



Jörg M. Schlager, BSc.

From Aerial Images to Virtual Driving Environments

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Friedrich Fraundorfer

Institute for Computer Graphics and Vision
Inffeldgasse 16, 8010 Graz, Austria

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date

Signature

Access Restriction

This thesis has been commissioned and funded by AVL List GmbH (hereinafter AVL). It contains descriptions of algorithms, methods and software implementation details which are not indented for the general public. According to the contract for services between the author and AVL, access to this document thus has to be restricted.

Acknowledgments

Special thanks go to the team and management of AVL's skill area "DV - Vehicle". Not only have I had the pleasure of working in this department for the past eight years, but this collaboration and the support from my colleagues also created an opportunity for writing a master's thesis at a junction point of academic and industrial research.

In this context, my adviser at AVL, Dr. Rupert Scheucher, deserves particular mention for his part in defining the topic for this thesis as well as his continuous assistance and encouragement. In addition, I would also like to acknowledge Mr. Georg Knoll, Dr. Jürgen Holzinger and Mr. Thomas Schlömicher, as they all provided measurement data for the validation of this work.

Furthermore, this thesis would not have been possible without the support from Mr. Michael Maurer at the Institute for Computer Graphics and Vision (ICG), whose experience in training neural networks was invaluable during the most difficult phase of the implementation.

I am grateful to Prof. Dr. Friedrich Fraundorfer, my supervisor at the ICG, for providing ideas and insights as well as reviewing this work. Not only did he patiently lead me on the right track towards a successful completion, he also kept me motivated during periods of doubt and technical difficulties.

Last but definitely not least, I deeply appreciate the support from my wonderful wife Catrin, who reviewed countless draft pages and always remained sympathetic during a challenging time. This thesis is dedicated to our amazing daughter Elisabeth.

Third-Party Material

Throughout this work, aerial images obtained from basemap.at¹, a digital terrain model provided by Land Steiermark² and map data contributed by the OpenStreetMap³ project are being used. The aerial images and the terrain model are available under the Creative Commons By Attribution (CC BY 3.0 AT) License while the OpenStreetMap database is redistributed under the Open Data Commons Open Database License (ODbL).

¹Stadt Wien und Österreichische Länder bzw. Ämter der Landesregierung, 2015

²*Digitales Geländemodell - 10m - Steiermark*. (Land Steiermark, 2016a).

³Haklay and Weber, 2008

Abstract

The use of simulation tools as part of a modern vehicle development process often relies on the availability of realistic virtual driving environments. We present a multi-step workflow for modeling such virtual landscapes based on publicly available data sources.

At the heart of the proposed system, the semantic segmentation of orthographic aerial images is predicted by a convolutional neural network. We evaluate two different network architectures using a newly generated ground truth data set and show that the well-known VGG16-network produces excellent results for our application. A significant improvement of those results is furthermore achieved by means of a fully integrated, conditional random field.

The segmented images are then used as foundation for the refinement of OpenStreetMap roads. We parameterize our model in terms of the location of each way-point and the corresponding road width and aim to align this data with the original aerial images. Moreover, we analyze the current state of the art and demonstrate that our newly developed algorithm, based on nonlinear optimization and with a stronger focus on road topographies, can provide even better results. In addition, the same optimization tools can be exploited to create smooth road surfaces from low resolution digital terrain models.

Finally, the proposed methods are applied to create a three-dimensional visualization of a virtual landscape. We conclude this example by showing that a vehicle simulation can successfully be conducted in this environment, thus establishing the suitability of the proposed workflow for the task at hand.

Keywords: Virtual Environment, Vehicle Simulation, Aerial Image, Neuronal Network, Convolutional Neuronal Network, CNN, Semantic Segmentation, CRF, Non-linear Optimization, NLO, Visualization, Render Engine

Kurzfassung

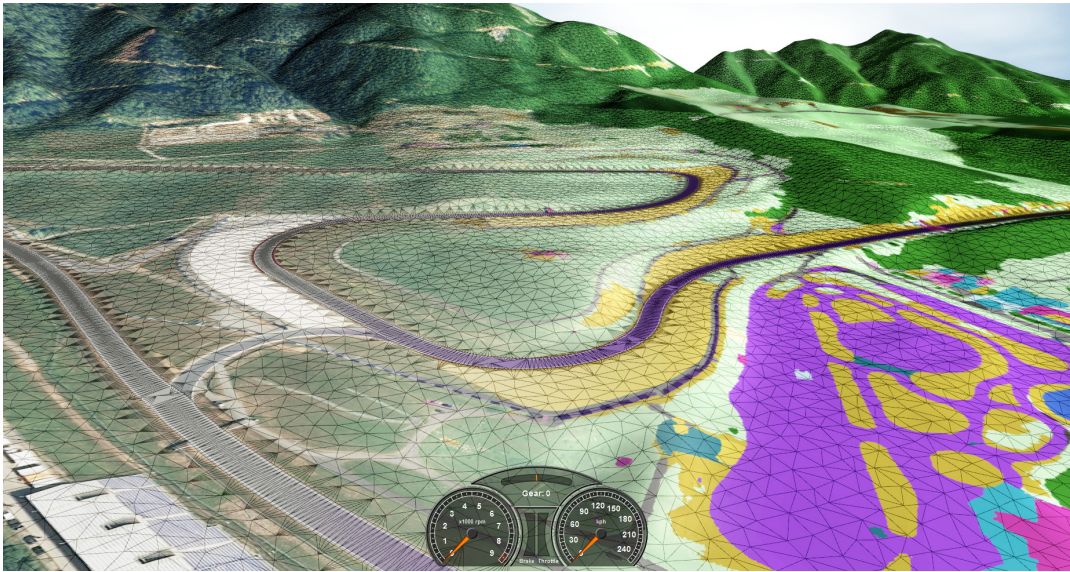
Realistische virtuelle Umgebungen sind eine grundlegende Voraussetzung um die Entwicklung von Fahrzeugen und Steuergeräten durch Simulationsmodelle zu unterstützen. Wir präsentieren einen mehrstufigen Prozess für die Erzeugung geeigneter Landschaften aus frei verfügbaren Datenquellen.

Kern der vorgestellten Methode ist die semantische Segmentierung von Luftbildern mittels faltender neuronaler Netzwerke. Wir evaluieren zwei verschiedene Architekturen anhand eines neu erstellten Referenzdatensatzes und zeigen, dass das bekannte VGG16-Netzwerk auch in diesem Anwendungsfall exzellente Ergebnisse generiert. Eine weitere signifikante Verbesserung ist durch ein voll in das neuronale Netzwerk integriertes Conditional Random Field möglich.

Die segmentierten Luftbilder dienen im nächsten Schritt als Grundlage für die Optimierung von bestehenden OpenStreetMap Kartendaten und die Abschätzung der jeweiligen Straßenbreite. Dazu analysieren wir den aktuellen Stand der Technik und demonstrieren, dass ein neu entwickelter Algorithmus basierend auf nichtlinearer Optimierung unter starker Berücksichtigung der Straßentopographie qualitativ bessere Ergebnisse liefert. Anhand von grob aufgelösten Geländemodellen können die Kartendaten unter Verwendung derselben Optimierungswerkzeuge anschließend auch dreidimensional ausgewertet werden.

Schlussendlich wird die Anwendung der genannten Methoden anhand eines konkreten Beispiels bis hin zur vollständigen, dreidimensionalen Visualisierung einer virtuellen Landschaft thematisiert. Die erfolgreiche Durchführung von Fahrzeugsimulationen in dieser Umgebung dient als abschließender Beleg für die Eignung des präsentierten Verfahrens.

Stichworte: Virtuelle Umgebung, Fahrzeugsimulation, Luftbild, Neuronales Netzwerk, CNN, Semantische Segmentierung, CRF, Nichtlineare Optimierung, NLO, Visualisierung



Source of original aerial image: basemap.at; License: CC-BY 3.0 AT.

Contents

1	Introduction	1
1.1	Virtual Roads for Vehicle Simulations	1
1.2	Requirements for Virtual Roads	2
1.3	Problem Statement and Background	3
1.3.1	Road Map Extraction	3
1.3.2	Modeling the Environment	4
1.4	Constraints, Objectives and Delimitation	6
1.5	Outline of the Thesis	7
2	Semantic Segmentation of Aerial Images	9
2.1	Background and Methodology	9
2.1.1	Machine Learning	9
2.1.2	Neural Networks	11
2.1.3	Classification Networks	19
2.1.4	Classification Benchmarks and State of the Art	22
2.1.5	From Classification to Semantic Segmentation	24
2.1.6	Conditional Random Fields for Refinement	28
2.2	Training and Validation Data	30
2.2.1	A Projection for Aerial Images	30
2.2.2	Image Resolution	32
2.2.3	Surface and Object Classification	32
2.2.4	Ground Truth Data Sets	32
2.3	Segmentation Experiment	34
2.3.1	Deep Learning with Caffe	34
2.3.2	Network Architectures	34
2.3.3	Supervised Training	35
2.3.4	Discussion of Results	37
2.4	CRF Refinement	45
2.4.1	Estimation of Mean-Field Parameters	46
2.4.2	Supervised Training	47
2.4.3	Discussion of Results	48
2.5	Summary	53

3	Road Map Refinement	54
3.1	Background and Methodology	55
3.1.1	Map Projections	55
3.1.2	Validation Data	57
3.2	Refinement by Inference	58
3.2.1	Markov Random Fields	58
3.2.2	Image-based Features	58
3.2.3	Evaluation of Results	60
3.3	Refinement by Nonlinear Optimization	62
3.3.1	Nonlinear Least-Square Optimization Problems	62
3.3.2	Problem Formulation	64
3.3.3	Experiments and Results	67
3.4	Three-dimensional Road Maps	71
3.4.1	Problem Formulation	71
3.4.2	Optimization Method	72
3.4.3	Experiments and Results	73
3.5	Summary	75
4	Visualization and Practical Applications	76
4.1	Render Engine	76
4.2	Mesh Generation	77
4.3	Application Example	78
4.3.1	Generation of a Virtual Driving Environment	78
4.3.2	Simulation Results	82
5	Conclusion	84
5.1	Interpretation of Results	84
5.2	Limitations	85
5.3	Outlook and Future Work	85
	Bibliography	86
	Miscellaneous Resources	90
	Appendices	92
A	Ground Truth Data	92
A.1	Pixel Classes	92
A.2	Ground Truth Images	95
A.3	Distribution of Pixel Classes	95
B	Network Architecture	97
B.1	VGG16-8s	97
B.2	GoogLeNet-8s	99
C	Segmentation Results	103
D	Third Party Software	104

List of Figures

1	From Aerial Images to Virtual Driving Environments	iv
1.1	Pipeline Overview	7
2.1	Neuron Model	12
2.2	Error Surface	13
2.3	Hidden Units	15
2.4	Classification Network Architecture	19
2.5	Convolution	20
2.6	Receptive Field and Stride	21
2.7	Segmentation Network Architecture	24
2.8	Inclusion over Union	27
2.9	Web Map Tile Service	31
2.10	Ground Truth Data for Segmentation (I)	33
2.11	FCN Training Analysis	36
2.12	Segmentation Stride	38
2.13	Pseudocolor Images	39
2.14	Segmentation Performance (IoU)	41
2.15	GoogLeNet Confusion Matrix	43
2.16	VGG16 Confusion Matrix	44
2.17	Network with CRF Layers	45
2.18	CRF Training Analysis	47
2.19	Segmentation Results II	49
2.20	CRF Segmentation Results (IoU)	50
2.21	CRF Confusion Matrix	52
3.1	Universal Transverse Mercator (UTM) System	56
3.2	Validation Data	57
3.3	Random Forest Classification	59
3.4	Refinement Results	61
3.5	Road Graph Refinement	64
3.6	Classification Error Cost	66
3.7	Road Graph Interpolation	67
3.8	Road Graph after NLO-based Refinement.	70
3.9	Road Gradient Energy vs. Height Error	74
3.10	Three-Dimensional Refinement	75

4.1	AVLViewer Screenshot	77
4.2	Workflow Expample Part I	80
4.3	Workflow Expample Part II	81
4.4	Simulation Results	83
4.5	Simulation Example	83
A.1	Ground Truth Data for Segmentation (II)	94
A.2	Pixel Class Distribution	96

List of Tables

2.1	Speed vs. Accuracy I	37
2.2	CRF Kernel Parameters	46
2.3	Speed vs. Accuracy II	48
3.1	Refinement Results for Inference	61
3.2	Cost Factors	65
3.3	Refinement Weights for Nonlinear Optimization	68
3.4	Refinement Results for Nonlinear Optimization	68
3.5	Weights for Nonlinear Optimization of Road Height	73
3.6	Road Height Estimation	73
A.1	Pixel Classes for Various Terrains	92
A.2	Pixel Classes for Buildings	93
A.3	Pixel Classes for Transportation	93
A.4	Auxiliary Pixel Classes for Race Tracks	93
A.5	Pixel Class Distribution	95
C.1	Segmentation Performance (IoU)	103
D.1	Third Party Software	104

List of Abbreviations

ADAS	A dvanced D river A ssistance S ystems
ALS	A irborne L aser S canning
CDT	C onstrained D elaunay T riangulation
CNN	C onvolutional N eural N etwork
CRF	C onditional R andom F ield
COCO	(Microsoft™) C ommon O bjects in C Ontext challenge
DEM	D igital E levation M odel
DoF	D egrees O f F reedom
DSM	D igital S urface M odel
DTM	D igital T errain M odel
FCN	F ully C onvolutional N etwork
GPS	G lobal P ositioning S ystem
ILSVRC	I mageNet L arge S cale V isual R ecognition C hallenge
IMU	I nertial M easurement U nit
IoU	I nclusion o ver U nion
LIDAR	L ight D etection A nd R anging
LoD	L evel o f D etail
MAP	M aximum a P osteriori (estimate)
MLS	M obile L aser S canning
MRF	M arkov R andom F ield
OSM	O pen S treet M ap
PASCAL	P attern A nalysis, S tatistical modelling and C omputational L earning
RDE	R eal D riving E missions
RNN	R eurrent N eural N etwork
UTM	U niversal T ransverse M ercator - P rojection
VOC	V isual O bject C lasses
WMTS	W eb M ap T ile S ervice
XiL	X (may stand for H ardware, M odel or S oftware) i n the L oop

Chapter 1

Introduction

1.1 Virtual Roads for Vehicle Simulations

Front-loading vehicle development by the use of simulation tools is of vital importance to the automotive industry, as it reduces the number of required prototypes and the time-to-market for new products. In consequence, the evaluation of physical components, new algorithms and embedded control units in a simulated environment - which is known as X-in-the-loop (XiL) simulation - has become a widely accepted procedure.

A common use case is the prediction of a vehicle's performance and the impact of modifications to vehicle components on a given virtual test track. Advanced applications may also include a driving simulator, where a human driver-in-the-loop experiences all modifications to a (partially) simulated vehicle first hand. In this context, not only the track but also the surroundings are of interest and have to be visualized in real time. Recent emission regulations, complex new technology such as advanced driver assistance systems (ADAS) and a restricted number of available track test days all further add to the significance of providing detailed virtual driving environments for real-time vehicle simulations.

While there are approaches focusing on the creation of purely procedural, fictional road networks (Campos et al., 2015), modeling real roads has notable advantages for many applications. Most importantly, a faithful recreation of existing road topography facilitates correlations between simulation results and actual measurement data. It is therefore beneficial for the initial parameterization and validation of simulation models and it also increases the relevance of simulation results with respect to solving real-life problems.

If the virtual environment is close to reality, a driving simulator may not only be used to optimize the setup of race cars but it may also assist drivers in mastering new tracks. Likewise, a modern engine testbed may be employed to predict the outcome of real driving emissions (RDE) tests during early development stages. The automatic creation of large-scale virtual driving environments, which mirror existing, real-world roads, is therefore of practical relevance to today's industry.

1.2 Requirements for Virtual Roads

In the context of XiL simulations, we can identify three distinct groups of requirements for virtual roads: In the first place, there are functional needs and specifications which have to be met in order to create stable simulations. Secondly, application-specific minimum standards for the reproduction quality of real-world environments have to be observed. Last but not least, there also is a strong demand for real-time visualizations of said environments.

Starting out with the functional requirements, virtual three-dimensional roads have to exhibit certain topographic properties in order to be usable for the purpose of vehicle simulations. A steady progression of the curvature over distance and a smooth definition of both gradient and banking are key factors in ensuring numerically stable simulations. This may be seen as an analogy to the construction of real roads, where engineers aim to create a smooth ride for the comfort and safety of drivers. Applying the same design principles - such as the use of clothoids for the transitions between corners and straights - to virtual roads therefore not only increases the stability of the simulation but also contributes to the generation of meaningful results.

In addition, any model which is to be executed on real-time systems must also meet hard performance requirements. It is however difficult to quantify the limits exactly, as the rate of simulation is usually adjustable and varies between applications. Based on this author's personal experience with a variety of vehicle dynamics packages^{1,2,3}, it is nonetheless assumed that any industrial grade simulation calls for the ability to evaluate the road model at least 1.000 times per second per tire.

At this calculation rate, a vehicle traversing an urban area with an estimated average velocity of 10 m/s would thus sample the virtual environment at centimeter-level resolution. This renders any exact recreation of real-world, large-scale road topographies impractical, as an enormous amount of data would have to be available in-memory in order to be analyzed under hard real-time conditions. Therefore, a good compromise between the contradictory demands for large open world scenarios and accurate, detailed representations of real-world roads has to be achieved. The balance may however be shifted in either direction, based on the application at hand. For the simulation of a typical RDE test, it is sufficient to provide a low resolution road network and terrain model which albeit has to enclose hundreds of square kilometers. On the other end of the spectrum, the simulation of a permanent race track requires a highly detailed model of a confined environment, as the exact locations of curbs and potholes will influence the racing line and lap times.

