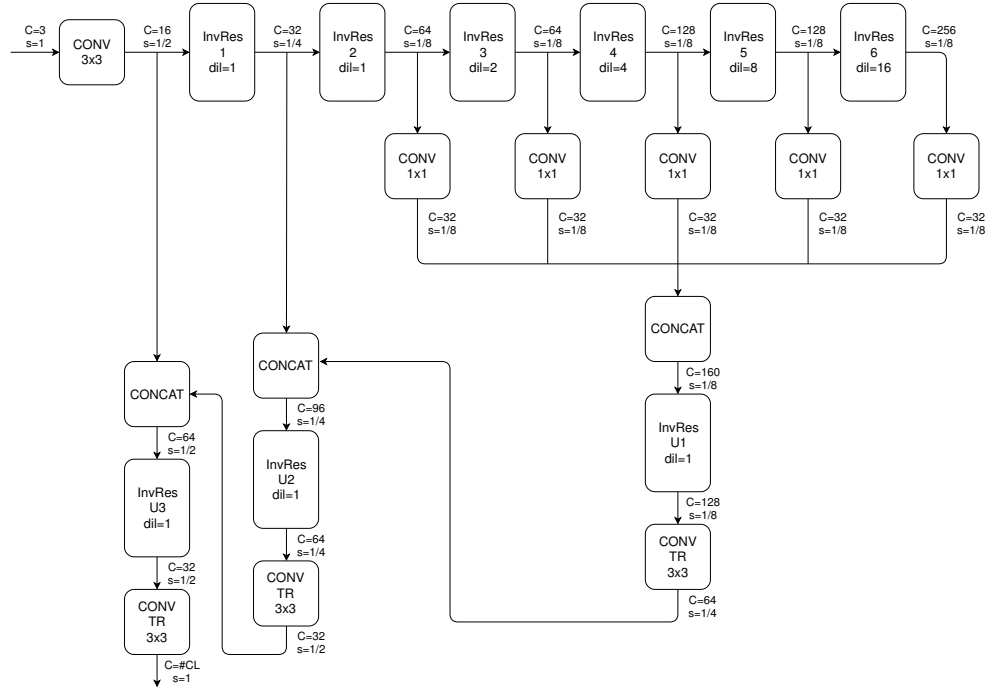


Figure 3.15: Inverted residual based architecture (IRNet). In the connections between blocks C represents the size of the input channel while s indicates the downsample ratio of the feature maps.

3.2.5 Residual dilated pyramid based architecture

The network based on the residual dilated pyramid (RDPNet) follows a classical encoder-decoder architecture without skip connections between the two modules.

The encoder is divided into three main parts composed by multiple residual dilated pyramid modules. The first encoder block is preceded by two downsampler block while the second encoder block is preceded by a single downsampler block. The downsampler block is the same as the one described for the residual bottleneck architecture. The total downsample ratio at the end of the encoder is 8x.

The encoder blocks present the following configuration:

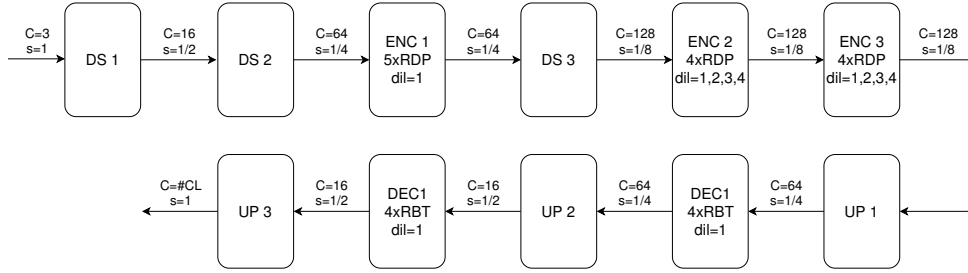


Figure 3.16: Residual dilated pyramid based architecture (RDPNet). DS indicates the downsampler block, RDP defines the residual dilated pyramid block and RBT the residual bottleneck module. In the connections between blocks C represents the size of the input channel while s indicates the downsample ratio of the feature maps.

- Block 1: 5 residual bottleneck all with a dilation rate of 1
- Block 2: 4 residual bottleneck with a dilation rate multiplier respectively of 1,2,3,4
- Block 3: 4 residual bottleneck with a dilation rate multiplier respectively of 1,2,3,4

The network has 1.8 million parameters and requires 6.1 billion multiply-add (MADD) operations at inference time. A representation of the full architecture is shown in figure 3.16.

3.3 Loss functions

Loss functions measures how well the model is performing in terms of prediction on the training set. In the following paragraphs are described the different loss functions used to train the networks presented in this thesis.

3.3.1 Cross Entropy loss

The most commonly used loss function in semantic segmentation is the cross entropy loss. This loss, also called log loss, measure the performance of a model whose output represents a probability value. It is widely used as a loss function for classification model and in the case of segmentation task it can be applied to measure how each pixel prediction differs from the actual pixel label. In other words for segmentation networks the final loss is the sum of the cross entropy loss of every pixel in the final prediction image.

The cross entropy loss can be used for binary prediction or multi-class prediction. In the case of binary classification the formula of the cross entropy loss is shown in equation 3.10.

$$L = -(t \log(p) + (1 - t) \log(1 - p)) \quad (3.10)$$

Where t is the target value and p represent the probability of the model output. To force the output to be in the range of 0 and 1 a sigmoid function must be used as a final layer of the network.

In the case of multiclass classification problem the cross entropy loss can be computed using equation 3.11.

$$L_c = -x_c + \log \left(- \sum_j \exp(x_j) \right) \quad (3.11)$$

Where x_c represent the probability predicted for the true class. In this case to obtain as output a probability distribution over the classes a softmax function must be used as the last layer of the network. The softmax function $s_c(x)$ for a class c is shown in equation 3.12.

$$s_c(x) = \frac{e^{x_c}}{\sum_j e^{x_j}} \quad (3.12)$$

A common problem for segmentation task is that the class distribution over the dataset is often imbalanced. When the segmentation process present this kind of class imbalance the final model is likely not to perform optimally.

For example in the case of lane detection, the first class are the lane markings and the second is everything else in the image. In the case of road scene segmentation classes like sky and road are highly represented while classes like pedestrian and pole present significantly lower frequency. To overcome this problem every pixel contributions to the final loss should be weighted according to the corresponding class frequency. Low frequency classes will have higher weights while high frequency classes will have lower weights. A common way to compute class weight w_c , proposed in [31], is shown in equation 3.13.

$$w_c = \frac{1}{\ln(c + p_c)} \quad (3.13)$$

Where c is a hyper-parameter which has been set to 1.02 and p_c is the probability of class c .

3.3.2 Dice loss

Dice loss proposed in [59] is a loss specifically targeted to solve the problem of class unbalance. This loss can be applied only in the binary segmentation case. Usually this kind of segmentation presents a great imbalance between the number of foreground and background pixels.

Minimizing this loss, which is defined in equation 3.14, is similar to directly minimize the Intersection Over Union (IoU) score.

$$L_d(x) = \frac{2 \sum_i p_i t_i}{\sum_i p_i^2 + \sum_i t_i^2} \quad (3.14)$$

In the formula p_i indicates the predicted output for pixel i while t_i indicates the target value for pixel i .

This formulation of the objective function does not need any re-weighting procedure since it inherently avoid prediction heavily biased towards background.

In general cross-entropy gradient are nicer while dice loss gradient could lead to some instability during training. In the tests carried out, semantic segmentation networks for lane detection have been trained using both cross-entropy and dice loss. Networks trained using dice loss showed a higher segmentation accuracy and a faster convergence time.

3.3.3 Instance segmentation for lane detection

The loss used for the instance segmentation of lanes has been proposed in [60]. The approach is limited to a single category and a restricted number of instances which is fine in case of lane detection since the possible number of lanes in an image varies from 2 to 5.

The key insight of this method is that pixels belonging to the same lane instance should have a similar output distribution while the opposite should happen for pixels of different instances. To measure the probability distribution distance is used the Kullback-Leibler divergence. The term of the loss function for pixels distributions p_i, p_j that belong to the same instance is defined as in equation 3.15.

$$L_s(p_i, p_j) = p_i \log \left(\frac{p_i^*}{p_j^*} \right) + p_j \log \left(\frac{p_j^*}{p_i^*} \right) \quad (3.15)$$

p_i^* and p_j^* indicates that the distribution is assumed to be constant.

For pixels belonging to different instances the loss term is similar except that a hinge loss $L_h(x, \sigma)$ is applied to the KL divergence (equation 3.16).

$$L_d(p_i, p_j) = L_h \left(p_i \log \left(\frac{p_i^*}{p_j^*} \right), \sigma \right) + L_h \left(p_j \log \left(\frac{p_j^*}{p_i^*} \right), \sigma \right) \quad (3.16)$$

The hinge loss L_h is defined in 3.17.

$$L_h(p, \sigma) = \max(0, \sigma - p) \quad (3.17)$$

The two loss term can be combined to obtain the loss $L_I(p_i, p_j)$ (equation 3.18).

$$L_I(p_i, p_j) = l_{ij} L_s(p_i, p_j) + (1 - l_{ij}) L_d(p_i, p_j) \quad (3.18)$$

The indicator function $l_{ij} = 1$ when pixels i, j belong to the same instance and $l_{ij} = 0$ when pixels i, j belong to different instance.

Given the pairwise nature of the loss its complexity is quadratic with respect to the number of pixels of the foreground (lanes in this case). For this reason a sampling method has to be adopted to reduce the amount of required computation.