Finally, virtual roads and their surroundings also have to be visualized. The creation of interactive driving simulators is a very prominent usage example. Another application is the visualization of abstract simulation results as a post-processing step. While the physics simulation may be completely separated from the render engine,

¹AVL List GmbH, 2016. *AVL VSMTM*. <https://www.avl.com/-/avl-vsm-vehicle-simulation>

²IPG Automotive GmbH, 2016. *CarMaker[®]*. <http://ipg.de/simulationsolutions/carmaker>

³Mechanical Simulation, 2016. *CarSim[®]*. <https://www.carsim.com/products/carsim>

both components are connected by a shared definition of the virtual road. Any suitable road data model therefore must also include the additional information which is necessary for the generation of realistic three-dimensional landscapes, although these may be rendered at a reduced level of detail (LOD).

1.3 Problem Statement and Background

All things considered, several state of the art applications for vehicle simulations require virtual representations of real, existing roads which

1. steadily follow smooth trajectories,
2. can be evaluated on real-time systems,
3. strike a balance between world scale and resolution and
4. act as inputs to both simulation and visualization.

The creation of virtual driving environments, which fulfill these requirements, involves two related tasks: First of all, measurement data has to be interpreted in order to create a smooth, vectorized representation of an existing road network. Secondly, a three-dimensional terrain and environment model, which seamlessly integrates the road surfaces, has to be created for the purpose of adding context and realism to the virtual scene. To these ends, the utilization of a wide variety of different data sources and algorithms has been suggested.

1.3.1 Road Map Extraction

The analysis of aerial images is an obvious choice for the automatic replication of road networks. Hinz and Baumgartner (2003) proposed a strategy involving the exploitation of multiple views of the same scene and a digital surface model (DSM) of moderate resolution in order to mitigate the influence of occlusions. In addition, they used detailed knowledge about roads and their contexts in order to formulate scale-dependent models for the extraction of urban road networks.

Clode et al. (2004) presented a method which automatically detects roads solely from airborne laser scanning (ALS) data. Their method initially creates a digital terrain model (DTM) from a point cloud of airborne light detection and ranging (LIDAR) data. Roads are then extracted by analyzing the height and intensity variations of the measured point cloud with respect to the DTM. Finally, a set of filters is applied in order to create a binary labeled image.

A combination of the previously mentioned schemes was suggested by Hu et al. (2004). Their system fuses clues obtained from LIDAR data and high resolution aerial images in order to detect grid roads in dense urban areas. The height information permits to eliminate uncertainties due to occlusions caused by high-rise buildings, whereas the processing of aerial images provides contextual information on the scene. This is exemplified by image classification methods, which are applied for the detection of vehicles in order to distinguish parking lots from actual roads.

Cao and Krumm (2009) proposed a method for the automatic extraction of road network information solely from GPS traces. They created a routable road graph by grouping individual GPS traces with the help of simulated potential energy wells. This approach however only generates a single trajectory for each edge in the graph and does not provide any further information on the topography of the roads. The resulting road map's level of detail therefore is comparable to community driven, manual mapping efforts such as OpenStreetMap⁴. Mattyus et al. (2015) however demonstrated that a coarse, graph-based road map can be efficiently refined and augmented by exploiting aerial images. This process also results in detailed estimates for the width of each road segment.

A full three-dimensional reconstruction of road surfaces with the help of mobile laser scanning (MLS) is an interesting option whenever a faithful and detailed reproduction of the real world is of utmost importance. Yang et al. (2013) proposed a method for delineating roads from large-scale point clouds which relies on the detection and refinement of curb points. Their algorithm involves the partitioning of a point cloud in consecutive cross sections in order to filter out candidates for curb points. A refinement stage, which assumes a similar geometry of sequential cross sections, considerably improves the results. This strategy was successfully applied for the extraction of road boundaries in residential and downtown areas.

The above list of methods is by no means exhaustive. The topic of creating road maps with added topographical information has been researched at great length, and any combination of the given approaches may be used to achieve viable results.

1.3.2 Modeling the Environment

The generation of a three-dimensional, realistic virtual driving environment requires not only a definition of the road network but also a depiction of the terrain and scenery. Incidentally, the process of creating a suitable scene description is in many ways similar to the extraction of road features from measured data. First of all, many of the previously mentioned data sources - such as aerial images, digital elevation or surface models and aerial or mobile laser scanning data - may also be used for the recreation of buildings, forests and waterways.

The low spatial resolution of most publicly available data sources however makes it extremely difficult to deduce the exact three-dimensional shape of scenery items with an acceptable degree of accuracy. The camera position in a virtual driving environment adds to the significance of this issue, as triangulated background objects are often close to the near plane of the viewing frustum and implausible mesh surfaces thereby are immediately noticeable.

The exact reconstruction of every building and plant alongside the road network is however not absolutely necessary, as there is no direct correlation between the visualization of the landscape and the simulation results. Therefore, it is perfectly acceptable to model the environment using a procedural approach instead of per-

⁴Haklay and Weber, 2008. <http://www.openstreetmap.org>.

forming an accurate three-dimensional inference. To this end, buildings, vegetation and soil types are stored using a small set of parameters each and the actual objects are synthesized in real-time whenever they are visible from the current camera's position. Depending on the application at hand, it may however still be advisable to perform an exact reconstruction of important and recognizable landmarks such as bridges, cathedrals or race control towers in order to increase the authenticity of the procedurally generated landscape.

Surface Classification and Object Detection

One solution for the detection of different surface, soil and vegetation types is a pixel-level classification of LIDAR data and aerial images. Charaniya et al. (2004) provided such a method using manually implemented feature descriptors which are based on both color and height variations. This approach produces labels for individual pixels while disregarding spatial connections. Kluckner et al. (2009) suggested to use a more sophisticated but still compact feature representation which is based on covariance descriptors for a small number of properties. In addition, these authors proposed a two-stage algorithm for classification and semantic segmentation which also considers the relationships between neighboring pixels. Hence, the features are initially evaluated by random forest (RF) classifiers while the final labels are inferred from a conditional random field (CRF). This method also was expanded to create superpixel-based semantic segmentations (Kluckner et al., 2010).

Other methods for the detection of buildings and objects have also often been based on a semantic segmentation of the measurement data. Müller and Zaum (2005) exploited prior knowledge about the shape of man-made structures in order to define photometric and geometric features for the robust detection of buildings from aerial images. More recently, Sampath and Shan (2010) presented a method for the segmentation and reconstruction of polyhedral roofs from LIDAR data. In addition, they also suggested the applicability of their approach for the reconstruction of whole buildings, as they observed topographically consistent results (Sampath and Shan, 2010, p. 1566).

State of the art algorithms for solving classification and segmentation challenges are however almost exclusively based on deep convolutional neural networks (CNNs) (Krizhevsky et al., 2012; Everingham et al., 2015). Furthermore, Girshick et al. (2014) demonstrated that a CNN which has been trained for a classification task might also achieve up-to-date results for object detection. They also provided evidence that it is possible to transfer the knowledge of a pre-trained CNN to a different domain (Girshick et al., 2014, p. 581). Consequently, a classification network may be trained on a large set of publicly available and thoroughly labeled auxiliary images while the domain-specific fine-tuning of the weights is achieved using a small set of actual orthographic aerial images.

In conclusion, semantic segmentation, classification and object detection are all important tools for the extraction of information from measurement data. These methods are therefore essential for the creation of a parameterized description of

the virtual driving environment. Each topic has however been the focus of intensive research over multiple decades, and a comprehensive comparison of all proposed algorithms is clearly outside of the scope of this thesis. Nevertheless, we will analyze selected state of the art approaches and evaluate their practical applicability to the task at hand.

1.4 Constraints, Objectives and Delimitation

The availability of measurement data - or often lack thereof - is a key factor when selecting a method for the creation of virtual driving environments. Many of the algorithms described in the previous sections have been proven on small and carefully chosen test data sets of extraordinary high quality. In practice, the acquisition of high-resolution measurement data for a specific large-scale region of interest is however often either difficult or expensive⁵. When data is supplied by third parties, the licensing conditions regularly also prohibit the redistribution of substantial parts of the data set and enforce severe restrictions for the release of any derivative works⁶.

The primary objective of this thesis therefore is to suggest a workflow for the creation of virtual driving environments, which provides sophisticated results in the absence of dedicated measurements and adapts well to large-scale, real-world problems. To this end, we propose to apply state of the art computer vision and machine learning algorithms to publicly available data sources which are distributed under permissive licenses. Furthermore, we aim to prove the suitability of a single, integrated pipeline for the extraction of both road networks and detailed scenery descriptions from those same sources.

Consequently, data released under open government regulations is of particular interest. The results of the *Shuttle Radar Topography Mission* (also see Van Zyl, 2001) are a prime example, as the complete digital elevation model of the earth, sampled at a 30 m interval, has recently been placed in the public domain⁷. Likewise, many governments⁸ have committed themselves to provide orthographic aerial images with typical resolutions of 10-50 cm/pixel for unrestricted use as part of their open data strategies.

In an attempt to further narrow down the scope of this work, we assume that a semantic segmentation of said orthographic aerial images is an excellent foundation for all further steps in the workflow. In addition, we will evaluate only segmentation methods based on convolutional neural network architectures, as those have been shown to consistently outperform other algorithms (Everingham et al., 2015, p. 126).

⁵At the time of writing, the Austrian government advertises ALS data with a resolution of 1 point per m at a cost of 80 EUR per km² (Land Steiermark, 2016b).

⁶Master Terms: Google Maps for Work. Intellectual Property Restrictions. <https://www.google.com/work/earthmaps/legal/emea/premium-maps-terms.html>. Accessed 2016-03-16.

⁷United States Geological Survey, 2016. *Shuttle Radar Topography Mission*. <http://srtm.usgs.gov>

⁸By way of illustration: Stadt Wien und Österreichische Länder bzw. Ämter der Landesregierung, 2015. Schweizerische Eidgenossenschaft, 2016. New South Wales Government, 2016.

1.5 Outline of the Thesis

With these constraints and limitations in place, it is now possible to more precisely identify the most important aspects of this thesis. Based on the brief survey of the overall topic given in section 1.3, we can suggest an efficient and well-structured workflow for the creation of virtual driving environments. Figure 1.1 depicts the pipeline of the proposed system and draws special attention to some of the key processes. It also acts as a blueprint for the structure of this thesis, as the remaining chapters are closely linked to the individual steps of this workflow. Each technical chapter thus contains a separate, in-depth discussion of related work and an evaluation of selected state of the art methods for a single, specific task.

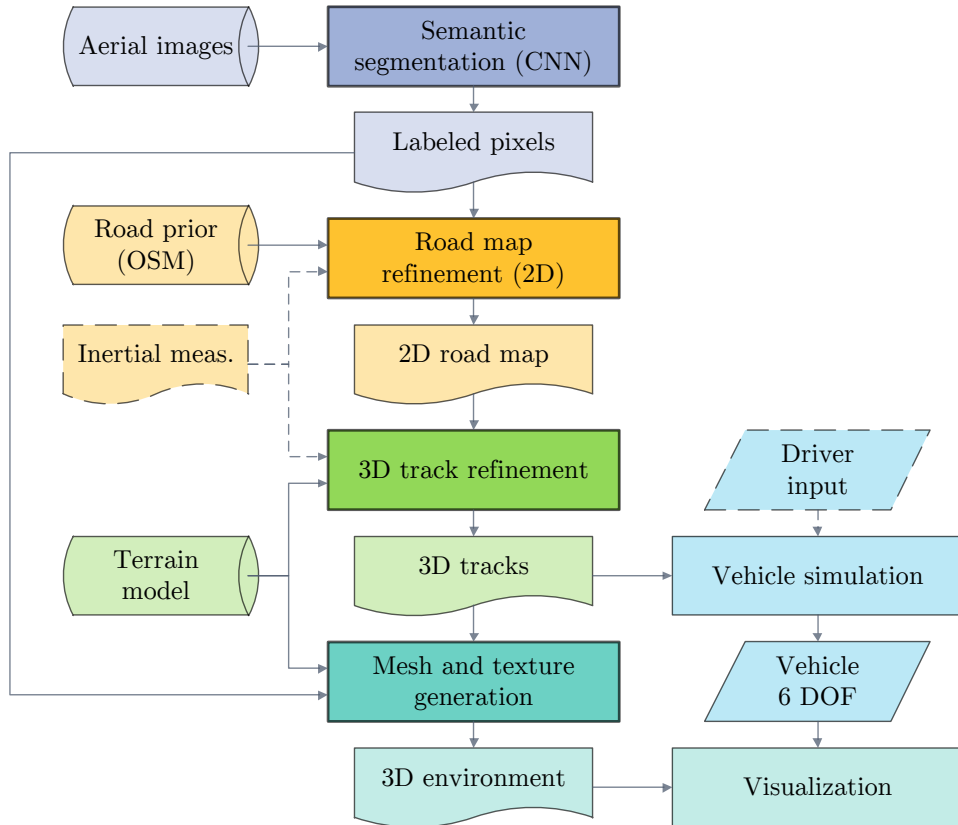


Figure 1.1: Overview of the proposed pipeline for the creation of virtual driving environments from aerial images. The center column contains the key steps in descending order.

Semantic Segmentation of Aerial Images. The main focus of this part is the evaluation of several CNN-based methods for the per-pixel classification and semantic

segmentation of orthographic aerial images. To this end, we discuss different network architectures and demonstrate how a CRF can be used to improve classification accuracy. All experiments are conducted on a small set of manually labeled ground truth orthophotos, which have been created specifically as part of this work.

Road Map Refinement. The suggested process for the creation of a topographically correct map of the road network consists of two consecutive stages. First of all, the results of the semantic segmentation step are utilized to refine road priors obtained from inertial measurements and OpenStreetMap data. For this purpose, a probabilistic method based on the inference of road segment properties in a Markov random field (MRF) is applied (Mattyus et al., 2015). We also formulate an alternative algorithm, which interprets the refinement as a nonlinear optimization problem, and compare the results of both approaches.

The second and final stage extends the description of the road network’s topography to three dimensions. A simple projection of the individual road segments onto a three-dimensional terrain model is however insufficient, as a number of constraints on the resulting surface have to be observed. Therefore, we once again treat this task as a nonlinear optimization problem and define a set of associated cost functions to describe the requirements.

Visualization and Practical Applications. At last, we aim to prove the suitability of the proposed approach by solving real-world problems. Thus, the creation of virtual driving environments by applying the previously described methods and algorithms is demonstrated. The chapter includes the description of a method for the construction of three-dimensional meshes from the previously extracted definitions. Furthermore, the architectural details of a suitable, tile-based render engine are laid out, although the procedural generation of vegetation and buildings is merely implied.

Conclusion. Finally, the thesis is concluded with a summary of the most important findings. Moreover, we reflect on the limitations of the suggested approach and provide ideas for further research.

Chapter 2

Semantic Segmentation of Aerial Images

The first step towards the creation of virtual driving environments is - according to the proposed workflow - the selection of a method for the automatic, per-pixel classification of orthographic aerial images in order to generate a semantic segmentation. As result of the previously established constraints, we interpret this task as a machine learning problem which has to be solved using neural networks. The opening pages of this chapter therefore give an overview of related work and provide the theoretical foundation for the subsequent experiments.

2.1 Background and Methodology

All background material in this section is presented in a progressive way, that is, each new topic builds on previously explained principles. Initially, the task at hand is defined and located in the wide field of machine learning. We then proceed to discuss neural networks and their specific application to image classification. This section is followed by an explanation on how these same classification networks can be adapted to produce individual predictions for each pixel. Finally, we demonstrate that a CRF can be utilized to efficiently model the relationships between neighboring pixels in order to generate a refined semantic segmentation of an image.

2.1.1 Machine Learning

To solve a well-defined and computable problem, we usually formulate an algorithm which produces the desired output from a given input. There are however cases when a task is so complex that a manual implementation of a solution becomes impracticable. A per-pixel classification of aerial images most certainly meets this criterion, as the class of a single pixel depends not only on the local color property but also on yet unknown features of a contextual area which may span the whole image.

Therefore we aim to create a system which independently learns appropriate heuristics in order to resolve the assignment. In this regard, a widely accepted, formal definition of machine learning has been provided by Mitchell:

A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E . (Mitchell, 1997, p. 2)

In addition, programs which fit this description can be further classified into three different groups according to the type of feedback which is available to them (see Russell and Norvig, 2010, p. 694-695):

First of all, there are algorithms for **unsupervised learning**. In this case, the desired output is not yet known and it is therefore impossible to provide any feedback in order to guide the learning process. Programs of this class are however expected to detect similarities in different input examples. Most commonly, such methods are used to cluster homogenous instances based on automatically detected features. A perfect example for the application of this pattern is the automatic grouping of similar and related web documents. To this end, an algorithm has to learn which document features can be used to optimize the intra-cluster similarities and the inter-cluster distances.