The global loss term for instance segmentation is expressed in equation 3.19.

$$L_I = \frac{1}{|S|} \sum_{i,j \in S} L_I(p_i, p_j) \quad (3.19)$$

S is set of sampled pixels, the number of pixels in S has been fixed to 200.

Pixels of the background (i.e. non lane pixels) have to be considered as well for the final loss. To include them a specific element in the output vector, which defines the probability of background, has to be added. In this way a cross entropy like loss can be used to classify pixels of the background and foreground where the probability of foreground is given by summation of the non background element of the output vector (3.20).

$$L_{bg} = -\frac{1}{N} \sum_i^N \left(b_{ij} \log p_{i0} + (1 - b_{ij}) \log \left(\sum_{k=1}^n p_{ik} \right) \right) \quad (3.20)$$

Where b_{ij} is an indicator function defined as $b_{ij} = 1$ if pixel ij is background and $b_{ij} = 0$ otherwise.

An additional unary term has been used to penalize false positive in order to keep the predicted lanes thin and avoid that they assume a blob shape especially in the far range 3.21.

$$L_{fp} = \alpha \frac{1}{N} \sum_i^N \left(b_{ij} \sum_{k=1}^n p_{ik} \right) \quad (3.21)$$

Where α is a hyper-parameter that balance the false positive penalty weight.

To summarize the final loss function can be written as 3.22.

$$L = L_I + L_{bg} + L_{fp} \quad (3.22)$$

3.3.4 Curve fitting loss for lane detection

The output of a lane detection segmentation network can be used as a feature extractor for the subsequent steps of model fitting or tracking.

The main drawback of these methods is that the fitting and tracking is carried out in bird's eye view space, hence relying on the hypothesis of perfectly flat terrain. This leads to erroneous results in the presence of uphill and downhill, and when the

points are far from the vehicle.

In order to solve this problem, following the work in [10], an additional CNN has been trained to obtain a homography conditioned on the input. This homography identifies an ideal plane where the fitting procedure is easier.

The homography network receives as input the original image and outputs six values which represent the six degrees of freedom of a homography transformation. More specifically has been observed that to successfully train the network the biases in the last linear layer have to be initialized with a homography estimated through vehicle-road extrinsic calibration.

The homography network is a simple architecture build using consecutive blocks of 3x3 convolutions, batch normalization and ReLU. The input undergoes a 8x down-sample and the last two layers are fully connected.

The homography is applied to the lane points $\mathbf{p}_i = [x_i, y_i]$ detected with an instance segmentation network obtaining a set of transformed point $\hat{\mathbf{p}}_i = [\hat{x}_i, \hat{y}_i]$. These points are then fitted using a least-squares closed-form solution for an n -degree polynomial. In this case n has been fixed to 3.

The fitted points $\hat{\mathbf{p}}_i^* = [\hat{x}_i^*, \hat{y}_i^*]$ are then projected back to points in the image space $\mathbf{p}_i^* = \mathbf{H}^{-1}\hat{\mathbf{p}}_i^* = [x_i^*, y_i^*]$.

The homography network is trained using a custom loss that minimize the distance between the reprojected fitted points and the original detected ones (equation 3.23).

$$L = \frac{1}{N} \sum_i (x_i^* - x_i)^2 \quad (3.23)$$

In figure is shown a comparison between the lanes fitted using a classic homography transformation from the camera calibration and the lanes fitted using the conditioned homography from *H-Net*.

3.4 Training

The process of training allows a model to learn a particular task given a certain amount of data. The training adjusts the model weights in order to minimize a loss function

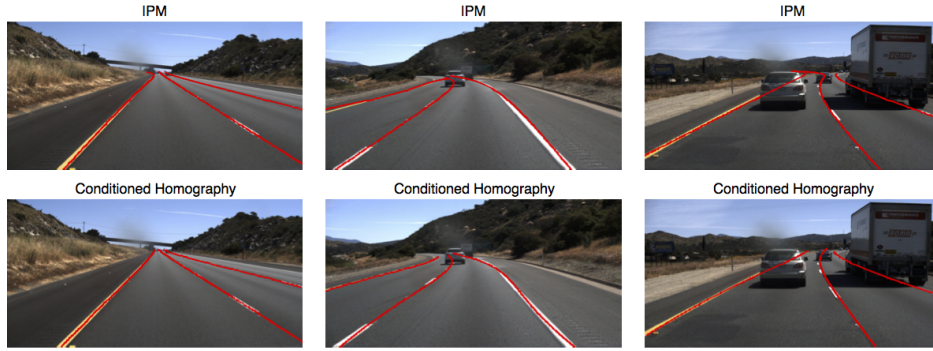


Figure 3.17: Comparison between the lanes fitted using a classic homography transformation from the camera calibration and the lanes fitted using the conditioned homography from *H-Net*

using an iterative algorithm like gradient descent. The main issues that can be encountered during this procedure are related to the high dimensionality of the problem. The optimization problem is non-convex and the learning process can potentially end in suboptimal local minima.

Recently Adam represents the start of the art of the optimization algorithms and is the recommended choice for training deep neural networks. The algorithm has been used to train the networks developed in this work and will be briefly described in the next paragraph.

3.4.1 Adam optimizer

Adam [61] is a recently proposed extension to stochastic gradient descent based on adaptive estimates of lower-order moments. Basic stochastic gradient descent methods keep a single learning rate for all weight updates and the learning rate does not change during training. Adam combines the advantages of two other algorithms for first-order gradient optimization: Adaptive Gradient (AdaGrad) and Root Mean Square Propagation (RMSProp).

AdaGrad defines a different learning rate for each parameter of the model. The main issues of Adagrad are that the learning rate monotonic decrease is too aggressive and often cause to stop learning too early. RMSProp maintains as well a per weight learning rates but weights are updated using a moving average of the gradients. In this way the learning rates do not get monotonically smaller with beneficial effect on the training.

Adam update rule is shown in equation 3.24. The optimization procedure computes the moving average for the gradient m and the gradient squared v with parameters β_1 and β_2 to control the decay. In the paper is suggested to use $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

$$\begin{aligned}
 m &= \beta_1 m + (1 - \beta_1) \nabla x \\
 v &= \beta_2 v + (1 - \beta_2) \nabla x^2 \\
 x &= x - \alpha \frac{m}{\sqrt{v} + \epsilon}
 \end{aligned} \tag{3.24}$$

In addition weight decay has been used during training as a form of L2 regularization. In the case of networks trained from scratch the Xavier weight initialization scheme has been used.

3.4.2 Data augmentation

Data augmentation is a really popular way used to reduce overfitting in deep learning models. The training process typically requires a huge amount of data to allow the model to reach an adequate capability to generalize and avoid overfitting. This kind of technique extensively proved to be helpful in increasing the network representational power. In practice data augmentation artificially increase the size of the training set through random slight adjustments to the training images performed independently for every epoch.

- Flip: it consists of randomly flipping the training image vertically or horizontally. In this work only horizontal flip has been used since vertically flipping road scenes, which present a strong vertical semantic meaning, can be deleterious for the model final performance.

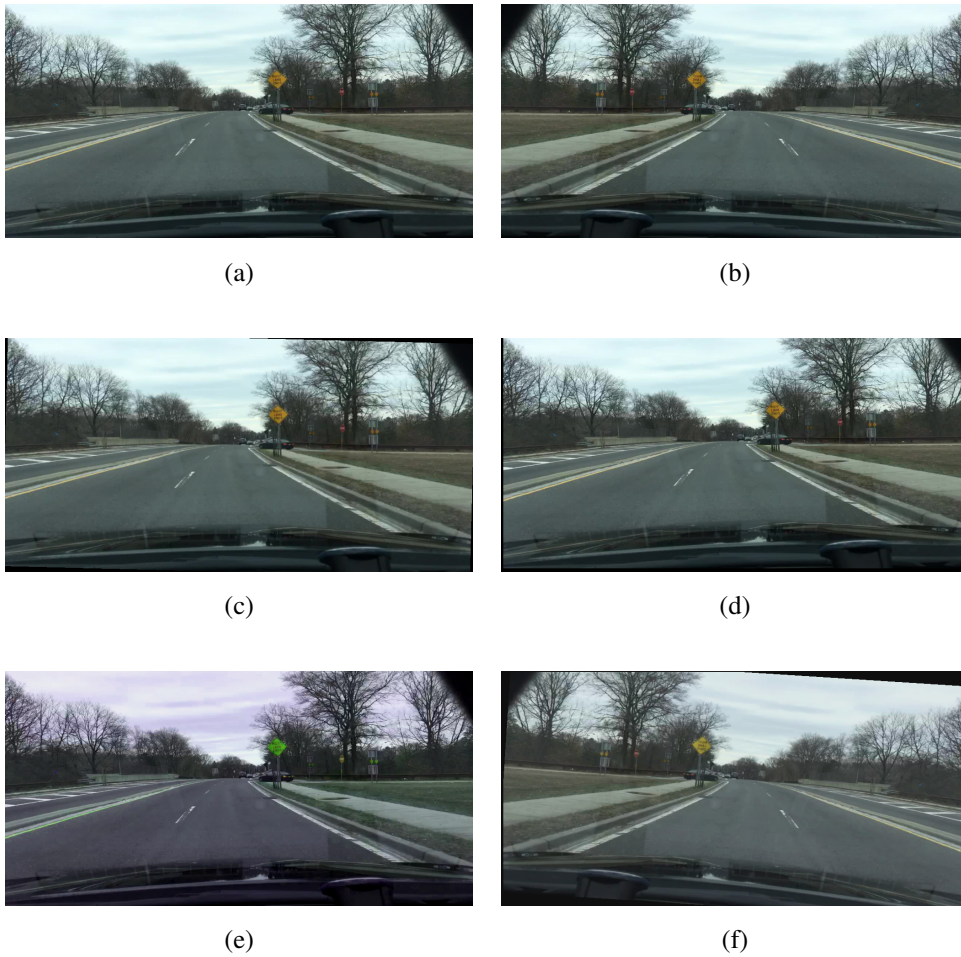


Figure 3.18: Examples of data augmentation. In 3.18a is shown the original image, in 3.18b is shown the horizontally flipped image, in 3.18c is shown the rotated image, in 3.18d is shown the translated image, in 3.18e is shown the image with color jitter transformation, in 3.18f is shown the image transformed using a composed data augmentation.

- Rotate: it consists in rotating the image around the center by a randomly computed degree angle. The range chosen is 3 degree. The non-visible pixels

generated by the transformation are set to zero.

- Color jitter: it consists of randomly varying the image brightness, contrast and saturation. This transformation is performed in HSV color space.
- Crop: given that the original image resolution is fairly higher than the resolution of the images fed to the network it is possible to create a virtually new training image by cropping the farthest part of the road scene.
- Rescale: it has been realized in two different ways. In the first case the image is randomly cropped by a few pixels (in the range of 0-10) along one of the borders and then is resized to the network input size. The second consist of randomly changing the aspect ratio of the image (the width-height ratio values varied in the range of 0.85 and 1.15).
- Translation: it consists in shifting the image horizontally or vertically by a random number of pixels. The non-visible pixels generated by the transformation are set to zero.

These transformations can be composed together and applied to the training images to further increase the variability of the dataset. In figure 3.18 are shown the different types of data augmentation transformations applied to an image.

3.4.3 Finetuning a pretrained model

Training a convolutional neural network from scratch may not always be feasible or advisable. It relies a lot on weight initialization which is performed using random technique and requires a huge amount of training data.

A common technique is to train the neural network on a huge generic dataset (like ImageNet) and then use this pretrained network as a fixed feature extractor. This means freezing the weights of the first layers of the network which usually represent low level features like edges or patches and update only the deeper layers which have a higher level of visual representation. The features of the lower layers are indeed generic and they can be useful and shared between different tasks. On the other hand

features from deeper layers are more task specific.

Another strategy is to finetune a pretrained network which means continue the process of backpropagation and optimization of the weights of the network. This strategy is commonly used in the case where the task-specific dataset is small while a bigger dataset which presents similar characteristic is available. The bigger dataset is used to pretrain the network and the smaller one is successively used to specialize the network. For example, in the case of lane detection the training has been performed on a network pretrained on the Cityscapes dataset. While the final objective of the networks are different the data used for training share the same context (e.g. road and street scenes).

3.5 Model optimization

The inference step of a convolutional neural network is usually a computationally intensive and memory expensive task. Optimizing these two aspects is of crucial importance to enable the application of CNN based algorithms to run on real-time in low power devices. This is a characteristic of primal importance for tasks like autonomous driving. In the next sections two different methods for improving inference time will be explained: weight pruning and layer cascade network. Weight pruning reduces the number of parameters through kernel sparsification. On the other hand layer cascade network sparsify the feature maps by dividing the architecture into multiple stages and each stage process only the pixels that have not been classified with enough confidence in previous stages.

3.5.1 Weight pruning

The aim of weight pruning is to reduce the number of parameters by removing unnecessary weights without affecting the final performance of the model. Once applied, the method should decrease both the required computation time and storage size. The procedure used in this thesis follows the work of Han [62] which consist in a three step iterative process. A diagram of the procedure is shown in figure 3.19.

Step one is the standard training process where all weight in the network are learned.

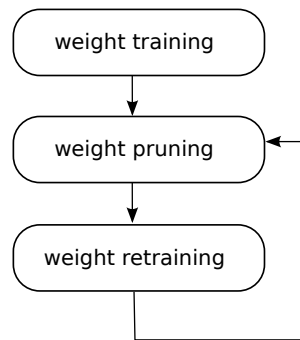


Figure 3.19: Diagram of the three steps weight pruning procedure

Step two and three are iteratively repeated until the required level sparsification is reached or until the accuracy of the network prediction drops below a defined threshold. The purpose of step two is to understand and select which weights are unnecessary to the model and won't cause performance degradation. Two different ways of weight removal have been implemented:

- Threshold based: all weights below a certain threshold are removed from the network
- Percentile based: a given percentage of smallest weights are removed from the network

Step three consists in retraining the network with pruned weights. This process has been implemented as a masked convolution where pruned elements of the filter are simply not forward or backward propagated.

Reiterating this procedure (step 2 and 3) allows the network to progressively adapt the surviving connections therefore it helps to improve the final accuracy after the sparsification process.

3.5.2 Layer cascade network

The idea of a layer cascade architecture, initially proposed by Li in [63], is that in a generic segmentation network the majority of pixels can be classified with high

confidence within a few layers. These are usually pixels within spatially large objects or pixels belonging to a class which has characteristic really different from the others in the scene. On the other hand, pixels placed near the objects boundaries or pixels of thin objects are typically harder to classify correctly. For example in a dataset of urban street images like Cityscapes pixels belonging to classes like sky, road, and car are likely to be classified with high confidence, while pixels belonging to pedestrians, poles or bicycle usually presents a lower accuracy value.

A layer cascade network is a modular architecture, composed by a series of stages, where shallow stages are trained to classify easy pixel regions, while deeper stages focus on the classification of the remaining hard pixel regions. This can be accomplished by forwarding to deeper layers in the network only the pixels that are not classified with enough confidence in earlier layers. The diagram of the layer cascade architecture is shown in figure 3.20.

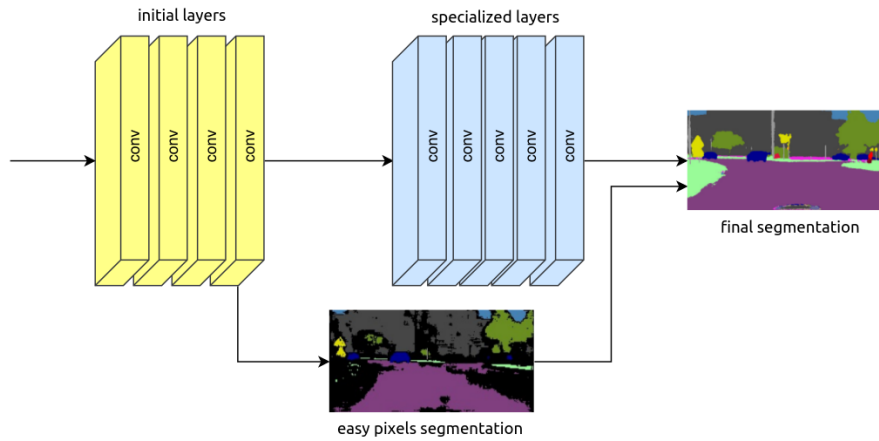


Figure 3.20: Layer cascade architecture

This architectural design can potentially provide advantages in terms of segmentation accuracy and inference time. Indeed allowing the deeper layers to focus just on the hard pixels should improve the overall learning capabilities of the network. At the

same time if most of the pixels are classified in earlier stages the number of operations left in deeper layers is drastically reduced. The layer cascade network can be trained end-to-end by defining a specific loss function for every stage of the network and then optimize them jointly. In later stages the gradient will be back-propagated only from the harder pixels so the weights in these stages will learn only characteristic of those pixels. On the other hand weights on earlier stages will also account for pixels back-propagated from deeper stages allowing them to learn the global image context. The layer cascade principle has been applied to the popular ENet and ERFNet architectures, by separating these networks in two stages. In particular the stage split has been placed in an intermediate layer of the encoder for both networks.

The confidence threshold used to decide whether a pixel can be considered hard or easy define the sparsity for the feature maps of the following stages. A series of test using different thresholds has been carried out to evaluate the variation of the segmentation accuracy related to the obtained sparsity (table 3.1).

In figure 3.21 are shown the segmentation results of the easy pixels after the first stage using two different thresholds.

Threshold	Enet		ERFNet	
	Sparsity %	Error %	Sparsity %	Error %
0.995	44.4	3.72	46.26	2.21
0.99	51.99	4.76	53.21	2.98
0.985	56.3	5.39	57.24	3.43
0.98	59.41	5.82	60.0	3.88
0.97	63.69	6.47	63.95	4.3
0.96	66.62	6.95	66.72	4.81
0.95	69.01	7.3	68.8	5.24

Table 3.1: Level of sparsity and segmentation error obtained on ENet and ERFNet using different thresholds values in the first stage.