A second group of programs learns how to achieve an overall goal in a dynamic environment by finding the optimal actions which lead towards success. This is called **reinforcement learning**, because the feedback to the algorithm is provided by a series of reinforcements, that is rewards or punishments which are based on certain objectives. For instance, a program which learns how to play chess cannot be guided by explicit feedback. Due to the vast number of possible states of the board, it is not feasible to precisely determine the best possible next move. On the other hand, it is obvious that decisive wins should be rewarded and devastating losses should be punished. The program may therefore use this kind of feedback on the outcome of the game in order to deduce which actions consistently yield good results in the long run. (Mitchell, 1997, p. 367ff; Russell and Norvig, 2010, p. 830ff)

Finally, there are problems where a sufficient number of training examples, each consisting of input quantities and the corresponding output data, are readily available. In such a scenario, an algorithm can improve through direct, explicit feedback after each execution. This process is called **supervised learning**, because the training progress is observed and actively directed so that the system can learn from selected examples. Furthermore, we can distinguish between two different types of supervised learning: Problems with continuous output spaces are called **regression** tasks, whereas a discrete output space is characteristic of **classification** assignments.

A semantic segmentation of aerial images could be interpreted as a task which consists of clustering similar pixels, thereby posing an unsupervised learning problem. However, further processing of said images also requires the calculation of a single, predefined label for each cluster in order to recreate different surface types. Therefore

we reformulate the assignment as a supervised classification problem for individual pixels, thus implicitly forming clusters of adjacent elements which are joined together by the same label.

With this stipulation in place, it is now straightforward to describe the proposed technique in terms of the previously given, formal definition of machine learning. A suitable algorithm has to learn heuristics in order to fulfill classification tasks (T) for individual pixels based on experience (E) provided in the form of manually labeled aerial images. The performance (P) of this system can easily be measured by the percentage of correctly labeled output pixels, although we will also use more sophisticated indicators later on.

2.1.2 Neural Networks

Neural networks are applicable to a wide range of machine learning problems. They are however especially useful for solving supervised classification tasks, as training them by providing examples of input-output combinations leads to excellent results. Furthermore, recent studies have pointed out the superiority of classification networks over alternative algorithms such as decision trees (Everingham et al., 2015, p. 126; Russakovsky et al., 2015, p.235). This section therefore briefly describes the fundamental principles of neural networks while also offering some insight into how these networks actually learn to solve problems. A full discussion on these topics can be found in Mitchell (1997, p. 81ff) and Russell and Norvig (2010, p. 727ff).

Historical Approaches

Modeled after the basic physiology of neurons and synapses inside a brain, artificial neural networks are also formed by individual units (nodes) and their weighted connections (links). Early work - dating back as far as 1943 - represented neurons as elements which were either switched on or switched off (McCulloch and Pitts, 1943). The transition to an activated state was the result of a stimulation caused by links to other units. It was shown that logical operators could be implemented by such primitive models, thereby enabling a network of suitable complexity to calculate the result of any computable function. Since then, more fine-grained and realistic representations for nodes have however been developed. (Russell and Norvig, 2010, p. 16)

A Mathematical Model for Neurons

Figure 2.1 visualizes a mathematical model for a single unit of a neural network. The depicted node computes an output value by first calculating a linear combination of real-valued inputs. Then, an activation function f maps this sum to the desired output range. Linear transformations, sigmoid functions and hard thresholds are all common choices for activation functions. As the type of the function f is also often used to describe a specific model, this consequently leads to a designation as linear,

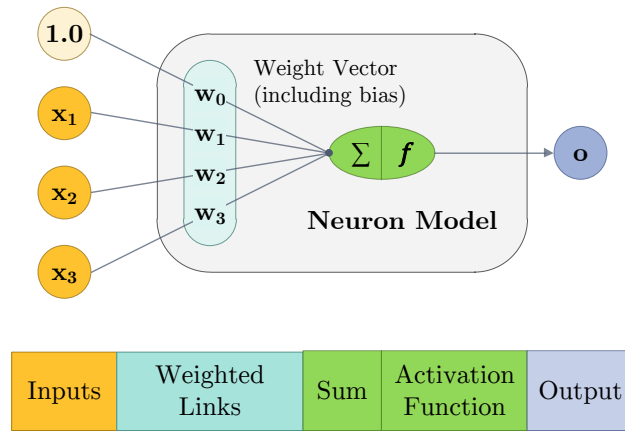


Figure 2.1: Schematic of a model for a single neuron with an additional bias input.

sigmoid and thresholded unit. By adding an additional constant input (and the associated bias weight), it is furthermore possible to translate an activation function along the x-axis. The equation for the output value o is thus

$$o = f \left(\sum_{i=0}^n x_i w_i \right) \quad (2.1)$$

where x_i references the input connections and w_i describes the elements of the weight vector \vec{w} . Through the selection of a convenient activation function and by modifying the elements of the weight vector, this model can be adapted to produce the desired output from given input data.

Training Rules

Whenever suitable labeled examples are available, it is possible to automatically optimize the weight parameters. This optimization process is the method which actually enables a unit to learn from experience. We first analyze the training procedure for thresholded units where the output is calculated as $o = \text{sgn}(\vec{w} \cdot \vec{x})$. Such nodes are also called perceptrons and are used for many simple classification tasks. In the special case of linearly separable examples, the weight optimization can be conducted according to the **perceptron training rule**. To this end, the weights are randomly initialized and iteratively updated whenever the calculated result for an example does not match the desired target output t :

$$\begin{aligned} \Delta \vec{w} &= \eta(t - o)\vec{x} \\ \vec{w} &\leftarrow \vec{w} + \Delta \vec{w} \end{aligned} \quad (2.2)$$

Thereby, η is a positive constant called the **learning rate**, which moderates the influence of individual examples, and $(t - o)$ is the output error. This algorithm has

been proven to converge in a finite number of iterations, provided that the learning rate is small enough. (Mitchell, 1997, p. 89)

The perceptron training rule however only covers a very specific case, as a unit may well include a different activation function and linearly separable training examples are not always available. For all these other scenarios, it is possible to apply the **delta rule** instead, which leads towards a best-fit approximation for the weight vector.

We first derive this rule for a single linear unit where the output is given as $o = \vec{w} \cdot \vec{x}$. The basic idea is to find a weight vector so that the sum of the squared error terms over all examples out of the training data set D is minimized (see Mitchell, 1997, p. 89ff):

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \quad (2.3)$$

On the assumption that D remains constant during training, the elements of the weight vector can be interpreted as axes of a multi-dimensional space. The resulting error $E(\vec{w})$ may then be calculated for each point in this space, thus forming an error surface as shown in figure 2.2.

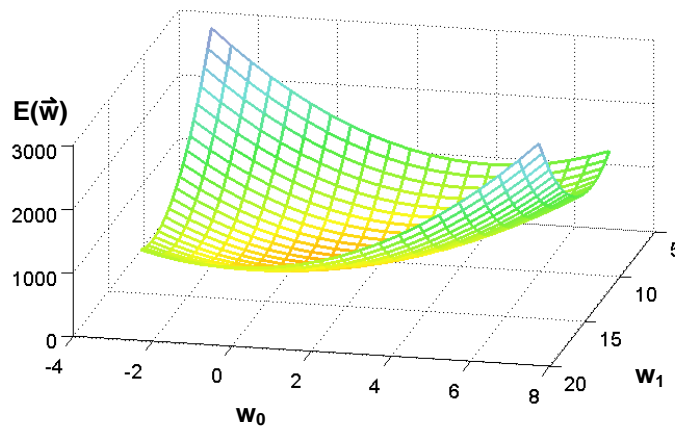


Figure 2.2: Error surface for a weight vector with two elements. The acceptable values for the elements $w_0[-3, +7]$ and $w_1[+6, +16]$ form the hypothesis space.

Therefore, we can apply the **gradient descent** algorithm to minimize the total error. To this end, the direction of the steepest gradient along the error surface is computed by calculating the partial derivatives vector of the error $E(\vec{w})$ with respect to the current values of the weight vector. By incrementally moving in the opposite

direction of this gradient, the output error is ultimately minimized:

$$\begin{aligned}\nabla E(\vec{w}) &\equiv \left[\frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \dots, \frac{\delta E}{\delta w_n} \right] \\ \Delta \vec{w} &= -\eta \nabla E(\vec{w}) \\ \vec{w} &\leftarrow \vec{w} + \Delta \vec{w}\end{aligned}\tag{2.4}$$

While η once again denotes the learning rate, this positive constant is now interpreted as the step size of the algorithm and a negative sign is added to accommodate for the direction of movement. The individual components of the gradient vector $\nabla E(\vec{w})$, as derived in Mitchell (1997, p. 92), are furthermore calculated as:

$$\begin{aligned}\frac{\delta E}{\delta w_i} &= \frac{\delta}{\delta w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\delta}{\delta w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2 (t_d - o_d) \frac{\delta}{\delta w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\delta}{\delta w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ &= \sum_{d \in D} (t_d - o_d) (-x_{i,d})\end{aligned}\tag{2.5}$$

By replacing the components of $\nabla E(\vec{w})$ in equation 2.4 with equation 2.5, we finally deduce the simplified rule for weight updates in the case of linear units according to the gradient descent algorithm:

$$\Delta w_i = \eta \sum_{d \in D} ((t_d - o_d) x_{i,d})\tag{2.6}$$

Stochastic Gradient Descent

An alternative to the computationally expensive derivation of the surface's gradient at each step - which requires the calculation of the exact error over all training examples - is the **stochastic gradient descent** (SGD) algorithm. In this case, the weights are updated after each computation of the error for a single training example, thereby transforming equation 2.6 into:

$$\Delta w_i = \eta (t - o) x_i\tag{2.7}$$

When the learning rate η is chosen small enough, these incremental steps can be a sufficient approximation of the true gradient descent (Mitchell, 1997, p. 94).

On a final note, the perceptron training rule 2.2 appears to be identical to the weight update equation 2.7. Whereas the former is however only applicable to thresholded units, it is possible to derive the delta update rule for all nodes with differentiable activation functions.

Multi-Layer, Feed-Forward Networks

While single units are still used for some basic classification tasks, a direct linear connection between the inputs and outputs is not sufficient for modeling complex relations. Therefore, intricate networks of nodes with non-linear activation functions are necessary in order to solve sophisticated problems such as the semantic segmentation of images. Figure 2.3 depicts a simplified structure that nonetheless contains the basic building blocks of said networks. The individual units in this schematic are

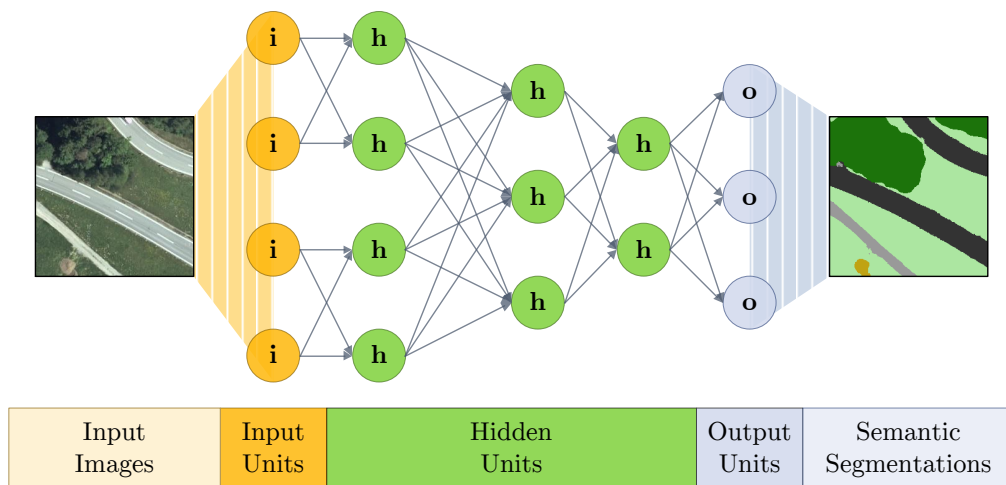


Figure 2.3: Schematic of a feed-forward multi-layer network with hidden units.

organized in several consecutive layers. As edge cycles are strictly prohibited, the nodes and links form a directed acyclic graph (DAG). This specific architecture is called a feed-forward network (also see Goodfellow et al., 2016, p. 168ff). Despite of this simplification, the relation between the weights of the hidden units and the multiple outputs of the network is no longer obvious.

Network Error and Loss Functions

In order to formulate a network training algorithm, we first have to deal with the fact that networks may have many different outputs, all contributing to the total error. One solution is to treat the individual outputs as components of an output vector. This changes equation 2.3 to:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{Outputs}} (t_{d,k} - o_{d,k})^2 \quad (2.8)$$

However, the calculation of an unweighted sum of the output errors may not be a suitable performance measure for all tasks. Specifically, some errors may be more

significant than others, depending on the application at hand. Considering a network which labels e-mails as either spam or genuine content, it is evident that forwarding unwanted messages is preferable to suppressing any of the few important personal messages (Russell and Norvig, 2010, p. 710). Thus, the performance of two networks which produce the same classification error according to equation 2.8 may be assessed quite differently by actual users.

Therefore a more general approach to the calculation of the network error is sometimes needed. To that end, we may define a **loss function** with arbitrary characteristics based on the network's multiple outputs and their corresponding target values for the computation of a single **loss** value. This scalar, which closely correlates with the perceived performance of the network, then acts as a replacement for the objective error and has to be minimized during training. In order to be able to use this strategy in conjunction with certain optimization algorithms, it is however necessary to ensure that said loss function also is differentiable over its entire domain.

Training Feed-Forward Networks

Based on the previous sections, the minimization of a suitable loss function's response by adapting the SGD algorithm is an obvious choice for the training of feed-forward networks. To this end, the outputs of all units in the network once again have to be differentiable. Therefore, networks consisting of sigmoid units are very popular, as these include a non-linear activation function which fulfills all requirements. Furthermore, the σ -function itself is not only bounded but also facilitates the calculation of derivatives and thereby the computation of gradients:

$$\begin{aligned}\sigma(y) &= \frac{1}{1 + e^{-y}} \\ \frac{\delta\sigma(y)}{\delta y} &= \sigma(y)(1 - \sigma(y))\end{aligned}\tag{2.9}$$

By using this relation, the SGD weight update step for a single sigmoid unit with a quadratic loss function can be derived in a similar way to equations 2.5 and 2.6:

$$\Delta w_i = \eta \underbrace{(o(1 - o)(t - o))}_{\text{error term } \delta} x_i\tag{2.10}$$

This equality directly applies to all output units of the network because the training examples include the corresponding target values. However, the error term δ also has to be independently computed for each hidden unit in order to update the matching weights, and the target values for these nodes are not part of the training data set.

In order to find a method for calculating the yet unknown δ -terms, we first reflect on the fact that the inputs of any unit in a feed-forward network are outputs of a previous layer as visualized in figure 2.3. Equation 2.1 therefore leads to the assumption that each input connection also forwards the output error of the preceding unit. These errors are thus passed on and each of them is responsible for a certain fraction

of the subsequent unit's total output error (Russell and Norvig, 2010, p. 733). The relative contributions of said errors thereby depend on the weight parameters which are associated with each input connection.

This observation is very useful, as it can be reversed in order to calculate the δ -terms for hidden units. To that end, we deduce that the output error $(t_h - o_h)$ of any hidden unit is partly responsible for the error terms of all connected nodes in the succeeding layers. Furthermore, the degree of correlation with said subsequent error terms is proportional to the strength of the link between the nodes. This enables us to intuitively formulate an equation for the error term δ_h of a hidden sigmoid node h based on the error terms δ_k of all nodes connected to this hidden units' output:

$$\begin{aligned} (t_h - o_h) &\equiv \sum_k w_{k,h} \delta_k \\ \delta_h &\leftarrow o(1 - o) \sum_k w_{k,h} \delta_k \end{aligned} \tag{2.11}$$

Here, $w_{k,h}$ is the weight parameter of the unit k , which denotes the significance of the output o_h for the calculation of δ_k and thereby determines the connection strength between both nodes. A comprehensive derivation of this update equation is presented by Mitchell (1997, p. 101ff).

By using rule 2.11 to update the weights of hidden units, we can formulate an incremental algorithm for training feed-forward networks based on the SGD method. At first, the weights of the network are initialized with small random values. The optimization then continues by repeating the same steps for each training example until a termination condition defined in terms of the loss value is met:

1. The network is evaluated in forward direction to calculate the output values.
2. Error terms for the output layer are computed according to the delta rule.
3. These errors are then propagated throughout the network in reverse direction. Therefore, the δ -terms are independently calculated for each hidden unit.
4. Finally, the weights of all units are updated based on the learning rate.

The dissemination of the output error starting from the final layer thus is the key concept which enables us to efficiently optimize the weights of feed-forward neural networks in an end-to-end training procedure. Hence, this method is called **the back-propagation algorithm** (Rumelhart et al., 1986).

There are two common variations of this procedure: One approach is to accumulate the gradients over consecutive examples before updating the weights, thereby getting closer to a true gradient descent. A second alteration introduces an additional momentum α in order to update the weights, that is an unit's Δw_i value for the training example (n) also depends on the corresponding value during the previous iteration ($n - 1$) (also see Mitchell, 1997, p.100):

$$\Delta w_i(n) = \eta \delta x_i + \alpha \Delta w_i(n - 1) \tag{2.12}$$

In addition, the value of each weight may also be reduced by a certain - usually quite small - factor after each iteration. This procedure is known as **weight decay** and effectively adds a penalty for large weights, thus avoiding complex error surfaces while favoring simpler hypotheses.

Over-Fitting and Cross Validation

Neural networks are however still susceptible to **over-fitting**. If the size of the network in terms of individual units and weight parameters is large enough, the full set of training examples can be memorized. This leads to excellent classification results on the training data but poor performance in all other scenarios, because the network is not forced to deduce universally valid rules from the provided examples.