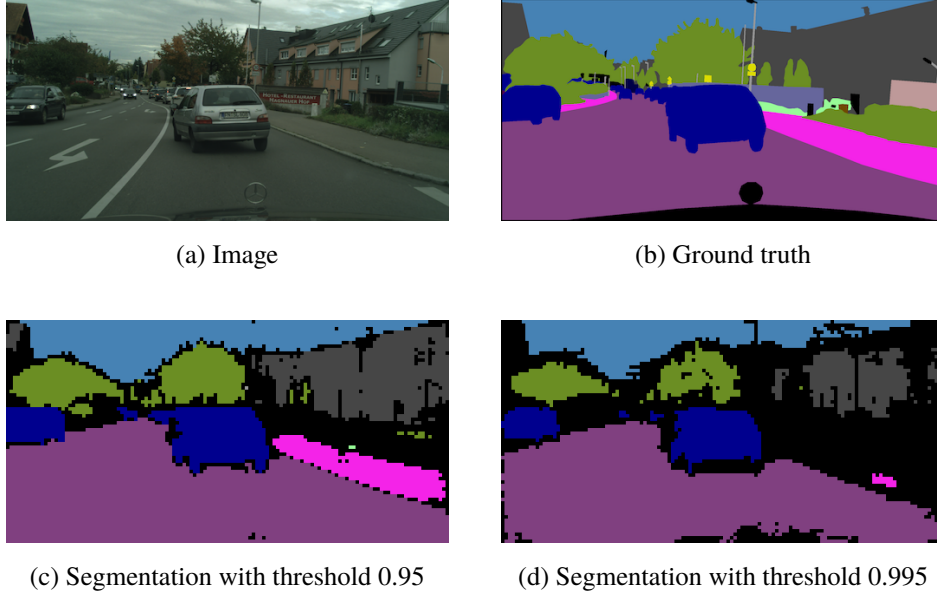


Figure 3.21: Segmentation of the easy pixels after the first stage using two different thresholds. In figure 3.21a is shown the original image, in 3.21b the segmentation ground truth, in 3.21c the segmentation output after stage 1 with a threshold of 0.95 and in 3.21d the segmentation output after stage 1 with a threshold of 0.995

The selected architectures present particular layers like dilated convolution or dropout that could interfere with the propagation of sparse feature maps through the network.

In particular the large receptive field of dilated convolution could have a negative effect on border regions since the filter could potentially focus on pixel removed by previous stages losing contextual information. To verify the impact of dilation a test has been performed to evaluate the segmentation error for a modified cascade layer ERFNet with and without the presence of dilated convolution. In the second case dilated convolution have been replaced by regular convolution. The result are shown in table 3.2 and demonstrates that in this case dilated convolution are still beneficial in terms of improving the model accuracy.

Dilated	Threshold	IoU %
Yes	0.985	66.17
No	0.985	60.79

Table 3.2: Intersection over union results on ERFNet with and without using dilated convolution.

The other concern was related to the use of dropout in the deeper stages since the original ENet and ERFNet architecture presented this layer. Dropout randomly disables neurons in a layer during training. Since the layer cascade operates a deterministic form of neuron removal the potential issue is the excessive loss of contextual information. As for dilated convolution a test has been performed to compare the segmentation accuracy with and without using dropout on the ENet architecture. Similarly to the previous case dropout proved to be helpful for final model segmentation accuracy as shown by the results in table 3.3.

Dropout	Threshold	IoU %
Yes	0.985	59.63
No	0.985	57.53

Table 3.3: Intersection over union results on ENet with and without using dropout.

Chapter 4

Results

This chapter presents the results obtained on the datasets illustrated in the previous chapter using the proposed architecture, loss functions and training procedures. The deep learning framework used to train and test the presented networks was PyTorch, an open source machine learning library written in Python.

The chapter is structured in the following way. First will be presented the result of the dense semantic segmentation on the Cityscapes dataset. In the second section will be described the results obtained for the lane detection segmentation on the TuSimple and BDD100K datasets. The final section will show a qualitative comparison between a traditional and a CNN-based lane markings detector.

4.1 Results for semantic segmentation on Cityscapes

This section describes the results obtained on the Cityscapes dataset on the task of semantic segmentation using the different architectures presented in the previous chapter: the residual bottleneck network (RBTNet), the inverted residual network (IRNet) and the residual dilated pyramid network (RDPNet).

The networks have been trained for 300 epoch with a batch size of 8 using Adam optimizer to minimize a multi class cross entropy function. For Adam the following hyper-parameters have been used: a learning rate value of 0.0005, a beta 1 value of

0.9, a beta 2 value of 0.999 and a weight decay value of 0.0001.

The aim of these tests is to understand the representative capabilities of the proposed networks on a popular benchmarked dataset. In this way, a comparison with other network architectures present in the literature has been carried out. In particular ENet, ERFNet and ESPNet have been chosen since they are efficient in terms of computation and memory given their relatively low number of parameters. Along with them, another set of networks have been selected for the comparison, namely SegNet, DeepLab, RefineNet, PSPNet. These networks, when released, represented the state of the art in terms of accuracy for the Cityscapes dataset.

The metric used to evaluate the results of the segmentation is the Intersection over Union (IoU) score computed per class and per categories.

Network	IoU (class) %	IoU (categories) %	# params (M)
RBTNet	59.5	81.6	0.422
IRNet	62.1	83.2	1.18
RDPNet	63.9	84.1	1.8
ENet	58.3	80.3	0.364
ERFNet	69.7	87.3	2.06
ESPNet	60.3	82.2	0.364
SegNet	57.0	79.1	29.5
DeepLabv3+	82.1	92.0	44.04
RefineNet	73.6	87.9	42.6
PSPNet	78.4	90.6	65.7

Table 4.1: Results on the Cityscapes benchmark. IoU for class and categories is reported along with the number of parameters of the networks. The first set of networks are the ones proposed in this work. The second set is composed by lightweight and efficient networks. The last set instead presents networks designed to be as accurate as possible.

The results are summarized in table 4.1. In addition to the IoU scores for each architecture is also reported the number of parameters of the model (in millions). The proposed architectures presents performance comparable with the state of the art networks that focus on efficiency.

In figure 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 are shown qualitative segmentation results. The images in the figures represent respectively (from top left to bottom right): the original RGB image, the ground truth labeling, ERFNet segmentation result, RBTNet segmentation result, IRNet segmentation result and RDP segmentation result. For ERFNet a pretrained version available on the ERFNet Github project webpage has been used.

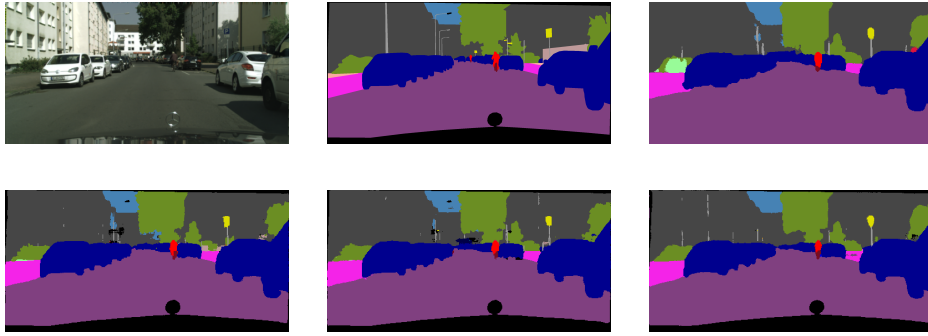


Figure 4.1: Qualitative segmentation results on Cityscapes.

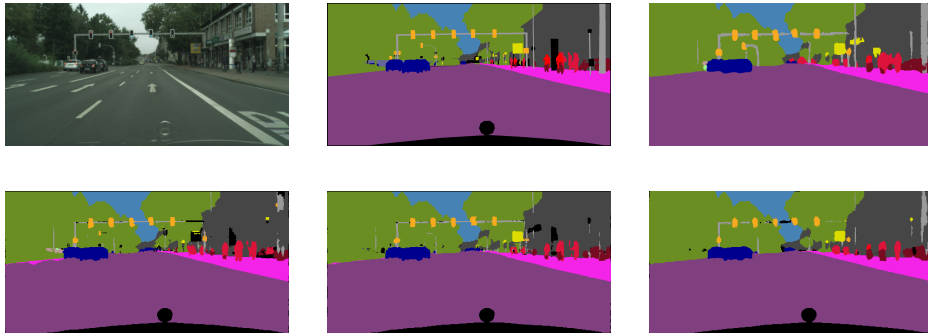


Figure 4.2: Qualitative segmentation results on Cityscapes.

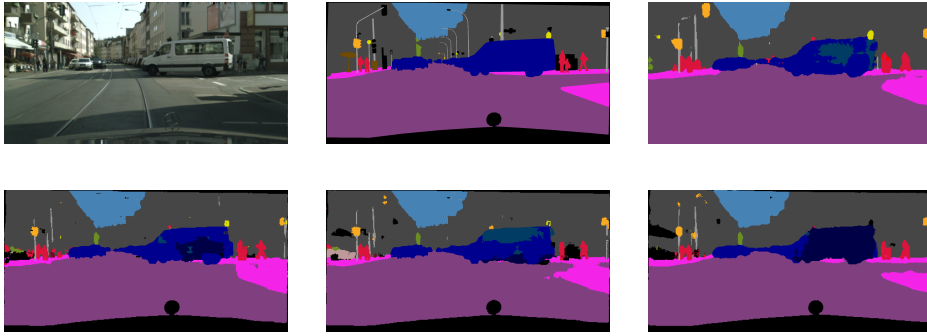


Figure 4.3: Qualitative segmentation results on Cityscapes.

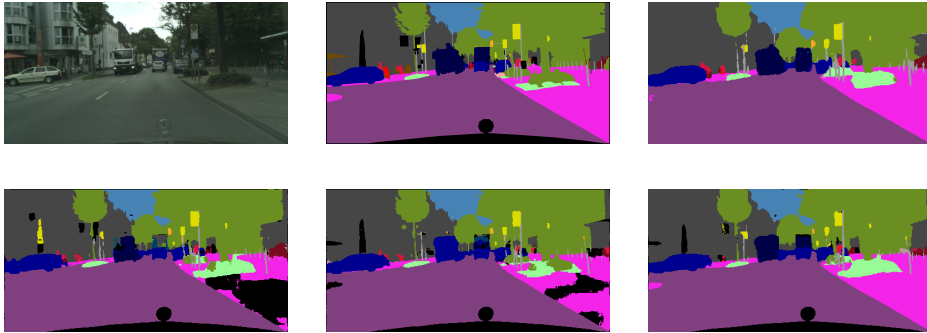


Figure 4.4: Qualitative segmentation results on Cityscapes.

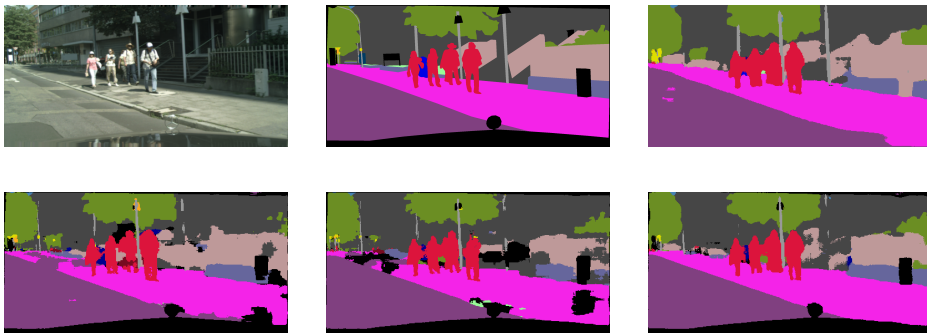


Figure 4.5: Qualitative segmentation results on Cityscapes.

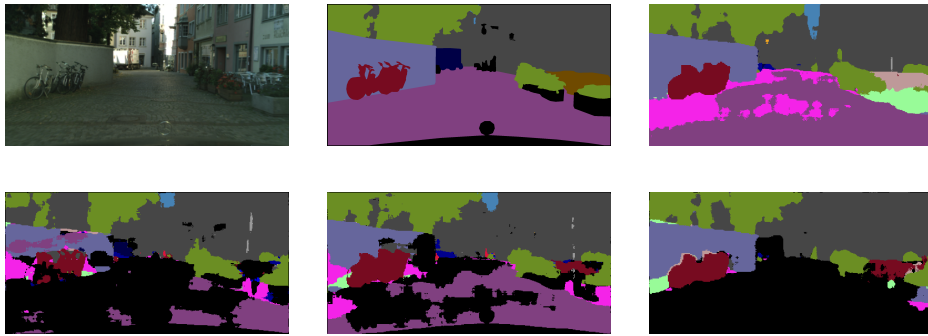


Figure 4.6: Qualitative segmentation results on Cityscapes.

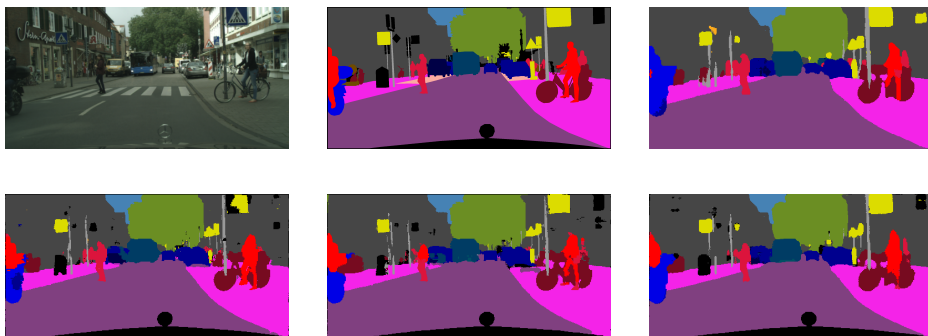


Figure 4.7: Qualitative segmentation results on Cityscapes.

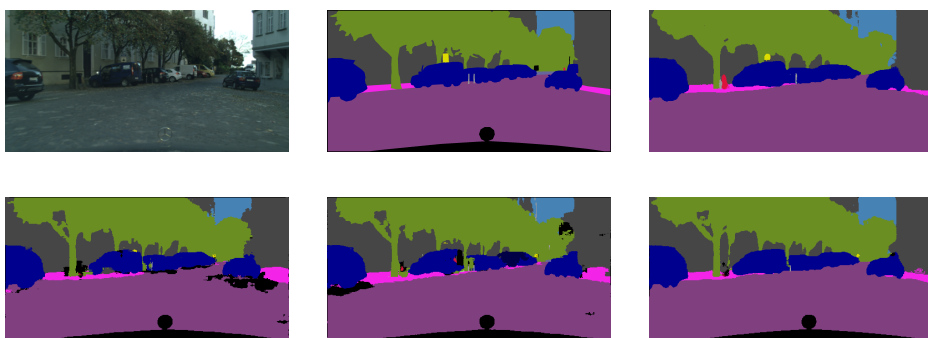


Figure 4.8: Qualitative segmentation results on Cityscapes.

4.1.1 Weight pruning results

Weight pruning is a technique that allows reducing the number of parameters in a network with a limited impact on the final accuracy. In this section are presented the results obtained by applying weight pruning to different networks trained on the Cityscapes dataset. In particular table 4.2 reports the IoU score at different level of sparsity for ENet and ERFNet architectures.

Sparsity	IoU %	
	ERFNet	ENet
0%	69.7	58.3
50%	69.7	58.1
55%	69.6	58.2
60%	69.6	58.2
65%	69.3	58.1
70%	69.5	57.9
75%	69.4	58.1
80%	69.2	57.7
85%	69.0	57.5

Table 4.2: IoU class score of ENet and ERFNet on the Cityscapes benchmark applying weight pruning with different sparsity level.

The initial percentage of pruning in these tests was 50% and has been increased by 5% for each retraining phase until reaching a 85% percentage level of sparsity. Each retraining phase was composed of 50 epoch. The pruning has been applied in an iterative manner which means that starting from the initial percentage value the following pruning phase has been applied to previously pruned network and not the original one. The results highlight that even with a sparsity level of 85% the drop in accuracy is almost negligible (around 2%).

4.1.2 Cascade network results

In this section are presented the results for the layer cascade network optimization. The model evaluated in the tests is the layer cascade version of ENet (ENet-LC), and in particular comparison with the regular version the network has been carried out. In table 4.3 are shown the IoU performance of ENet compared with ENet-LC using different thresholds. As the acceptance threshold decreases the accuracy decrease but this is associated with an increased sparsity in the network hence faster execution times. The network that delivers the best performance is ENet-LC with a threshold value of 0.985.

Network	Threshold	IoU %
ENet	-	59.19
ENet-LC	0.99	59.46
ENet-LC	0.985	59.76
ENet-LC	0.98	59.16
ENet-LC	0.95	58.56

Table 4.3: Comparison of IoU class score between ENet and ENet-LC on the Cityscapes benchmark.

4.2 Lane detection results

This section describes the results obtained on the TuSimple Lane Detection and BDD100K datasets on the task of lane detection using segmentation. The architectures trained and evaluated in these tests are the ones presented in the previous chapter: the residual bottleneck network (RBTNet), the inverted residual network (IRNet) and the residual dilated pyramid network (RDPNet).

The following tests will be presented: instance segmentation results, qualitative critical cases analysis, binary segmentation results and a comparison between CNN based lane detection and lane detection using classic Computer Vision methods.

4.2.1 Instance segmentation result

The metrics used for the evaluation of the networks trained for instance segmentation are a modified version of Intersection over Union for instance segmentation (instanceIoU) and the TuSimple lane detection score, which was described in detail in section 3.1.2, and is composed by three indexes: accuracy, false positive and false negative. Given a set of ground truth instances and predicted instances the instance Intersection over Union is computed in the following way:

- Step 1: For each ground truth instance is computed the IoU value with respect to each predicted instance.
- Step 2: For each ground truth instance the maximum value of IoU with respect to a predicted instance is considered.
- Step 3: The average value of these maximum IoU scores is computed.
- Step 4: Step 1 to 3 are repeated with the two set swapped and another average IoU value is obtained.
- Step 5: The minimum of the two average values is the final score.

All the networks were pretrained for 300 epoch using binary semantic segmentation on the TuSimple and BDD100K datasets. The loss used to pretrain the network was the dice loss function. The following finetuning procedure, which lasted 300 epoch, optimized the instance segmentation loss function described in section 3.3.3 using the Adam algorithm with a learning rate value of 0.0005, a beta 1 value of 0.9, a beta 2 value of 0.999 and a weight decay value of 0.0001. During training the following data augmentation types were enabled: horizontal flip, rotation, translation, crop, rescale and color jitter. The image resolution used was 512x256. Another important hyperparameters is the line thickness of the lanes on the label images. Using a resolution of 512x256 a line width of four pixels has been chosen, an example of a label image is shown in figure 4.9.