It is therefore necessary to actively supervise the training phase to ensure that the network will generalize well. A simple precaution, which helps in recognizing critical advances, is to partition the collection of labeled examples in separate training and validation sets. Furthermore, the performance of the network with respect to the validation data has to be analyzed regularly during training. When the network's accuracy continues to improve on the training set, but results on the validation data deteriorate, then this is a strong indication that over-fitting might have occurred. However, it is still necessary to continue training for some iterations in order to be sure that the abort has not been caused by a local minimum of the validation set error (Mitchell, 1997, p. 111). (also see Russell and Norvig, 2010, p. 708-709)

A more sophisticated method for the prediction of an algorithm's performance on yet unseen test data - which also uses all of the available examples for training - is **cross validation**. To this end, a labeled collection is split into an arbitrary number of k partitions which facilitate a k -fold cross validation. With a small data set of 10 items, it is for instance possible to create 10 partitions of single items ($k=10$) or 5 item pairs ($k=5$). The training phase is then repeatedly executed, so that each partition acts as a validation data set exactly once. The mean value computed from all k individual validation errors is an estimate for the overall performance of the network on unknown data. After the architecture has been validated in this way, all of the examples may be used during a final training loop (Mitchell, 1997, p. 112).

Unfortunately, while the evaluation of neural networks is usually quite fast, the process of training them using large example collections is comparatively slow. A cross-validation approach only aggravates this issue, as the number of required resources increases linearly with the number of generated partitions. It is therefore this author's opinion, that while such a procedure is theoretically superior to other strategies, it is not applicable to the training of all deep neural networks.

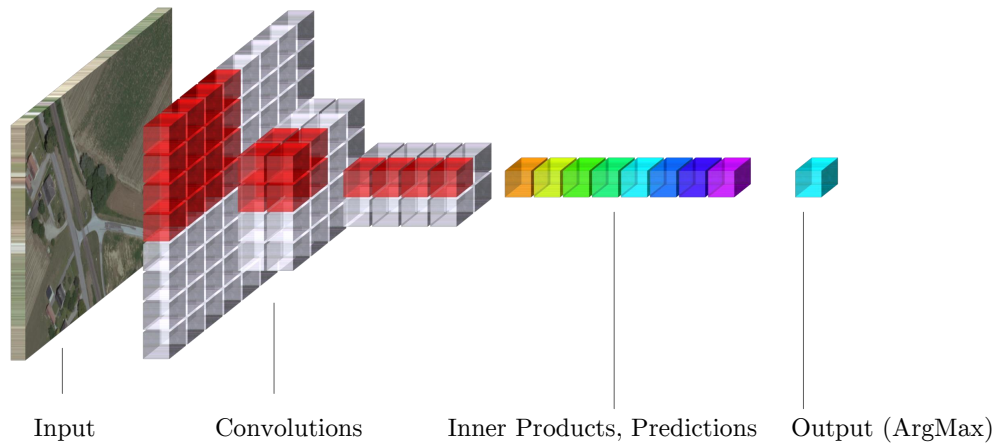


Figure 2.4: Classification network architecture. The visualization demonstrates the dimensions of the output data blobs for each layer.

2.1.3 Classification Networks

State of the art classification networks do not rely solely on sigmoid units. The training rules from section 2.1.2 also apply to all other layer-based, feed forward networks as long as the derivatives of the loss and activation functions can be calculated. Thus, many different layer types with tunable weight coefficients have been developed. Figure 2.4 visualizes a typical architecture of an image classification network which employs some of those layers.

The input to this network is made up from the image’s preprocessed pixel data, where each of the color components forms a separate, two-dimensional matrix. Therefore, the binary large objects (data blobs) which pass through the network can be described in terms of $C \times W \times H$, where C is the number of channels, W the width and H the height of a matrix. The term **bottom blob** is often used to refer to any of the input connections of a given layer. Likewise, the output of a layer is also known as the **top blob** and the final layer of the network, which calculates the loss value, thus is the top layer. The next sections contain a short overview of specialized computer vision network layers.

Convolution Layer

A convolution kernel is a matrix of coefficients which is repeatedly shifted across and multiplied with an input image to create an output image. Different convolution kernels have long been used in image processing in order to apply certain effects or to enhance images. Common applications include sharpening and blurring of pictures as well as enhancing or detecting edges (see figure 2.5).

LeCun et al. (1998) first suggested to utilize these convolution kernels in order to detect yet unknown features in the input data. A key insight is that the coefficients of

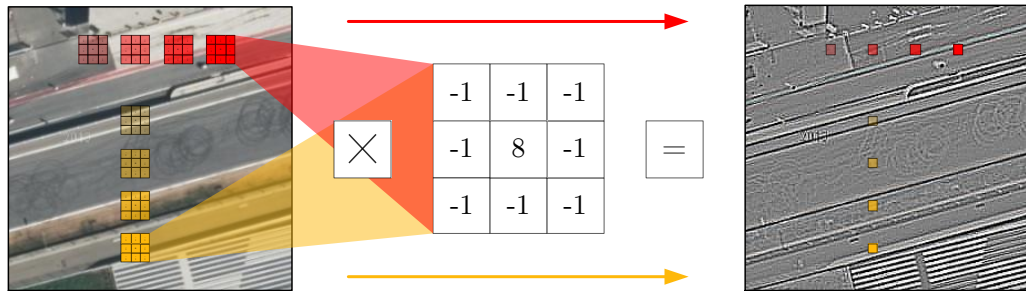


Figure 2.5: Convolution. A 3×3 normalized convolution kernel for edge detection.

a single convolution kernel can be trained in the same way as the coefficients of a single neuron, that is using the back-propagation algorithm. The kernel's coefficients and thus the exact nature of the detected feature are thereby determined automatically during the training stage. (LeCun et al., 1998)

A convolution layer usually contains multiple different kernels and each kernel is responsible for detecting a single significant feature such as a vertical or horizontal line, sharp edges or strong local variations in a single color channel. This feature detection is the foundation of the classification process and thus is generally performed early on. The number of weight coefficients for a convolution layer depends on the number of different kernels and the size of each kernel, but it is independent from the dimension of the input data. In order to combine local features and thereby detect larger and more complex structures, multiple convolution layers with small kernels are often stacked to form deeper networks.

By visualizing the weights of a convolution kernel after the training stage it is also possible to generate an image which is prototypical for the feature this kernel has adapted to. Images can also be generated from multiple consecutive convolutional layers, thereby producing complex artworks which are representative of the network's knowledge. This technique is called inceptionism and has been implemented in Google's Deep Dream¹ software (see Mordvintsev et al., 2015).

Pooling Layer

The pooling layer combines adjacent cells and thereby reduces the number of elements of a data blob. Average pooling is a common variant which simply calculates and promotes the average value of all inputs. A pooling layer is however also often applied directly after a convolution layer in order to identify the highest probability value for a given feature within a small region. This usage is called maximum pooling, and the output thus is the maximum of all inputs.

¹<https://github.com/google/deepdream/blob/master/dream.ipynb/>

Receptive Fields

As we progress from layer to layer, the dimensions of the data blobs may change significantly. In order to describe the relationship between the input and output dimensions of a layer or network, two terms are frequently used: The **receptive field** of an output is composed of all inputs which affect its values. The **stride** on the other hand gives the distance between two receptive fields for adjacent output elements. Using these terms, we can calculate the output width of convolutional and pooling layers from their respective input widths (see Jia et al., 2014):

$$W_{\text{output}} = \frac{W_{\text{input}} + 2 \cdot \text{padding}_{\text{horz}} - W_{\text{kernel}}}{\text{stride}_{\text{horz}}} + 1 \quad (2.13)$$

A formula analogous to equation 2.13 is also used to calculate the height of the output, and figure 2.6 demonstrates both relations.

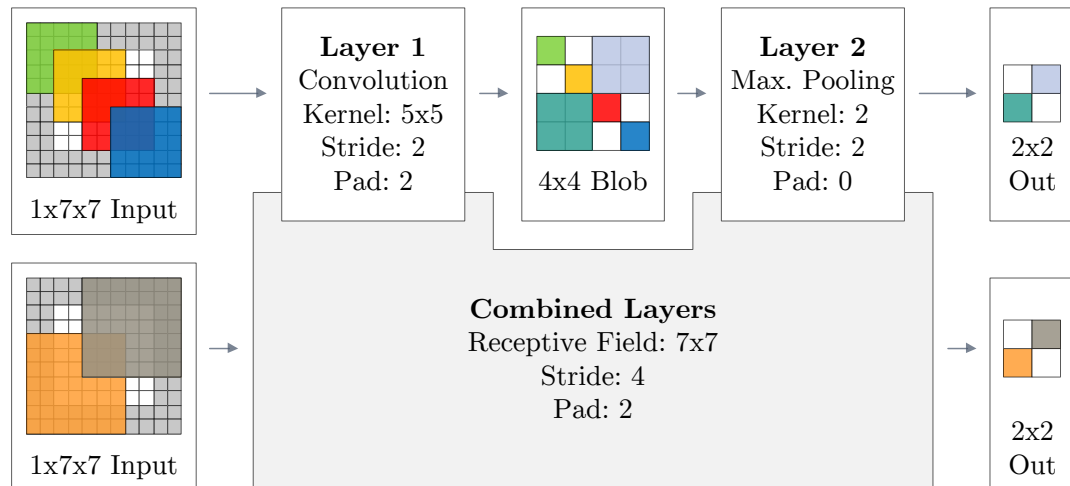


Figure 2.6: Receptive field and stride. Top row: Consecutive application of convolution and max pooling layers. The $1 \times 7 \times 7$ input blob is first padded by 2 pixels, yielding a $1 \times 11 \times 11$ data structure. A single 5×5 convolution kernel is then applied 16 times using a stride of 2 pixels which creates the temporary $1 \times 4 \times 4$ data blob. A max pooling layer further reduces the dimensions to the final $1 \times 2 \times 2$ output blob. Bottom row: Equivalent single-layer network - kernel size, padding and stride all influence the receptive field.

Inner Product Layer

An inner product is a fully connected layer, that is all inputs i are directly connected to all outputs o . The receptive field for each output thus encompasses all inputs. Furthermore, the width and height of the output blob are always 1. Thus, the number of required weight coefficients equals $C_i \cdot W_i \cdot H_i \cdot C_o$. As a result, inner product layers

can contain a huge number of coefficients and are often only used at the final stages of a classification network in order to combine independent features into probability predictions for predefined output classes. After successful training, an inner product layer may for example contain weight coefficients which correctly associate the learned features "furry", "sharp ears" and "vertical pupils" with the predefined output class "cat". A common approach is to use an inner product in order to create a network with a real-valued output vector with T elements, where each element predicts the probability for a single, predefined class.

SoftMax Layer

To train such a classification network, it is necessary to normalize the T predicted output probabilities. The **SoftMax** function is used for this purpose, as it transforms a vector of real valued inputs into a new vector of the same size, whose values add up to a total sum of 1. This function can therefore be used to calculate the desired probability distribution \hat{p} based on the unnormalized input values \vec{x} and the additional weight vector \vec{w} (see Jia et al., 2014). The normalized probability p for the j -th class under the condition of \vec{x} is thereby given as:

$$p(t = j|\vec{x}) = \frac{e^{x_j w_j}}{\sum_{i=0}^{T-1} e^{x_i w_i}} \quad (2.14)$$

ArgMax Layer

In order to solve a multinomial classification problem for T classes, the final output of the algorithm however has to be an index t within the range of $[0, 1, \dots, T-2, T-1]$. The index t is determined by the **ArgMax** function which returns the index of the element with the highest probability.

2.1.4 Classification Benchmarks and State of the Art

The performance of image classification networks is typically assessed using one of several well-known ground truth data sets. One example of such an annotated collection is the ImageNet database (Deng et al., 2009), which is the foundation for the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Russakovsky et al. (2015) provide a retrospective on the history and progress of this competition. The annually held PASCAL Visual Object Classes (VOC) (Everingham et al., 2015) contest is another widely accepted benchmark for the evaluation of classification networks.

Two important measures for comparing the success of a network are the top-1 and the top-5 error rates: The top-1 error is the fraction of images where the label predicted by the ArgMax function differs from the ground truth data, i.e. the network returns a wrong classification result. In a similar way, the top-5 error rate is the fraction of samples where the 5 most likely classes according to the network's prediction do not contain the correct label.

AlexNet

Krizhevsky et al. (2012) first trained a network named AlexNet, which consists of **stacked convolutional layers**, for the ILSVRC 2010 and ILSVRC 2012 challenges. In order to prevent over-fitting, this network also includes **dropout units**, which randomly disable neurons of the first two fully connected layers during training. This additional improvement has been developed by Hinton et al. (2012), who suggest to use a dropout factor of 50% for hidden layers and 20% for input layers.

In consequence, the AlexNet architecture significantly outperformed all other ILSVRC 2012 entries with a top-5 error rate of 15.3%, while the runner-up in this competition achieved a top-5 error of 26.2%. Due to this huge improvement in classification accuracy, AlexNet marks a turning point which popularized the use of CNN networks for classification tasks and strongly influenced many newer approaches (Rusakovsky et al., 2015, p. 232).

Visual Geometry Group (VGG) - Networks

Simonyan and Zisserman (2014) investigated the influence of the depth of a CNN on its performance based on the classic network architecture of LeCun et al. (1998). They concluded that deeper neural networks can result in considerably better classification accuracy. To this end, they presented a homogenous architecture consisting of 16 to 19 layers with 3×3 convolution filter kernels and designated their most successful networks as VGG-16 and VGG-19.

The VGG-19 network is the second-best entry in the ILSVRC 2014 classification challenge with a top-5 error rate of 7.32% and the winner of the ILSVRC localization challenge in the same year. This demonstrates the versatility of this homogenous network architecture, which also has been proven to transfer well to other tasks and data sets. (Simonyan and Zisserman, 2014)

GoogleNet

The GoogleNet architecture by Szegedy et al. (2015) is the winner of the ILSVRC 2014 classification challenge with a top-5 error rate of 6.67%. This entry not only achieved an improvement in classification accuracy, it also has been designed for a more **efficient usage of computational resources**. To this end, it is necessary to limit the number of parameters of the 22 layers deep network. This is partly possible due to multiple "Inception" blocks, which use 1×1 convolutional filters in order to reduce the dimensionality of data blobs. In addition, GoogleNet employs average pooling layers at the top instead of the more common fully connected layers. By these means, it is possible to increase the width and depth of the network while keeping the computational requirements constant (Szegedy et al., 2015). Furthermore, the comparatively small number of network parameters results in proportionally less training and evaluation time.

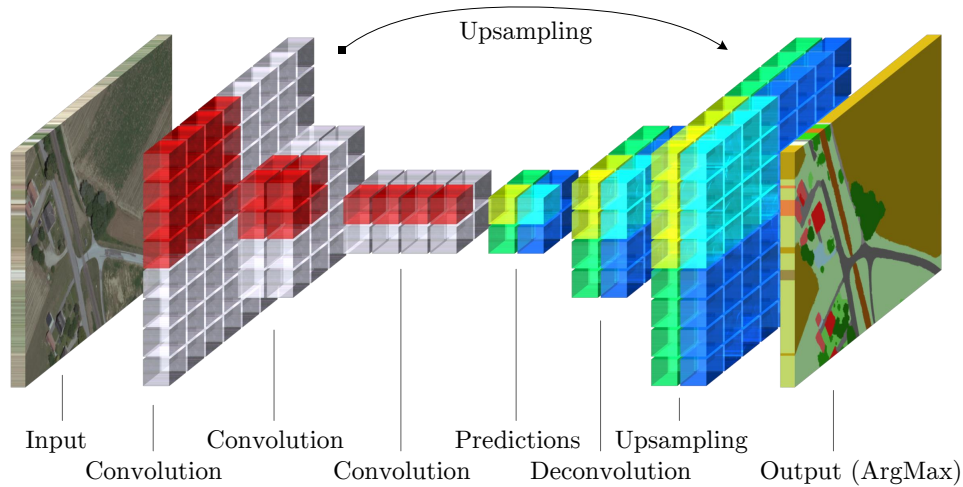


Figure 2.7: Segmentation network architecture. The visualization demonstrates the dimensions of the output data blobs for each layer.

2.1.5 From Classification to Semantic Segmentation

A semantic segmentation network - as shown in figure 2.7 - differs from a classification network as it calculates a separate class for each pixel of the input image. Thus, a huge annotated data set which contains correct labels for all pixels of all examples is required to train such a deep neural network from start to finish. Many large, publicly available ground truth data collections are however only annotated with a single label per image which describes the class of the most prominent foreground object.

Long et al. (2014) therefore suggest to modify the proven classification network architectures described in section 2.1.4 and adapt only the final layers to generate the desired output. The main advantage of this approach is that the convolution layers - which are responsible for the feature detection - can be trained using the simpler classification network. When the classification results are satisfactory, the optimized parameters are then passed on to the segmentation network. Thus, the knowledge of the classification network - which has been obtained during the training on a huge ground truth collection - is reused in order to create a semantic segmentation. In addition, the parameters of the remaining layers can be trained on much smaller example data sets, which may even be created specifically for a given task.

Net Surgery

In order to create a separate prediction for each pixel of an image, the first step is to convert all fully connected layers into convolution layers. The parameters of each convolution layer have to be chosen so that the number of weight coefficients remains unchanged when compared to the initial layer type. This makes it possible to reuse the weights of the inner product layers and transplant them to the new convolution layers. Each convolution kernel thereby inherits the complete knowledge on how to

convert individual feature representations into class labels. This method is known as **net surgery** and has - according to the examples provided with Caffe (see Jia et al., 2014) - first been suggested by Rowland Depp.