Table 4.4 reports the instance IoU score for different architectures and for three different training and test set. In the first case, the networks were trained using a

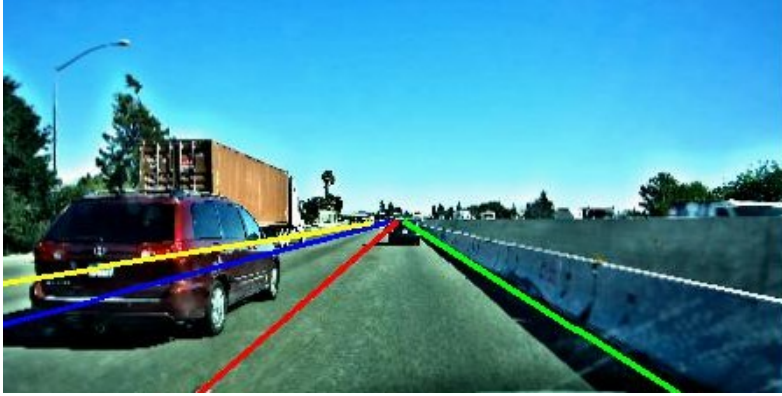


Figure 4.9: Example of lane label image. The thickness used for the lanes line on the image is four pixels wide.

training set composed only by the TuSimple lane detection data and subsequently tested on the TuSimple test set. In the second case, the networks were trained and tested using a dataset comprising both TuSimple and BDD100K. Finally, in the last case, the networks were trained on the TuSimple dataset and then tested on a dataset composed by TuSimple and BDD100K.

Network	TS IoU (%)	TS+BDD IoU (%)	TS / BDD(test only) IoU (%)
RBTNet	69.56	63.21	53.26
IRNet	71.54	65.02	54.81
RDPNet	73.63	67.96	55.36
ENet	63.36	57.89	48.21
ERFNet	74.58	68.69	57.52
ESPNet	66.12	60.47	50.36

Table 4.4: IoU score using TuSimple (TS) dataset as training set and test set, TuSimple (TS) and BDD100K datasets as training set and test set, TuSimple (TS) as training set and TuSimple (TS) and BDD100K datasets as test set

Table 4.5 reports the training set, validation set, test set sizes for the TuSimple and BDD100K datasets.

Name	Train	Val	Test
TuSimple	2500	500	626
BDD100K	6500	1483	2000

Table 4.5: Training set, validation set, test set sizes for the TuSimple and BDD100K datasets

The results using only the TuSimple dataset both for training and testing report the highest IoU score. This can be easily explained by the fact that the TuSimple dataset is a lot less variable dataset than the BDD100K since it presents only highways scenarios at daytime. For this reason the final trained models have less representative and generalization capabilities. Indeed the models trained only on TuSimple when tested on BDD100K exhibit inferior performances since are unable to handle urban, bad weather or nighttime scenarios.

As expected the results for the models trained on TuSimple and BDD100K obtained a higher accuracy with respect to the models trained only on TuSimple. Regardless, as will be shown later in the section in the qualitative results, the size of the dataset is still too small to allows the network to learn all the complexities of urban and unmarked roads.

To conclude the network that gives the best performances is ERFNet that, among the architectures included in the comparison, is the one that presents the highest number of parameters thus has potentially more representative capabilities.

In table 4.6 are reported the execution times of the forward pass for the tested networks. The inference time tests have been performed on an NVIDIA Tesla P40 GPU using Pytorch 0.4 with CUDA 8.0 and cuDNN as CUDA backend. The image resolution used to measure the forward pass times was 512x256.

Network	FW Time (ms)
RBTNet	7.3
IRNet	9.8
RDPNet	9.6
ENet	6.8
ERFNet	8.1
ESPNet	6.9

Table 4.6: Execution times (in milliseconds) for the forward pass.

Another test has been carried out to evaluate the performance using the TuSimple score. In this way, it was also possible to compare the trained network with other methods whose results for the challenge were published online. The results are reported in table 4.7.

Method	Accuracy	FP	FN
RBTNet	94.8	6.45	8.45
IRNet	95.1	5.12	5.93
RDPNet	95.6	4.15	6.13
Pan [9]	96.53	6.17	1.8
Hsu [60]	96.50	8.51	2.69
Neven [10]	96.40	23.65	2.76

Table 4.7: TuSimple lane detection results. Accuracy, false positive (FP) and false negative (FN) are reported.

4.2.2 Qualitative results and critical cases

In this section are presented some qualitative results. In figure 4.10 are shown qualitative segmentation results for the lane detection based on instance segmentation on

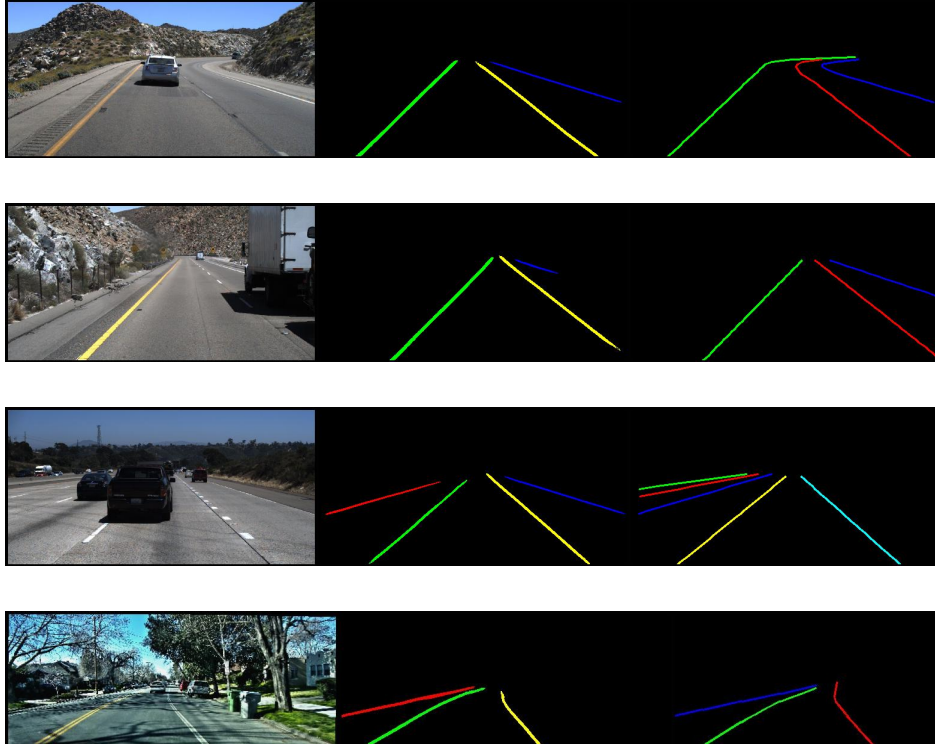


Figure 4.10: Qualitative instance segmentation results for lane detection. On the left column is shown the RGB image, on the center column is shown the predicted segmentation and on the right column is shown the ground truth segmentation.

the validation set. The color of the lanes in the predicted and ground truth does not match since the instance segmentation does not predict the lane instance based on the position. The only aim of instance segmentation is to assign to pixels belonging to the same lane the same instance id. The type of lane based on position can be easily recovered using geometric assumptions.

In figure 4.10 is shown a qualitative segmentation results on a test image displaying a urban street scene without lane marking. In figure 4.12 are shown some challenging scenarios where the network fails to detect the lanes correctly. In particular in figure 4.12a the unmarked lane is not correctly identified even in the presence of the line of



Figure 4.11: Qualitative instance segmentation results for lane detection. On the left column is shown the RGB image and on the right column is shown predicted segmentation over the image.



Figure 4.12: Qualitative instance segmentation results for lane detection in case of challenging scenarios where the network fails.

parked cars at both sides of the road. In figure 4.12b the barrier delimiter is wrongly detected as lane. In figure 4.12c the unmarked lane is not identified properly because of the snow condition, a scenario scarcely presents in the dataset. Finally in figure

4.12d are again not detected because of the dirt in front of the camera and the scarce road illumination. To solve these problems a lot more data is probably needed to help the network learns to detect the lanes in these corner cases.

In figure 4.13 are shown a couple of interesting cases of subsequent frames where in one frame the model correctly identifies the lanes while in the second frame, although being really similar to the other, it fails to detect the lanes.



Figure 4.13: Qualitative instance segmentation results for lane detection. Two cases where the network in subsequent frames obtains really different results.

To solve this issue the most reasonable choice would be to add a temporal relationship in the models. This could be realized for example by inserting a recurrent model like GRU or LSTM modules in the network. The biggest problem related to the realization of this improvement is that there is no annotated data over consecutive frames available. Alternatively, a standard tracking algorithm could be added as a post-processing step of the output of the network.

4.2.3 Binary segmentation results

This section reports the results obtained by the proposed networks on the task of semantic segmentation using two class: background and lanes. These models were used as the initial state for the procedure of instance segmentation training.

Table 4.8 reports IoU results for binary segmentation using two different loss: weighted cross entropy and dice loss. The datasets used to train the models were the TuSimple and BDD100K datasets with an image resolution of 512x256. As for the case of instance segmentation the training lasted 300 epoch using the Adam algorithm with a learning rate value of 0.0005, a beta 1 value of 0.9, a beta 2 value of 0.999 and a weight decay value of 0.0001. Data augmentation were enabled as well.

Network	CE IoU (%)	Dice IoU (%)
RBTNet	66.32	64.81
IRNet	69.98	67.82
RDPNet	72.11	70.43
ENet	64.65	63.01
ERFNet	74.77	72.81
ESPNet	66.28	64.74

Table 4.8: IoU score for binary segmentation for models trained on the TuSimple dataset and BDD100K dataset using cross entropy loss and dice loss

4.2.4 CNN based vs classic lane detection results

This section provides a comparison between the performance of a lane markings detector based on computer vision classic algorithms and the lane detection developed using CNN.