By replacing the inner product layers, the output dimensions of the new segmentation network are changed. While a fully connected layer always produces an output blob in the form of $C \times 1 \times 1$, the output size of a convolution layer depends on both this layer's parameters and the input dimensions (as shown in equation 2.13). In addition, the number of weight coefficients is now independent from the image dimensions. As a result, the new network - which is now a **fully convolutional network** (FCN) - can be used to segment inputs of arbitrary size, as larger input images merely increase the width and height of the prediction blob.

The exact relationship between the input image's dimensions and the number of returned predictions however varies with the architecture of the classification network. The two key factors are once again the size of the receptive field for a single output pixel and the stride between two adjacent outputs, which both depend on the parameters of all previous convolution and pooling layers:

$$W_{\text{prediction}} = \frac{W_{\text{input}} - W_{\text{receptive}}}{\text{stride}_{\text{horz}}} + 1 \quad (2.15)$$

When a net surgery is performed on a classification network, the resulting total stride is often of considerable magnitude. As demonstrated by equation 2.15, the number of predictions and thus the resolution of the output blob will therefore be much smaller than the size of the input image. Using numbers taken from real applications, we can provide a meaningful example: A network with a receptive field of 224×224 pixels and an effective stride of 32 pixels will convert a 512×512 pixel input image into a 9×9 blob of class predictions. To provide a prediction for each input pixel, it is thus necessary to add additional layers in order to increase the dimensions of the network's output so that the size of the input image can be matched.

Deconvolution Layer

The preferred way of increasing a data blob's resolution is through the use of deconvolution layers. Recalling section 2.1.3, a convolution multiplies an input image with a given kernel to achieve a certain effect or detect a specific feature. A deconvolution on the other hand reverses this process in order to partially recover an original input image from an output blob and known kernel coefficients. Thus, a deconvolution layer uses the same parameters as a convolution layer: It is defined by the number of different kernels, their sizes and respective weight coefficients and the stride. As a result, the formula for the output dimensions of a deconvolution layer can be directly derived from equation 2.13:

$$W_{\text{output}} = (W_{\text{input}} - 1) \cdot \text{stride}_{\text{horz}} - 2 \cdot \text{padding}_{\text{horz}} + W_{\text{kernel}} \quad (2.16)$$

When the stride parameter is chosen greater than 1, a deconvolution layer may thus be used to create an output data blob whose dimensions are larger than the input

dimensions. This layer is therefore commonly used to upsample the class predictions for an image in order to increase the resolution of the semantic segmentation.

Fuse Layer

The upsampling process however is not able to restore finer details and results in blurred shapes without sharp edges. Long et al. (2014) therefore suggest to also use information from earlier layers with a lower overall stride and fuse it with the output of the deconvolution layers. This results in more detailed class predictions.

Multinomial Logistic Loss Layer

A semantic segmentation network calculates the most likely class for each pixel separately. In order to use the previously described training procedures, it is therefore necessary to combine multiple probability distributions into a single loss value. We thus aim to create a loss function for training multiple labels at the same time.

When N is the total number of pixels, and \vec{t} is an integer vector which contains the correct label for each pixel, then \hat{p}_{n,t_n} is the probability that pixel n is assigned the correct label t_n . Summing up the individual logistic loss values and normalizing over the number of elements yields the formula for the **multinomial logistic loss** (see Jia et al., 2014):

$$E = -\frac{1}{N} \sum_{n=0}^{N-1} \log(\hat{p}_{n,t_n}) \quad (2.17)$$

Inclusion over Union

The multinomial logistic loss value is useful for guiding the training of the network. In order to evaluate a network's performance, other measures are however frequently used. The inclusion over union (IoU) value provides an accessible assessment of segmentation quality (see Everingham et al., 2015, p. 104ff). To that end, the segmentation results are first evaluated separately for each class. The formula for calculating the class-specific IoU-value reflects on the fact that both the number of correct classifications and the number of false positives influence the perceived performance:

$$\text{IoU}_c = \frac{\sum_n^N (t_n = c) \wedge (p_n = c)}{\sum_n^N (t_n = c) \vee (p_n = c)} \quad (2.18)$$

The quality of the segmentation for a given class c thus equals the number of correctly classified pixels out of the total $N = W \times H$ samples (where the predicted label p_n equals the ground truth label t_n) divided by the number of samples where either the ground truth label or the predicted pixel label equals c . Therefore, this formula punishes both false positives and false negatives and yields a result which is independent of the number of evaluated pixels per class. This approach enables us to compare the segmentation performance for unevenly distributed labels, where

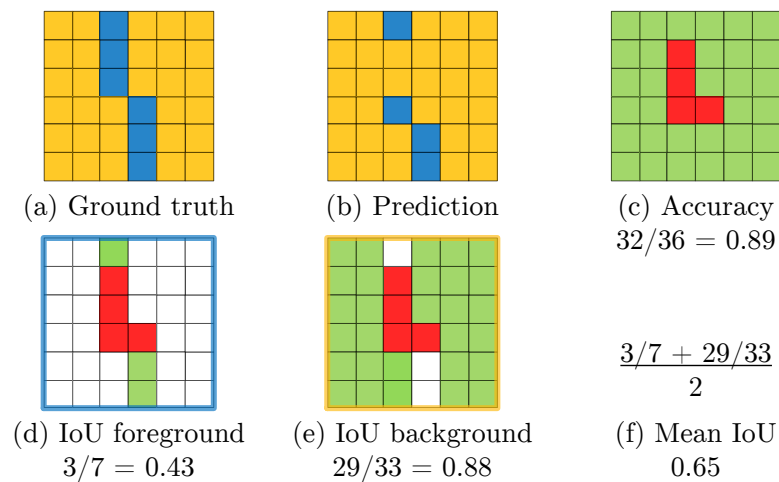


Figure 2.8: IoU for uneven class distributions. The top row uses the ground truth data (a) and the network’s predictions (b) to calculate an accuracy value (c). The bottom row visualizes the calculation of a mean IoU value (d-f) for the same data set.

the number of correctly classified pixels would be vastly misleading: When we consider a segmentation problem for 2 different classes as visualized in figure 2.8, the high accuracy with respect to the large number of background pixels overshadows the performance relating to the fewer foreground pixels. In real world scenarios however, we are especially interested in the detection of said foreground objects. Thus, the overall accuracy value is not representative of the network’s perceived segmentation quality and the mean IoU value is a much better way to evaluate the segmentation performance.

Segmentation Performance

Long et al. (2014) also analyzed the performance of several segmentation networks based on the classification architectures described in section 2.1.4 using the data provided by the PASCAL VOC 2011 segmentation challenge. They reported that the VGG-16 network with an output stride of 32 pixel (designated as FCN-32s) was able to achieve state of the art segmentation results at a mean IoU value of 59.4% despite of a simplified training procedure which used a fixed learning rate. By reducing the output stride of the VGG-16 network to 8 pixel (FCN-8s), it was furthermore possible to increase the segmentation accuracy to a mean IoU value of 62.7%, which clearly outperformed all previous VOC 2011 entries. Surprisingly, the modified GoogleNet was not able to match these values even though both networks achieved similar classification results. (Long et al., 2014, p. 5)

2.1.6 Conditional Random Fields for Refinement

A FCN computes a semantic segmentation without taking the spatial relationships between pixels into account. The prediction for each pixel is computed solely based on the respective receptive field and thus independently from all other pixels. It is however often possible to increase the reliability of the prediction by considering the classes of other, nearby pixels.

A single pixel which is entirely surrounded by a lot of other pixels with a high probability for a given class is very likely a part of the same object, thus a member of the same class regardless of any local features which may lead to a different prediction. In addition, pixels which share many features are also often members of the same class, even when there is a significant distance between them.

These two observations can be exploited in order to refine the prediction of the FCN. To this end, a conditional random field (CRF), that is a field of random variables where each variable may take on any of the predefined class identifiers, can be created. In order to find a solution to the segmentation problem, we furthermore define the energy of this CRF as the sum of several independent error potentials: Specifically, this energy should increase with any deviation from the initial prediction as well as with any violation of the aforementioned observations. The iterative minimization of the field's energy thus results in a **global** optimization of the semantic segmentation. With the additional introduction of weight factors for each of the potentials, we can furthermore control the exact composition of the total energy and thus the relative importance of each rule. (see Krähenbühl and Koltun, 2012)

A Fully Connected CRF Model

Krähenbühl and Koltun (2012) have presented an efficient model for a fully connected CRF, that is a CRF where each random variable is connected not only to its immediate neighbors but also to all other random variables spanning the entire image. They define the energy $E(\mathbf{x})$ for their CRF \mathbf{x} as the sum of unary and pairwise error potentials:

$$E(\mathbf{x}) = \underbrace{\sum_i \psi_u(x_i)}_{\text{unary potentials}} + \underbrace{\sum_{i < j} \psi_p(x_i, x_j)}_{\text{pairwise potentials}} \quad (2.19)$$

The unary potentials are based on a label compatibility function ψ_u which computes an error energy term out of the prediction of the FCN and the current assignment of each random variable. For T different predefined classes, this compatibility function can thus easily be implemented as $T \times T$ matrix where the predicted class determines the row index and the assigned class is used as column index. In order to favor the initially predicted class over all other options, the matrix is initialized with negative coefficients on the main diagonal. Furthermore, this simple implementation also enables us to optimize all coefficients of the compatibility matrix during a separate training state.

Likewise, each of the pairwise potentials ψ_p in equation 2.19 is partly the result of a label compatibility function μ between the two random variables. In addition, this function is also multiplied with the sum of weighted Gaussian kernels defined in an arbitrary feature space \mathbf{f} :

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \underbrace{\sum_m^K w^{(m)} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j)}_{k(\mathbf{f}_i, \mathbf{f}_j)} \quad (2.20)$$

The CRF model by Krähenbühl and Koltun (2012) uses two such kernels to model the observations from section 2.1.6:

$$k(\mathbf{f}_i, \mathbf{f}_j) = \underbrace{w^{(1)} \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\alpha^2} - \frac{|I_i - I_j|^2}{2\theta_\beta^2}\right)}_{\text{appearance kernel}} + \underbrace{w^{(2)} \exp\left(-\frac{|p_i - p_j|^2}{2\theta_\gamma^2}\right)}_{\text{smoothness kernel}} \quad (2.21)$$

Thereby, p designates the position of a pixel whereas I denotes the pixel's color vector. θ_α , θ_β and θ_γ are parameters which control the shape of the Gaussian kernels and thus the relative influence of similarity and proximity. In consequence, the appearance kernel models the fact that nearby pixels with similar features are more likely to be of the same class while the smoothness kernel removes small isolated regions. Similar to the unary potentials, all parameters of the binary potentials can be optimized during a separate training stage.

Krähenbühl and Koltun (2012) also implemented a mean field approximation to the CRF distribution, which normally requires an iterative message passing algorithm with a quadratic complexity to propagate the assignments of all random variables. They were however able to demonstrate that the message passing step for their specific model could also be implemented by Gaussian filters in feature space, which results in a linear complexity. This key observation enables us to efficiently use conditional random fields for the refinement of semantic segmentations.

Implementation as Recurrent Neural Network

The previously described CRF refinement procedure already significantly improves segmentation results. Zheng et al. (2015) have however demonstrated that an additional performance boost can be achieved by fully integrating the separate post-processing step into the neural network. To this end, they have extended the FCN-8s architecture with additional CRF refinement layers which implement Krähenbühl and Koltun's method. Thereby the class inference is performed by evaluating several layers iteratively, thus forming a recurrent neural network (RNN). Using this approach it is possible to perform an end-to-end training which optimizes both the weights of the FCN and the parameters of the CRF refinement stage at the same time. This strategy has been proven to be highly successful with a mean IoU value of 75.0% on the PASCAL VOC 2011 data set, which marks the current state of the art for this semantic segmentation benchmark. (Zheng et al., 2015)

2.2 Training and Validation Data

In order to evaluate and compare the previously described segmentation and refinement methods and to assess their appropriateness for the creation of virtual driving environments, it is necessary to use a validation data set which is representative of the actual problem. Existing, large annotated image collections (e.g. ImageNET, COCO, PASCAL VOC) are however hardly suitable for this purpose.

First of all, the photographs in these collections have been captured at different angles, focal lengths and resolutions. In accordance with the constraints set forth in section 1.4, the input to the segmentation stage however only consists of orthographic aerial images. Such images have already been preprocessed to correct for viewpoint changes, yielding an almost constant resolution and a similar perspective for all images. As a result, a classification network which operates on these preprocessed images may not require as many layers to reliably detect objects.

More importantly however, the available, annotated data sets do not contain suitable label classes for the reconstruction of the virtual environment. The recreation of a landscape requires the detection of surfaces such as fields, grass and boulders which normally are classified as background areas. In addition, some scenes may contain rare and particular objects such as tire walls, curbs and aprons, which are not part of any reference data collection.

For these reasons, the generation of a custom ground truth data set, specifically designed for evaluating the semantic segmentation quality of orthographic aerial images, is an integral part of this thesis.

2.2.1 A Projection for Aerial Images

The Web Map Tile Service (WMTS) standard - as developed by the Open Geospatial Consortium - is a broadly adopted specification for providing a map consisting of orthographic aerial images to the public. This standard defines both a tile-based coordinate system for different map layers and zoom levels as well as an interface for accessing the data²³⁴.

In order to create a map using the WMTS standard, it is necessary to convert geodetic coordinates, which describe the location of a point on the ellipsoidal surface of the earth, to Cartesian coordinates defined on a rectangular grid. The EPSG3857 projection - also known as pseudo-spherical Mercator projection - is widely used for this purpose:

$$\begin{aligned} x &= r \cdot \phi \\ y &= r \cdot \ln(\arctan(\pi/4 + \lambda/2)) \end{aligned} \tag{2.22}$$

Here, r is the equatorial radius of the earth. Throughout this thesis, ϕ is used to denote the longitude whereas λ symbolizes the latitude of a geodetic coordinate pair.

² *OpenGIS Web Map Tile Service Implementation Standard* (Masó et al., 2010).

³ *OGC Web Map Tile Service (WMTS) Simple Profile* (Masó, 2014).

⁴ *OWS-6 DSS Engineering Report - SOAP/XML and REST in WMTS* (Pomakis, 2009).

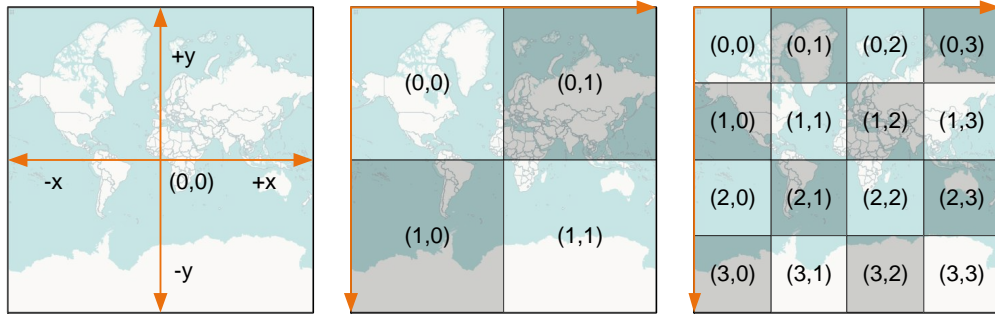


Figure 2.9: WMTS coordinate system. From left to right: EPSG3857 projection, constrained at 85.0511° North/South; WMTS zoom level 1 (2x2 tiles); WMTS zoom level 2 (4x4 tiles).

It has to be noted that the EPSG3857 projection given in equation 2.22 results in significant position and scale errors and is thus not an official, recognized geodetic system. Due to its simplicity, it is nonetheless a foundation of many popular web-based map services⁵ (also see Klokan Technologies; Petr Pridal, 2015).

When the acceptable range of λ is restricted, it is furthermore possible to effortlessly create an uniform grid with the origin at the center of four equally sized quadrants. In this case, both the width and the height of the projected region equal the circumference of the earth:

$$\begin{aligned}
 x_{max} &= y_{max} = \pm (r \cdot \pi) \\
 y_{max} &= \pm (r \cdot \ln(\tan(\pi/4 + \lambda_{max}/2))) \\
 \lambda_{max} &= \pm (-\pi/4 + 2 \cdot \arctan(e^\pi)) = \pm 85.0511^\circ
 \end{aligned}
 \tag{2.23}$$

While this limitation obviously prevents the mapping of the polar regions, it also facilitates the creation of a scheme for partitioning the Cartesian EPSG3857 space into equally sized, quadratic bitmap tiles. To that end, the WMTS standard introduces the concept of zoom levels as shown in figure 2.9.

The projected region is thereby divided into a grid of 2^n by 2^n tiles, where n represents the zoom level. The origin of this grid is however located in the top left corner, that is it is different from the origin of the EPSG3857 projection. In addition, the direction of the y-axis has been inverted in order to resemble the coordinate system commonly used by pixel images. As a result, the location of a tile in the world coordinate system - specified by row and column index - can be treated in the same way as the location of a single pixel within a tile, thus enabling efficient bitmap operations spanning multiple tiles.

⁵ WGS 84 / Pseudo-Mercator - Spherical Mercator, Google Maps, OpenStreetMap, Bing, ArcGIS, ESRI - EPSG:3857 (Klokan Technologies; Petr Pridal, Tomas Pohanka and Radim Kacer, 2015).