The classic lane detection approach works on the bird's eye view space and extracts features starting from a DLD filter. The image obtained after the application highlight regions that present a dark-light-dark pattern like lane markings. To overcome issues

caused by illumination or shadows effects an adaptive threshold is used to binarize the DLD result. Finally, a connected component algorithm is applied to the binary mask to cluster the markings.

Regarding the CNN based lane detection a network trained for instance segmentation is used for comparison, in particular the predictions generated by the ERFNet model are reported.

Both results require further processing if a parametric lane model is needed. In the case of the CNN prediction, the lanes are already clustered in the different boundaries although some additional checks may be needed to handle instance segmentation failure to group pixels belonging to the same lane. An example is shown in the second row of figure 4.16. On the other hand in the case of the classic lane detection, the final segments require additional processing in order to be grouped in one of the lanes. The results shown in this section will be purely qualitative and they will focus on the analysis of challenging scenes where the lack of context could easily mislead a classic algorithm.

In figure 4.14 are shown urban scenes presenting clear road markings. Regarding the classic lane detection in the image in the first row are present some outliers caused by the railing.

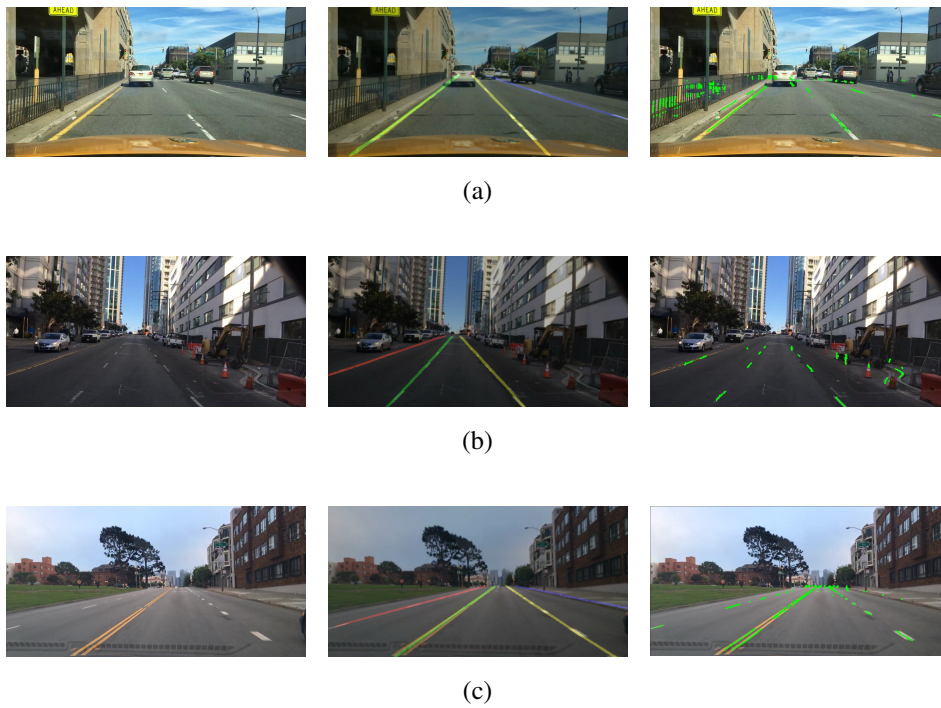


Figure 4.14: Qualitative comparison between CNN based lane detection and classic Computer Vision based lane detection. In the figure urban scenes are displayed presenting clear road markings. On the left column are shown the original images, on the center column are shown the CNN predictions, on the right column are shown the classic lane detection features.

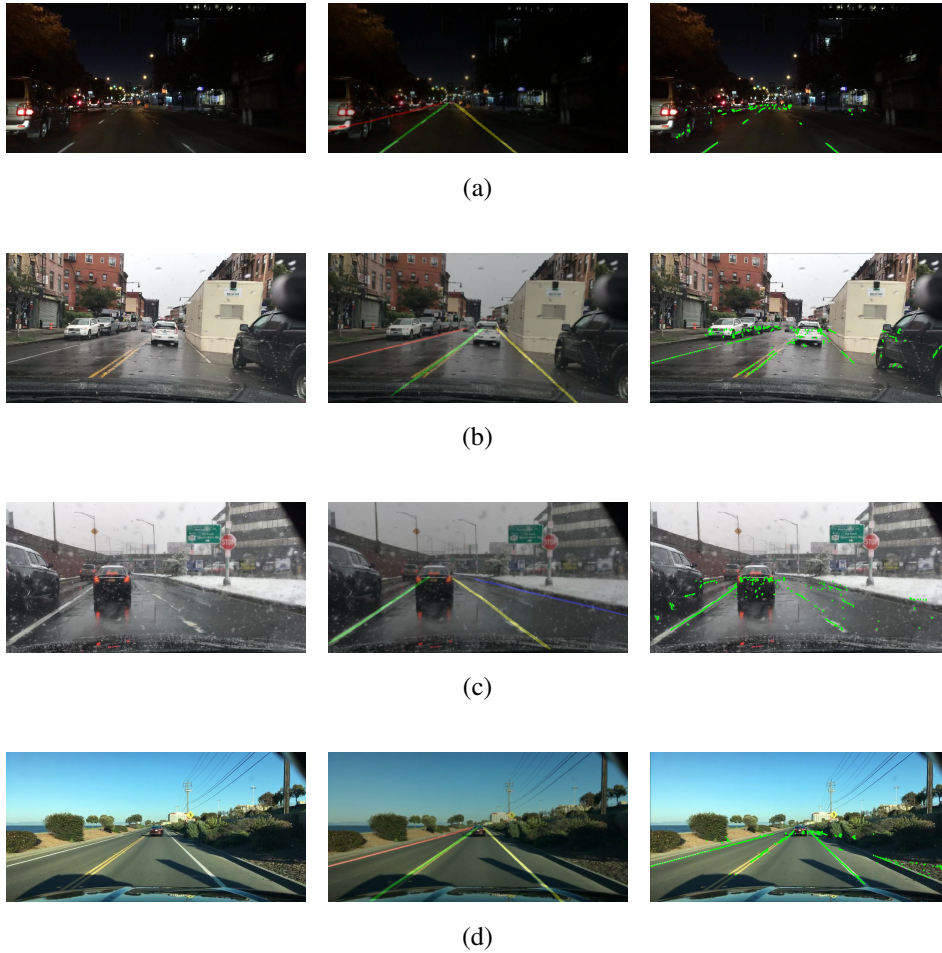


Figure 4.15: Qualitative comparison between CNN based lane detection and classic Computer Vision based lane detection. In the figure are displayed scenes presenting different weather and illumination conditions. On the left column are shown the original images, on the center column are shown the CNN predictions, on the right column are shown the classic lane detection features.

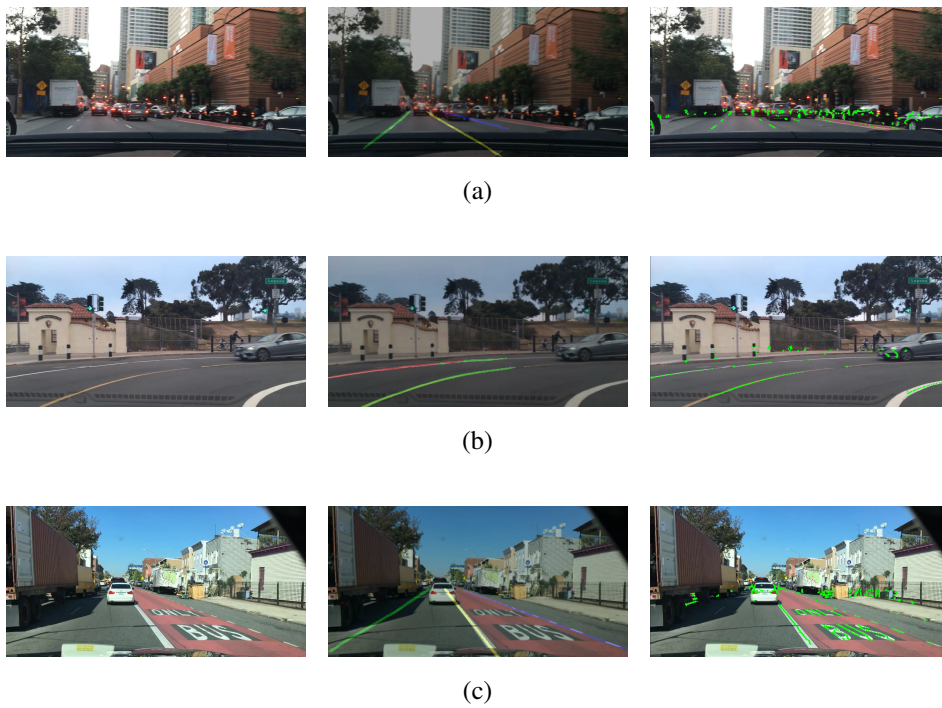


Figure 4.16: Qualitative comparison between CNN based lane detection and classic Computer Vision based lane detection. In the figure are displayed urban scenes characterized by challenging features (traffic, narrow turn and horizontal markings). On the left column are shown the original images, on the center column are shown the CNN predictions, on the right column are shown the classic lane detection features.

Chapter 5

Discussion

This thesis studied the lane detection problem using approaches based on Convolution Neural Network. This choice was motivated by the fact that in the last few years CNN demonstrates to be an extremely powerful framework to understand the context of a scene which is a key requirement in identifying the lanes in complex road environments.

The main contribution of this work is the development of different network architectures to solve the Lane Detection task as an image segmentation problem. In particular, in this thesis, three original model architectures have been proposed whose aim is to maximize the trade-off between accuracy and efficiency in order to ensure the execution of the algorithm in real-time.

Although Lane Detection represents one of the most active research topics in the field of autonomous driving, there are still very few methods that tackle the problem using an approach based on Deep Learning; moreover, they are only applied to the specific case of highway environment. For this reason, the thesis focused on the analysis and resolution of the lane detection problem both in easy scenarios (like highways or clearly marked roads) and in challenging environments. These are represented by urban and rural environment, which frequently presents unmarked or scarcely marked roads, or more generally by scenes that present critical lighting, traffic, weather, and environmental conditions.

The lane detection output was defined by road lanes boundaries. In particular, the following boundaries were considered: ego lane left, ego lane right, left lane and right lane. To obtain the desired output an approach based on instance segmentation has been chosen, whose prediction provides, along with the image pixel labeling, a unique label to different lane boundary instances.

The architectures presented in the thesis were trained using two different datasets for lane detection. The TuSimple Lane Detection Benchmark which contains almost 3000 images, acquired on US highways at daytime, and BDD100K which contains 100000 road video clips collected in a wide variety of conditions and scenario. For the latter, only 10000 images presented the lane boundaries annotations.

Lane detection, like other automotive related perception tasks, requires that the processing is executed in real-time. For this reason, when designing the proposed networks, the main focus was to maximize the efficiency using specific network blocks and layers.

The network architectures developed followed the popular encoder-decoder scheme for semantic segmentation and presented layers like dilated convolutions, which provide a larger receptive field without increasing the computational cost, factorized convolutions, and depthwise convolutions. Then to further reduce the required computation also residual bottleneck and pyramid of features blocks have been experimented.

Regarding the training procedure different loss functions, data augmentation techniques and finetuning strategies have been studied to maximize the model's final performances. In particular, the employed training loss allowed carrying out instance segmentation using a single branch network architecture that output directly the different lane instances without the need of an intermediate embedding representation. Additionally, a couple of alternative training procedures have been analyzed to further reduce the number of parameters of the networks and the required number of operations during the forward step.

The first optimization was achieved using a weight pruning technique that removes redundant networks parameters through an iterative training procedure which sparsify the filters of the model.

The second optimization was instead accomplished by transforming a standard net-

work into a layer cascade network that divides the architecture into multiple stages where each stage classify only the pixels that have not been predicted with enough confidence in the previous stages.

To evaluate the developed architectures a series of tests and comparison with other popular networks present in literature have been carried out.

Firstly the proposed models have been trained using Cityscapes dataset, a popular benchmark for the task of road scene semantic segmentation. The principal aim of these tests was to compare the proposed architectures with others in the state of the art on a widely used benchmark. The networks performances were analyzed considering a trade-off between efficiency (in terms of model size and required operations) and segmentation accuracy (evaluated using the Intersection over Union score). The proposed networks reported comparable results with other approaches in the state of the art that focuses on efficiency. After that, the presented architectures were trained for the lane detection task using the instance segmentation loss.

The networks have been evaluated using two different metrics: an IoU score modified for instance segmentation and the accuracy score proposed for the TuSimple lane detection challenge.

The tests carried out shows that the implemented architectures and training procedures are able to obtain results comparable to other approaches present in the state of the art on the TuSimple Lane Detection Challenge.

The presented network models offered very promising results in the case of highly complex and challenging scenarios; this has been shown through a qualitative comparison with a classic computer vision based lane detection. At the same time, some results highlighted that the trained networks were not always able to handle some of the most complex scenarios most probably because there were not enough examples to allows the networks to generalize to every possible scenario.

5.1 Future works

The networks presented in this thesis work in a frame-based manner which means that no temporal relationship is considered between consecutive images. A possible way to further improve the networks performances would be to insert recurrent modules in the architectures in order to obtain more stable and temporally consistent results.

Recently training using Adversarial network gained much attention between the Deep Learning community thanks to its capabilities to exploit high-level structure regularity in the data and detect inconsistencies in the predictions. Lane detection output presents regular and structured patterns, for this reason using adversarial training could be used to obtain a more realistic output that preserve the typical lane structure. In this case, the generator network could be the segmentation network while the discriminator could be a second network trained to distinguish ground-truth label and label predicted by the generator.

Finally one of the main problems of every Convolutional Neural Network is that it has almost to be considered as a black-box function and the learned visual representation in its layers is scarcely interpretable. For this reason, it is really tough to understand and debug the cases in which the network fails. In particular for the goal of the thesis would be extremely useful to understand why the network fails to identify the lanes in the most challenging examples and possibly fix without the need of millions of images.

Bibliography

- [1] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.
- [3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs (2016). *arXiv preprint arXiv:1606.00915*, 2016.
- [4] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.
- [5] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):263–272, 2018.
- [6] Alexandru Gurgian, Tejaswi Koduri, Smita V Bailur, Kyle J Carey, and Vidya N Murali. Deeplanes: End-to-end lane position estimation using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 38–45, 2016.

- [7] Seokju Lee, In So Kweon, Junsik Kim, Jae Shin Yoon, Seunghak Shin, Oleksandr Bailo, Namil Kim, Tae-Hee Lee, Hyun Seok Hong, and Seung-Hoon Han. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1965–1973. IEEE, 2017.
- [8] Jiman Kim and Chanjong Park. End-to-end ego lane estimation based on sequential transfer learning for self-driving cars. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1194–1202. IEEE, 2017.
- [9] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. *arXiv preprint arXiv:1712.06080*, 2017.
- [10] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. *arXiv preprint arXiv:1802.05591*, 2018.
- [11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

- [15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [18] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.
- [20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.
- [23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

- [24] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.
- [25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.
- [26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [27] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.
- [28] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.
- [29] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.
- [30] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian D Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Cvpr*, volume 1, page 5, 2017.
- [31] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.
- [32] Bin Yu and Anil K Jain. Lane boundary detection using a multiresolution hough transform. In *Image Processing, 1997. Proceedings., International Conference on*, volume 2, pages 748–751. IEEE, 1997.

- [33] Mirko Felisa and Paolo Zani. Robust monocular lane detection in urban environments. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 591–596. IEEE, 2010.
- [34] Hsu-Yung Cheng, Bor-Shenn Jeng, Pei-Ting Tseng, and Kuo-Chin Fan. Lane detection with moving vehicles in the traffic scenes. *IEEE Transactions on intelligent transportation systems*, 7(4):571–582, 2006.
- [35] Ioannis Katramados, Steve Crumpler, and Toby P Breckon. Real-time traversable surface detection by colour space fusion and temporal analysis. In *International Conference on Computer Vision Systems*, pages 265–274. Springer, 2009.
- [36] Amol Borkar, Monson Hayes, and Mark T Smith. Robust lane detection and tracking with ransac and kalman filter. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3261–3264. IEEE, 2009.
- [37] Hui Kong, Jean-Yves Audibert, and Jean Ponce. Vanishing point detection for road detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 96–103. IEEE, 2009.
- [38] Albert S Huang, David Moore, Matthew Antone, Edwin Olson, and Seth Teller. Finding multiple lanes in urban road networks with vision and lidar. *Autonomous Robots*, 26(2-3):103–122, 2009.
- [39] Christina Gackstatter, Patrick Heinemann, Sven Thomas, and Gudrun Klinker. Stable road lane model based on clothoids. In *Advanced Microsystems for Automotive Applications 2010*, pages 133–143. Springer, 2010.
- [40] Ernst D. Dickmanns and Birger D. Mysliwetz. Recursive 3-d road and relative ego-state recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):199–213, 1992.
- [41] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):16–26, 2008.

- [42] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using b-snake. *Image and Vision computing*, 22(4):269–280, 2004.
- [43] Jiang Ruyi, Klette Reinhard, Vaudrey Tobi, and Wang Shigang. Lane detection and tracking using a new lane model and distance transform. *Machine vision and applications*, 22(4):721–737, 2011.
- [44] Joel C McCall and Mohan M Trivedi. Visual context capture and analysis for driver attention monitoring. In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pages 332–337. IEEE, 2004.
- [45] Dario D Salvucci. Inferring driver intent: A case study in lane-change detection. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 48, pages 2228–2231. SAGE Publications Sage CA: Los Angeles, CA, 2004.
- [46] Wilfried Enkelmann. Video-based driver assistance—from basic functions to applications. *International Journal of Computer Vision*, 45(3):201–221, 2001.
- [47] Radu Danescu and Sergiu Nedevschi. Probabilistic lane tracking in difficult road scenarios using stereovision. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):272–282, 2009.
- [48] Bei He, Rui Ai, Yang Yan, and Xianpeng Lang. Accurate and robust lane detection based on dual-view convolutional neural network. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 1041–1046. IEEE, 2016.
- [49] Tusimple: Lane detection challenge, 2017. URL: <http://benchmark.tusimple.ai>.
- [50] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

- [51] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014.
- [52] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.
- [56] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.
- [57] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- [58] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. *arXiv preprint arXiv:1803.06815*, 2018.

- [59] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 240–248. Springer, 2017.
- [60] Yen-Chang Hsu, Zheng Xu, Zsolt Kira, and Jiawei Huang. Learning to cluster for proposal-free instance segmentation. *arXiv preprint arXiv:1803.06459*, 2018.
- [61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [62] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.
- [63] Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 2, 2017.