2.2.2 Image Resolution

In order to calculate the resolution of a single pixel in an aerial image distributed over WMTS, it is necessary to specify both the zoom level and the pixel dimensions of a tile. At zoom level 19, the earth's equator - with a radius of 6378.137 km and a circumference of 40075.017 km - spans 2^{19} or 524288 tiles. With an assumed tile width of 256 pixel, this yields a horizontal resolution of approximately 30 cm/pixel. As a result of the chosen projection, this resolution however also increases significantly depending on the latitude of a given point.

In the context of this work, all segmentation algorithms are trained and evaluated using WMTS map tiles provided by `basemap.at`⁶. This source supplies tiles covering the whole of Austria which are stitched together in order to create large, seamless input images. These images are captured at a latitude of approximately 47°N and their actual resolution thus equals roughly 20 cm/pixel.

2.2.3 Surface and Object Classification

As we aim to create a virtual driving environment based on real landscapes, it is necessary to create a vectorized map from these aerial images. Such a map is built upon abstract representations of concrete objects, where each object is a member of a predefined object class. While digital maps contain hundreds of different object classes⁷, it is extremely difficult to distinguish all of these solely based on aerial images. Thus, the number of recognized, different map features has to be limited.

Kluckner et al. (2010) classify pixels into five different groups: buildings, streets, grass, trees and water. While it is possible to reliably assign pixels to one of these clearly delimited classes, the resulting map is not detailed enough for the reconstruction of a landscape. We propose to shift the balance slightly towards a more detailed - but also sometimes ambiguous - representation by creating a segmentation for a larger total of 38 different pixel classes. To this end, we define 14 classes for different terrain and vegetation categories, 6 classes for buildings, 10 classes for transport routes and finally 8 classes specifically designed for the segmentation of race tracks. A full list of all 38 pixel classes is given in appendix section A.1.

2.2.4 Ground Truth Data Sets

Based on these pixel classes, a set of 16 aerial images (WMTS zoom level 17, 1024×1024 pixels) has been manually annotated. Figure 2.10 presents a result of this work. Although each of the carefully chosen images is representative of multiple pixel classes, some classes are only contained within a single bitmap. It is thus not feasible to randomly assign the annotated images to training and validation data sets, as this would result in very different distributions of the pixel classes.

⁶basemap.at (Stadt Wien und Österreichische Länder bzw. Ämter der Landesregierung, 2015)

⁷Haklay and Weber, 2008. <http://www.openstreetmap.org>.

As a solution to this problem, each larger aerial image is therefore first divided into 4×4 tiles (WMTS zoom level 19, 256×256 pixels). These smaller tiles are then partitioned into 4 groups according to the schematic also given in figure 2.10. This process is repeated for all of the original annotated images and yields a total of 256 different, annotated ground truth tiles, where each tile is part of one of four groups.

After an evaluation of the pixel class distribution in the four groups, the tiles in group 2 have been selected as validation data set whereas the tiles in the remaining three groups are used as training data. Thus, 75% of the tiles are available for training while 25% of the tiles are exclusively held back for the validation of the algorithms. A detailed analysis of the pixel class distribution over both data sets is given in appendix section A.3.

To increase the variability and the overall amount of available ground truth data, each of the tiles in both groups is furthermore mirrored and rotated in 90° steps. Initial experiments have shown that this measure significantly improves the learning abilities of all evaluated algorithms. As a result, a final number of 1536 training and 512 validation tiles is used for all experiments.

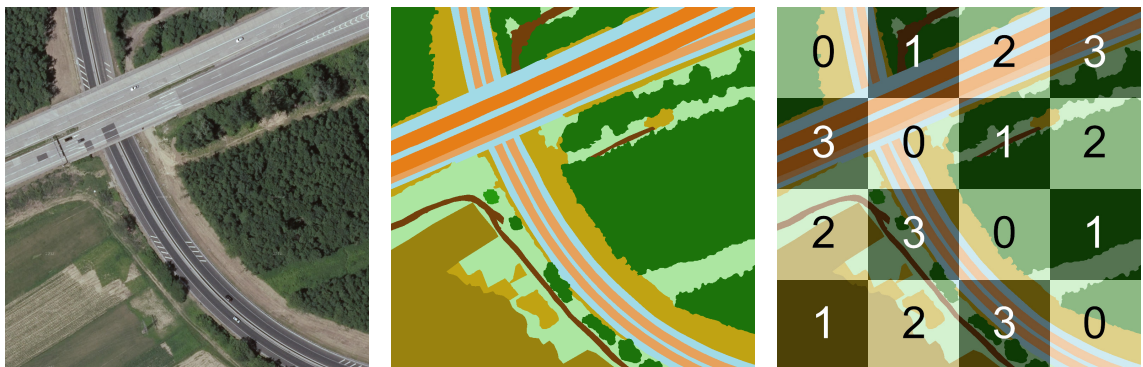


Figure 2.10: Examples of ground truth data (I). Freeway intersection, Styria, Austria. From left to right: Original aerial image^a, manually created segmentation, partitioning into training and validation data sets. See appendix A.2 for further examples.

^aDatenquelle / Source of images: basemap.at; License: CC-BY 3.0 AT.

2.3 Segmentation Experiment

2.3.1 Deep Learning with Caffe

This thesis aims to evaluate a practical application of neural networks. To this end, we utilize the excellent deep learning framework Caffe. A particular advantage of this system is that the architecture of a network is stored in a human-readable text format, thereby facilitating experiments without the need for recompilation of the program. In addition, Caffe also includes a number of different solvers amongst which is also an implementation of the SGD algorithm. Furthermore, the framework contains optimized GPU implementations for most performance critical computations which provide a considerable speedup when compared to their CPU equivalents. All these features enable us to concentrate on the definition of the network architecture and the actual training procedure rather than spending resources on implementation details. (also see Jia et al., 2014)

Most importantly however, Caffe is already very widely used for classification tasks. As a result, many specialized neural network layers are readily available, and there is a significant amount of academic work which uses them. This framework is thus not only very popular in the industry, it also drives many state of the art approaches for image classification and segmentation.

2.3.2 Network Architectures

Since Long et al. have proven that VGG-16 based systems achieve excellent results in the PASCAL VOC semantic segmentation tests, these networks have been the architecture of choice for many segmentation tasks. In order to adapt such a network to a custom problem with a different list of predefined classes, it is however necessary to modify the top layers and train them on the appropriate ground truth data set. This change to the final layers could also affect the relative performance of different network architectures. For this reason, we analyze the semantic segmentation performance of both VGG-16 and GoogLeNet based networks with respect to the task at hand.

For each architecture, a Caffe reference implementation⁸ of the original classification network is used as the starting point. At first, the top layers are replaced according to section 2.1.5 and the weights are transplanted with the help of IPython⁹ scripts in order to form a FCN. Moreover, input padding and deconvolution layers are added as necessary to create networks which produce a separate prediction for each input pixel. In addition, the number of output channels and thus the number of predicted classes is changed in order to match the custom ground truth data set.

These modifications yield networks with an output stride of 32 pixel before the up-sampling layers, which are thus hereinafter referenced as GoogLeNet-32s and VGG16-32s. The predictions produced by said networks are however somewhat blurry and

⁸see the Caffe model zoo: <https://github.com/BVLC/caffe/wiki/Model-Zoo>

⁹“IPython: a System for Interactive Scientific Computing” (Pérez and Granger, 2007).

lack the necessary details for the reconstruction of the roads and environment. By fusing the features detected by the penultimate convolution layer with these predictions as described by Long et al., finer details can however be added and the output stride is reduced to 16 pixel. The resulting networks are therefore known as GoogLeNet-16s and VGG16-16s. This procedure can be repeated once more and a full listing (in Caffe style) of the final networks with a stride of 8 pixel is available in appendix B.

Our networks differ from the examples provided by Long et al. as we decided to **eliminate zero-padding** for the input layers. This change becomes possible due to the fact that we aim to segment a continuous, spherical surface. Therefore, any padding can be replaced by an increase of the input dimensions, thus using pixels from neighboring image tiles instead of padding the image borders with zeroes. Due to limited computational resources we were not able to compare the best possible results for both approaches. Initial tests over 10.000 training iterations however have proven that the networks without zero-padding learn significantly faster while the computational requirements increase only marginally. Furthermore, this technique enables us to seamlessly stitch together the segmentation results of neighboring tiles later on.

2.3.3 Supervised Training

The training of both evaluated network architectures has been organized in several consecutive stages as suggested by Long et al. To this end, we first adapt the classification networks to the new ground truth data and train them with an output stride of 32 pixel. Once the results do not improve any further, we add the previously mentioned additional layers which reduce the stride to 16 pixel and perform an end-to-end training on these new networks. Finally, we start out from the fully initialized 16s-networks, once again add the necessary fusion layers and train the resulting 8s-networks end-to-end.

There is however one important difference: While Long et al. mainly focused on the detection of a few large structures spanning entire images, we have to detect many comparatively small objects covering only a few pixels. This is a direct result of the scale of the areal images, which is close to 20 cm / pixel. Thus, the cross section of a secondary road with a width of 4 meters is only 20 pixels wide. As a result, the computational effort is significantly shifted from the training of the 32s-networks to the fine-tuning of the 8s-networks.

All training steps are performed with a fixed learning rate of $1e^{-10}$, with a high momentum of 0.99 and a small weight decay factor of 0.0005. These extreme values are necessary as we decided to solve the network using the SGD algorithm without gradient accumulation, once again following the recommendations of Long et al. Each of the 1536 different ground truth examples therefore should have only a minimal influence on the network weights in order to gradually approach an optimal solution.

Figure 2.11 shows the progress of the 6 different FCN segmentation networks during the training stage. The performance of each network is evaluated on the validation

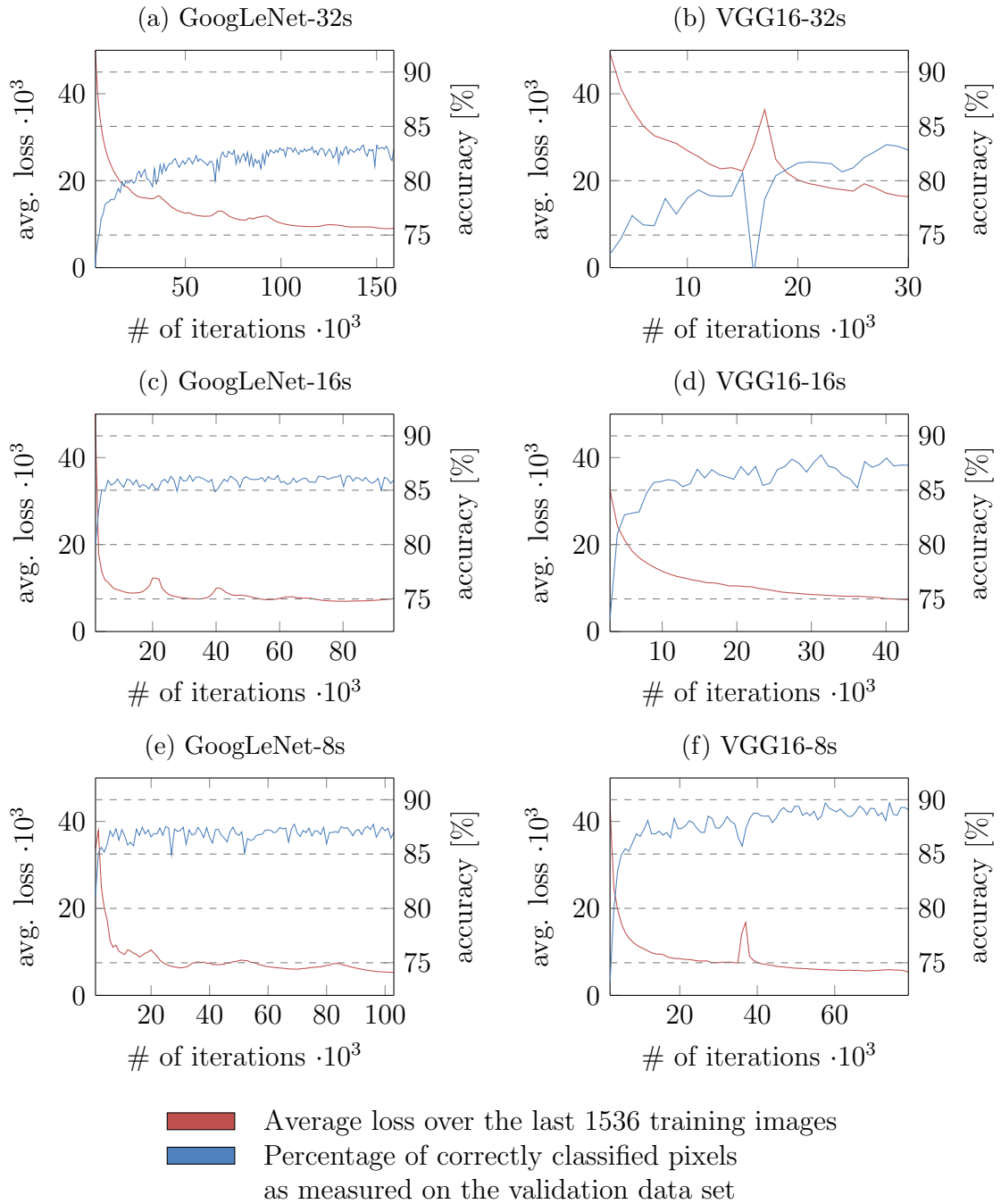


Figure 2.11: Training of 6 different fully convolutional segmentation networks.

data set after each 1000 iterations. Even though the loss value may be still decreasing with each iteration, the training has to be stopped once the validation results do not improve any further in order to prevent over-fitting. Table 2.1 shows the iteration of the final, best-performing snapshot for each network and the corresponding per-pixel segmentation accuracy.

Network	Snapshot Iteration	Time/Image [s]		Accuracy [%]
		Training	Validation	
GoogLeNet-32s	100000	0.476	0.169	83.0182
GoogLeNet-16s	50000	0.335	0.157	86.0416
GoogLeNet-8s	69000	0.319	0.156	87.3965
VGG16-32s	30000	1.483	0.411	82.8169
VGG16-16s	30000	1.484	0.412	87.5606
VGG16-8s	50000	1.483	0.410	89.4082

Table 2.1: Segmentation speed and accuracy of different neural networks.

Furthermore, table 2.1 also lists the average duration for the semantic segmentation of a single image tile with 256×256 pixels. All timings have been obtained using a customized Caffe executable on a Dell M6800 notebook with a Core i7 4800-MQ CPU and a NVIDIA Quadro K3100M GPU (768 CUDA cores @705MHz, compute capability 3.0, 4GB DDR5 RAM). While this setup is far from a high-end configuration, it still provides results which enable us to compare the relative computational efficiency of the analyzed networks.

The reverse calculation of the weight update deltas using the back-propagation algorithm is only required during the training stage which results in significantly shorter validation times, as is to be expected. In addition, the GoogLeNet derivatives consistently require less computational resources than the VGG-16 based networks. This behavior is easily explained by the fact that GoogLeNet has been designed with efficiency in mind. As a result, GoogLeNet-8s only requires approximately 5% of the weight parameters of the VGG-16-8s network.

This enhanced throughput however also comes at a cost: While the accuracy of the GoogLeNet-32s is able to match the performance of the corresponding VGG16-32s network, the reduced number of weight parameters in the convolutional layers makes it difficult to reliably detect finer structures as the output stride of the networks is gradually decreased. At a stride of 8 pixel, this difference of the relative segmentation accuracy is already very pronounced and amounts to more than 2 percentage points.

2.3.4 Discussion of Results

As has been explained in section 2.1.5, the per-pixel accuracy is however only a first estimate for the true quality of the segmentation performance. Therefore, we also present a full evaluation of the segmentation results of GoogLeNet and VGG16

by providing additional visual examples, comparing the respective IoU values and analyzing the confusion matrices.

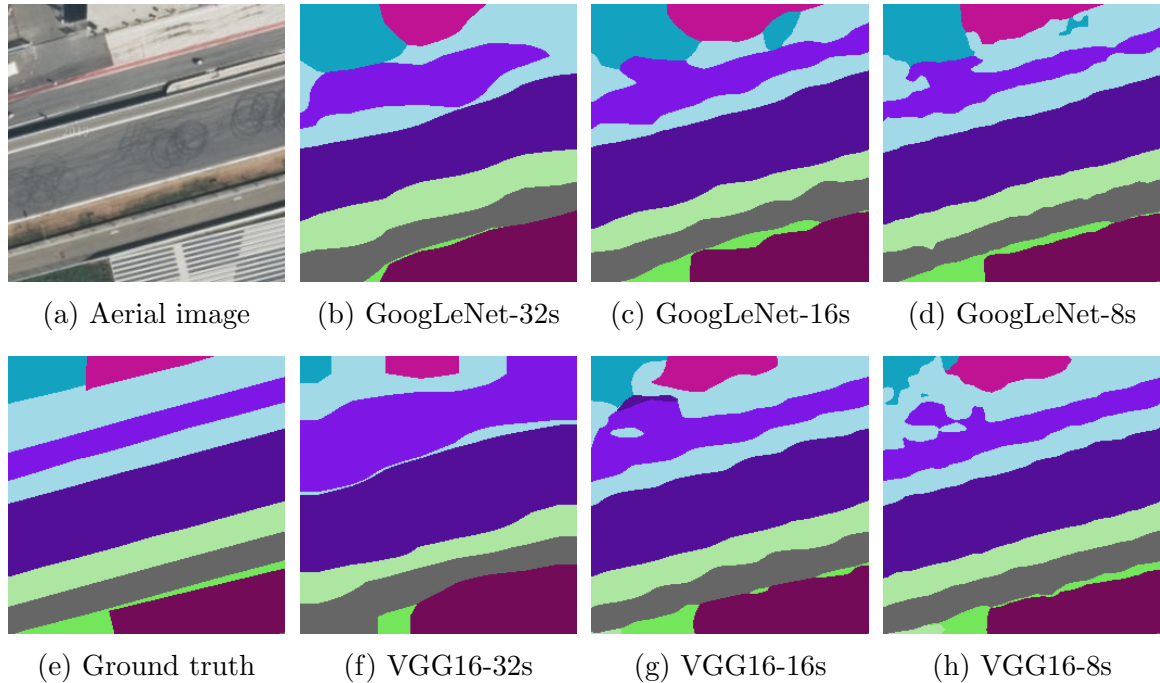


Figure 2.12: Examples of segmentation results for networks with different stride.

Figure 2.12 visualizes the effects of a decreased output stride. We deliberately use an image tile with many narrow sections and sharp, straight edges in order to clearly work out the benefits of using features from deeper layers to increase the output resolution. Initially, both GoogLeNet-32s and VGG16-32s produce blurry, blob-like regions. The detection of narrow stripes and sharp edges and thus the quality of the segmentation is however improved considerably by appending additional fusion layers to both networks. This visual confirmation also proves the results of table 2.1 and the detailed analysis in the next sections will therefore only cover the best performing networks, that is GoogLeNet-8s and VGG16-8s.

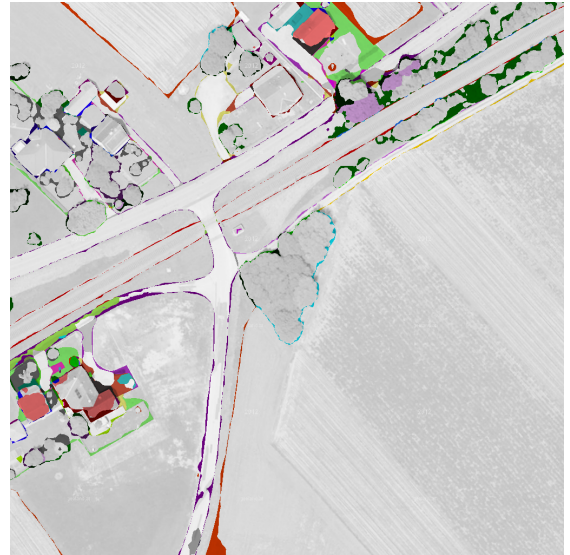
Pseudocolor Images

Through the use of pseudocolor images, the segmentation errors for a larger 1024×1024 pixel region are visually compared in figure 2.13. The original aerial image is first converted to a greyscale representation and a color is then imbued on all regions which are not correctly classified. Although the performance of GoogleNet-8s and VGG16-8s at first appears to be quite similar, there are some notable differences with respect to both training and validation tiles.

Tiles which were used during training are segmented with excellent accuracy by both networks. There are however some narrow, colored stripes at the borders of all regions which denote that these pixels were wrongly classified. These miss-classifications could be a result of the manually generated ground truth data. Even though extreme



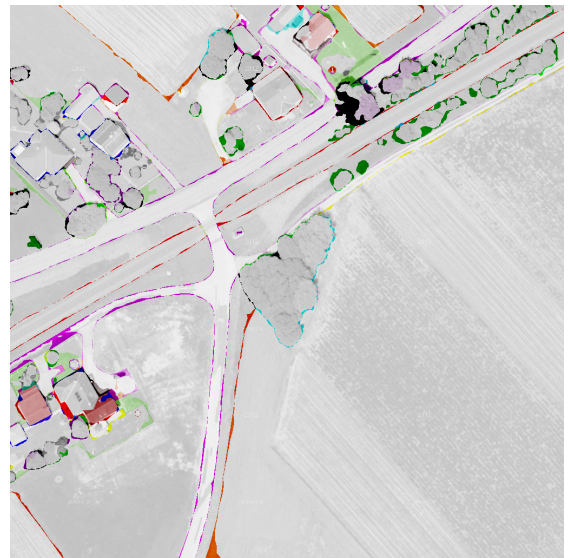
(a) Aerial image



(b) GoogLeNet-8s error



(c) Ground truth



(d) VGG16-8s error

Figure 2.13: Examples of segmentation errors. Darker regions in the ground truth image have also been used during training, whereas lighter regions are exclusively used for the evaluation of the network.

care has been taken in creating the data set, the exact border between neighboring regions is sometimes disputable - especially when we think of borders between objects such as macadam roads and the surrounding gravel and grass patches. Nevertheless, the colored stripes appear to be slightly wider for GoogLeNet-8s, thereby indicating an inferior edge detection performance.

More interestingly however, there are pronounced differences between the segmentation results for the 4 evaluation tiles. Specifically, the GoogLeNet-8s network apparently often succeeds to detect a region's border but then fails to assign the correct label. While this observation is also sometimes true for VGG16-8s, the latter network nonetheless achieves better overall results.

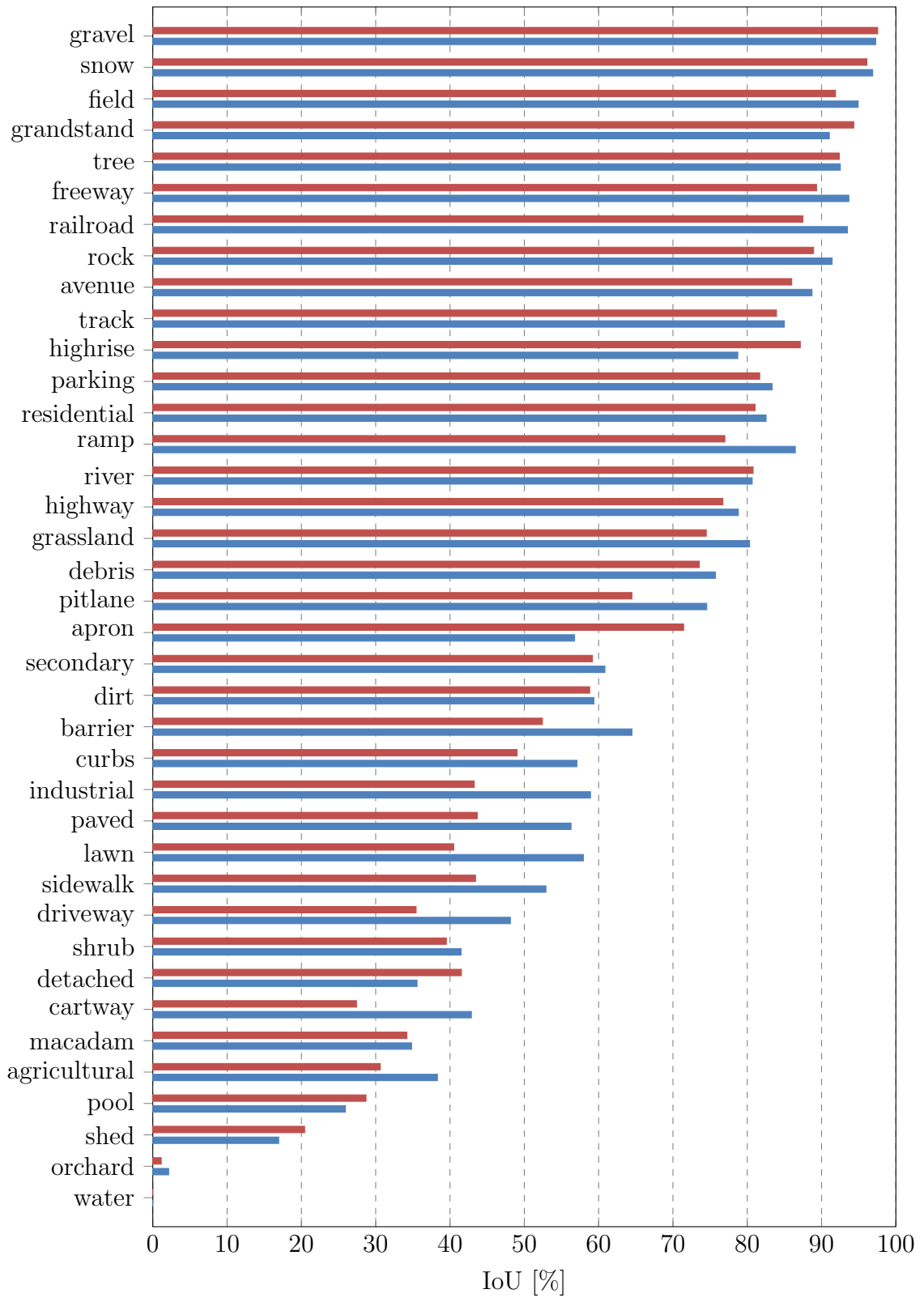
Inclusion over Union

In order to further analyze the cause for the wrongly assigned labels which affect the segmentation quality, we first calculate the class specific and mean IoU values for the evaluation data set as described in section 2.1.5. A comprehensive list of the results for all 38 classes is available in appendix C. GoogLeNet-8s achieves a mean IoU value of 61.20% and is clearly outperformed by VGG16-8s, which obtains a score of 64.66%. These overall results are in line with the per-pixel accuracy ratings, but a closer look on the class-specific performance reveals some peculiarities as visualized in figure 2.14.

Notably, both networks produce excellent results with IoU values of more than 75.00% for approximately half of the classes. We can therefore conclude that these classes have strong, unique and unambiguous features and are clearly marked in both training and evaluation data sets which makes it easy to detect them. The 'gravel' class is a perfect example, as it is used exclusively for the run-off areas of racetracks, always has a light yellow color with a homogenous texture and usually exhibits pronounced borders.

The next group - which makes up for about a quarter of the classes - is responsible for a large share of the performance difference between the two networks. This block contains classes which can be detected by adequately trained neural networks, as is demonstrated by the VGG16-8s system. Although a fraction of the training data may still be ambiguous, we thus gather that the provided examples and the distance between feature vectors are in principle sufficient to arrive at a correct judgment. Despite of this, GoogLeNet-8s obtains far lower IoU values and the class-specific difference amounts to more than 17 percentage points as exemplified by the results for 'lawn' and 'industrial' buildings. **We therefore conclude that the measured overall performance gap between the networks is truly representative for the segmentation quality and that the VGG16 architecture indeed outperforms GoogLeNet for our specific application.**

At the low end of the spectrum, approximately 25% of the classes are notoriously difficult to detect and both networks fail to achieve IoU values of more than 45%. A prime example of this is the 'water' class, which has been used during training but unfortunately is not part of the evaluation data set (see appendix A.3 for a full list

Figure 2.14: Segmentation performance of ■ GoogLeNet-8s and ■ VGG16-8s.

of the class distributions). Thus, the IoU value for this class drops to 0.00% as soon as a network incorrectly labels a single pixel of the evaluation data as water. The low rating for the 'pool' class on the other hand is very likely caused by insufficient training examples, as there are many different pool types and the evaluation set therefore contains pools with previously unknown feature vectors.

Upon closer inspection, there is however another possible reason for some poor results: The bottom half in figure 2.14 also contains quite similar pairs of classes such as cartways and macadam roads, sheds and detached houses or shrubs and orchards. We therefore suspect that both networks are confused by ambiguous feature vectors or - as it is also often difficult to manually pick the correct class from the aforementioned pairs - by incorrect ground truth data.

Confusion Matrices

Figure 2.15 presents the confusion matrix for GoogLeNet-8s. The network's performance is a result of both the percentage of correctly labeled ground-truth pixels (as shown on the main diagonal) and the number of wrong classifications (visualized by all other fields). Thus, each row adds up to 100 percent. The color scale has been chosen to exaggerate any differences from the expected results so that a 10% share of false positives is already clearly visible as bright orange field.

As has been suggested by the IoU values, there are some pairs of classes with high confusion levels. Most notably, 20.29% of sheds are classified as detached houses and likewise, 21.60% of detached houses are labeled as sheds. In addition, a significant fraction of industrial and agricultural buildings are also denominated as sheds. We can therefore conclude that these classes share many features and that it is difficult for GoogLeNet to distinguish them using just aerial images.

Furthermore, there is a second large group of miss-classifications, which also has been visible in the pseudocolor images: Many border regions of larger patches are incorrectly labeled as grassland, dirt or even paved surfaces. This indicates failures to precisely separate the objects such as roads from the surrounding terrain. This phenomenon is at least partially the result of the output stride of 8 pixel, which blurs the exact shape of objects. However, some of these errors may be unavoidable, as the manually created ground truth data set also includes flawed boundaries.

Finally, figure 2.16 shows the confusion matrix for the best-performing network VGG16-8s which features slightly lower overall error levels. Specifically, this network is less likely to confuse objects with grassland, dirt or paved surfaces than GoogLeNet-8s and therefore more often succeeds in detecting the correct patch boundaries. There is however still room for improvement in this area, and the correct labeling of different but similar building types remains an unsolved problem.

For these reasons, we will use the fully trained VGG16-8s network as foundation for all further experiments, as we aim to advance the segmentation performance by adding additional refinement steps.

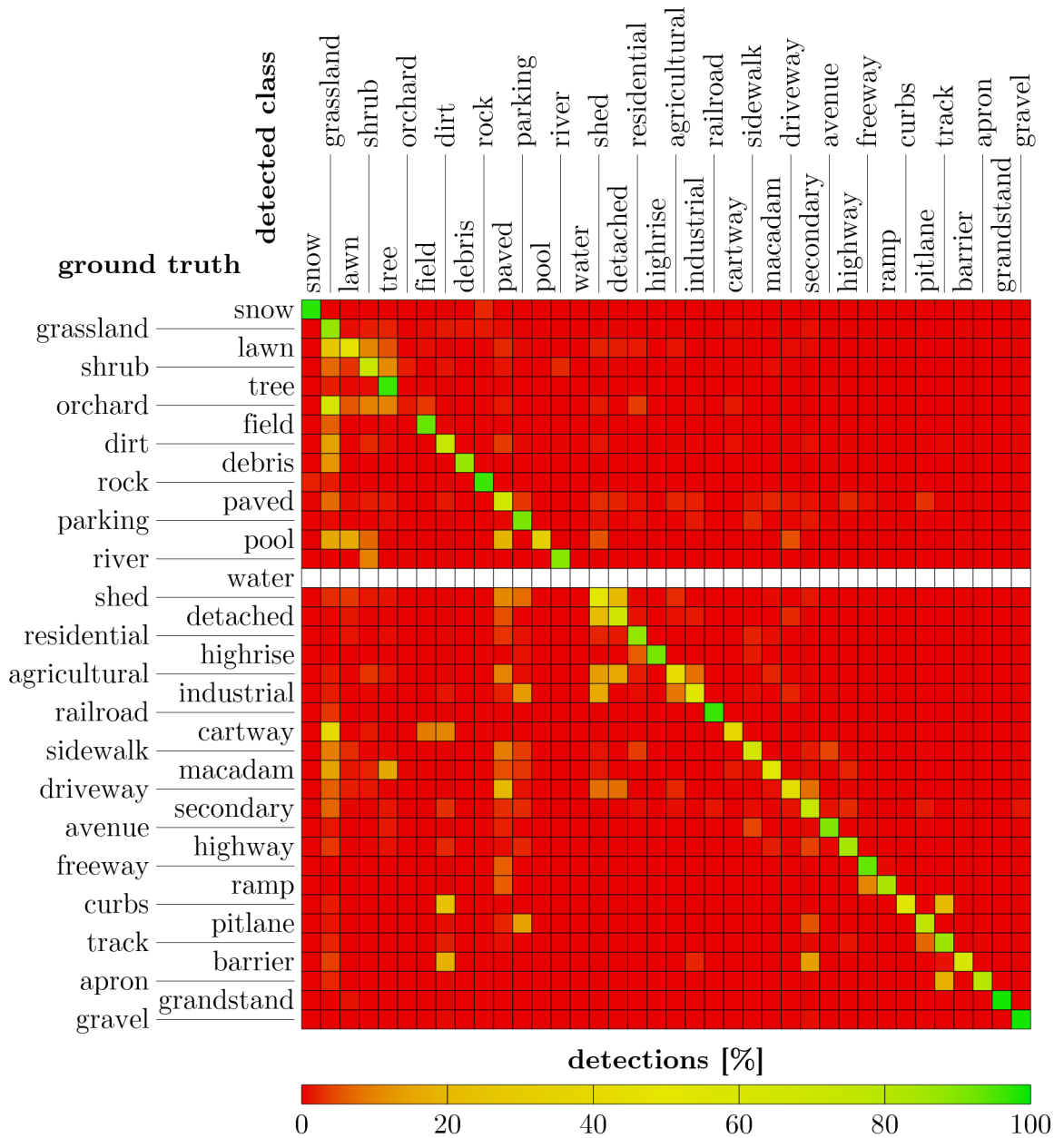


Figure 2.15: Confusion matrix for GoogLeNet-8s network.

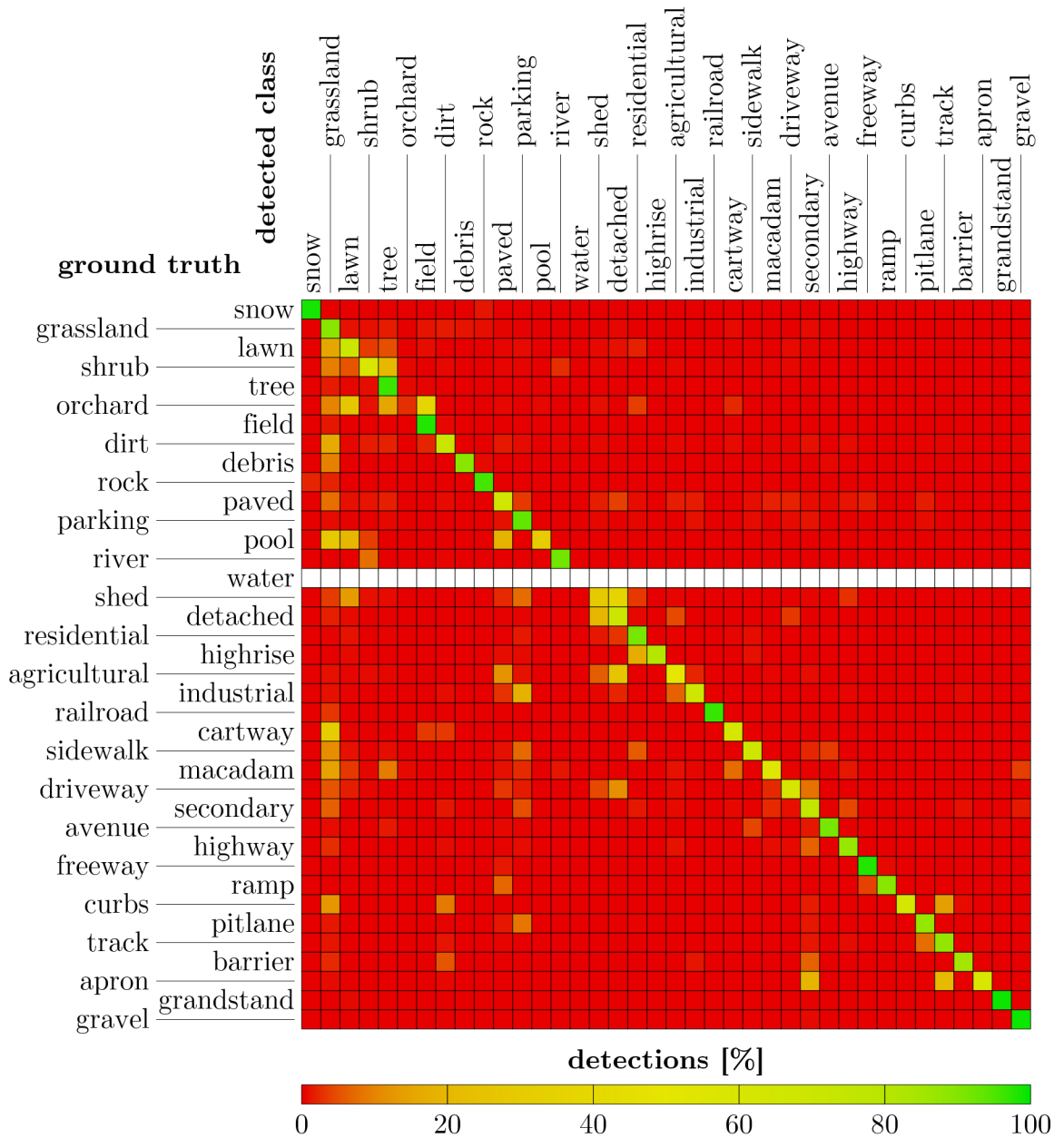


Figure 2.16: Confusion matrix for VGG16-8s network.

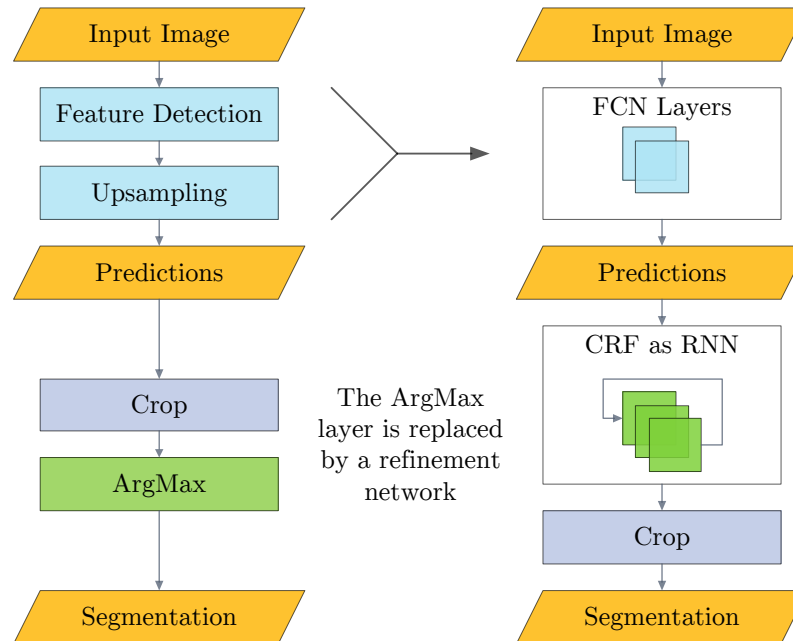


Figure 2.17: Adding additional CRF layers to an existing segmentation network.

2.4 CRF Refinement

A CRF is a state of the art method for refining a semantic segmentation. Summarizing section 2.1.6, we aim to exploit the fact that nearby pixels are of the same class if and only if they share many features. This observation enables us to improve the segmentation results at disputed object boundaries. In addition, we find that some classes are more likely to occur next to each other, which allows us to eliminate certain ambiguities. For these reasons, a CRF refinement is ideally suited to address both previously recognized deficiencies of the VGG16-8s semantic segmentation network.

Figure 2.17 visualizes the process of extending a regular FCN with additional refinement layers. This approach has the advantage that the optimization of the CRF kernel parameters can be conducted in the same way as the optimization of all other network parameters. Thus, we can also use the Caffe configuration and ground truth data described previously in order to perform an end-to-end training of the modified VGG16-8s+CRF network.

Our experiment is therefore based on the implementation of a CRF as RNN by Zheng et al. Departing slightly from their work, we choose to apply the crop layer only after the labels have been inferred. This modification enables us to obtain better overall segmentation results as the cropped regions are also calculated using valid input pixels due to the elimination of zero-padding (see section 2.3.2) and thus contribute meaningful information for the inference process.

2.4.1 Estimation of Mean-Field Parameters

While some parameters of the CRF are optimized during the training, others always keep their initial value. Most notably, the normalization parameters θ_α and θ_β in equation 2.21, which govern the shape of the appearance kernel, fall under the latter category. Furthermore, the θ_γ parameter, which is used to change the extend of the smoothness kernel, is also kept constant. It is thus very important to provide good estimates for these fixed parameters. In addition, we also aim to initialize the relative weights $w^{(1)}$ for the appearance kernel and $w^{(2)}$ for the smoothness kernel with suitable values before starting the actual training of the neural network.

Following the process described by Krähenbühl and Koltun, we use their standalone DenseCRF tool to optimize the values for the fixed parameters iteratively. To this end, we extract the class predictions of the VGG16-8s network for a larger, 1024×1024 pixel tile by dropping the final ArgMax layer. While this piece of information is not sufficient to actually learn a full label compatibility function, the predictions can still be used as input to a CRF with varying parameter values. The refinement performance is then assessed using the per-pixel accuracy value of the segmentation after several mean-field iterations. In order to optimize the CRF parameters, we first disable the smoothness kernel by setting $w^{(2)} = 0$. The values for $w^{(1)}$, θ_α and θ_β are then found using a grid search algorithm. Finally, we estimate the ideal parameters for the smoothness kernel.

Parameter Set	$w^{(1)}$	θ_α	θ_β	$w^{(2)}$	θ_γ	Acc. [%]
Krähenbühl and Koltun (2012) ¹	-	61.00	11.00	1.00	1.00	-
Krähenbühl and Koltun (2012) ²	10.0	80.00	13.00	3.00	3.00	88.73
Zheng et al. (2015, p. 4ff) ²	5.00	160.00	3.00	3.00	3.00	90.51
Our Estimation	5.20	13.50	11.00	1.50	1.75	90.51

¹ values taken from the original paper

² as extracted from the source code examples provided by the authors

Table 2.2: Initial weights and CRF kernel parameters vs. segmentation accuracy.

Table 2.2 lists the results of our parameter estimation procedure as well as other values for the fixed parameters, which have been successfully used for the refinement of different data sets. The accuracy for each configuration has been measured using our full validation set, after 10 mean-field iterations and without any additional training.

Remarkably, two of these very different parameter sets obtain the exact same overall improvement of segmentation accuracy, albeit their performance for individual images out of our 512 piece validation collection varies widely. The third set however causes a deterioration of results, that is the accuracy is actually worse than for a segmentation without additional refinement. We therefore conclude that it is necessary to hit an application specific sweet spot for the kernel weight $w^{(1)}$ in order to perform a successful refinement. This observation is also backed by the results of our grid search for the appearance kernel’s parameters.

In addition, we argue that the value for parameter θ_α , which controls the effect of long range connections, can be chosen from a wide range without significantly affecting the segmentation accuracy. This finding contradicts previous work by Krähenbühl and Koltun and Zheng et al. However, the scale and content of the segmented orthographic aerial images is rather different when compared to their examples. While larger structures such as industrial buildings may also span hundreds of pixels, many other objects in our case have a diameter of less than 15 pixels. Thus, our grid search returns an optimum for θ_α of only 13.5 pixels, which is only a fraction of the value used in other experiments.

In conclusion, the per-pixel accuracy of the semantic segmentation can be improved from 89.41% to 90.51% without any further training by simply applying the results of a straightforward CRF parameter estimation process.

2.4.2 Supervised Training

The separate optimization of the parameter matrix for the label compatibility function and the fine-tuning of the kernel weight matrices are the next logical steps to further improve the segmentation performance. As suggested by Zheng et al., we attempt to learn these parameters using the regular Caffe SGD solver and our complete training data set. To this end, the learning rate and weight decay factors for all layer parameters of the original VGG16-8s network have been set to 0. A constant overall learning rate, which is fixed at $1e^{-9}$, thus only affects the mean-field parameters. The weight decay has furthermore also been disabled for the kernel weights $w^{(1)}$ and $w^{(2)}$, as a normalization of these parameters quickly causes a deterioration of segmentation results.

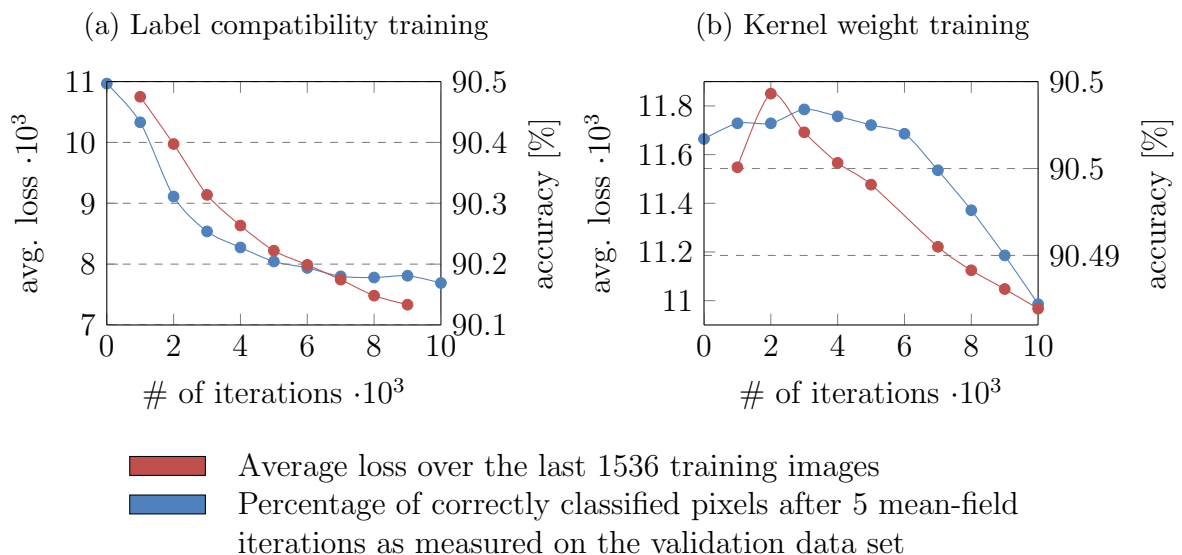


Figure 2.18: Training of CRF label compatibility function and kernel weights. Weight updates have been disabled for all other layers.

Figures 2.18a and 2.18b visualize the training progress for two independent experiments, each starting out from the original VGG16-8s network with an appended refinement layer and CRF kernel parameters as estimated previously. The network clearly adapts to the training data in both cases, but unfortunately not only the loss value but also the accuracy on the validation data set quickly decreases. Many other attempts to improve the segmentation performance of VGG16-8s+CRF by an end-to-end training procedure with various different learning rates and layer parameters have unfortunately also failed.

Despite of this, we conclude that a well-trained CRF would be ideally suited to boost the segmentation performance based on the rapidly decreasing loss values. The custom ground truth data collection however does not include enough training examples which prevents us from fully exploring the possibilities. Therefore, we can only analyze the refinement results of a CRF network without further training, which nonetheless scores significantly higher than VGG16-8s in most categories.

Network	Mean-field Iterations		Time/Image [s]		Accuracy [%]
	Training	Validation	Training	Validation	
VGG16-8s	-	-	1.483	0.410	89.4082
VGG16-8s+CRF	5	5	2.943	1.215	90.4967
VGG16-8s+CRF	5	10	2.943	2.169	90.5078

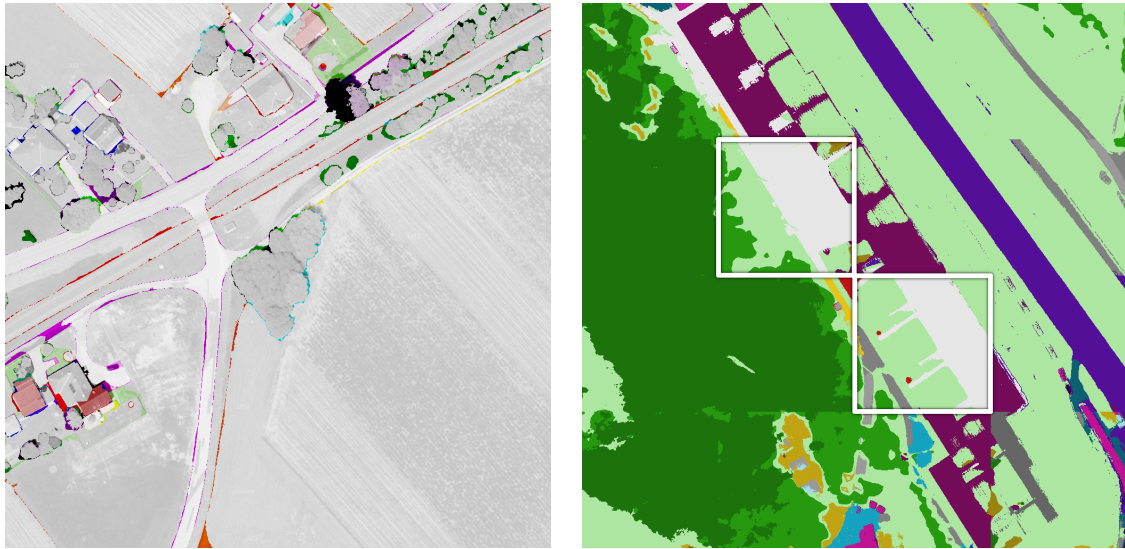
Table 2.3: Segmentation performance of neural networks. Number of mean-field iterations versus segmentation speed and accuracy.

2.4.3 Discussion of Results

Zheng et al. have suggested to train the RNN over 5 iterations while the actual segmentation should be performed over 10 iterations. According to their experiments, this setup prevents vanishing gradients during the training stage while also improving the final results slightly (Zheng et al., 2015, p. 9). Table 2.3 therefore also presents a comparison of the segmentation speed and accuracy for 5 and 10 mean-field iterations. Unfortunately, the RNN implementation has not been optimized for the GPU, which is clearly visible from the validation time per image. The available data nonetheless reveals that the refinement duration increases almost linearly with the number of mean-field iterations. Nonetheless, we will use a RNN with 10 iterations for all further evaluations due to the slightly better segmentation accuracy.

Pseudocolor Images and Stitching Errors

As can be seen in figure 2.19a, the CRF succeeds in optimizing the objects boundaries even though it has not been trained properly. The colored border regions which visualize classification errors are significantly smaller due to the additional refinement layer.



(a) Segmentation errors

(b) Stitching errors

Figure 2.19: Examples of segmentation results.

This improvement however also come at a cost for our application. Although we aim to segment a large area, the segmentation has to be executed separately for each of the many small WMTS image tiles due to limited computational resources. Later on, these segmented tiles are stitched together in order to form a continuous surface. The original FCN networks use a single, moving filter of fixed dimensions in order to label the input data and thus create a segmentation which is steady across tile boundaries.

The label inference algorithm on the other hand may completely change the annotation of a large region based on all other predictions which are part of the same image tile. Depending on the input data and CRF parameters, this can lead to noticeable segmentation borders at the tile boundaries. The problem is exaggerated by the use of more aggressive CRF parameters, that is increased values for the appearance kernel parameters $w^{(1)}$ and θ_α . Figure 2.19b visualizes this effect for an extreme parameter set. Additional post-processing steps are however always necessary to mitigate the issue.

Inclusion over Union

As a result of the refinement, the mean IoU value for the segmentation changes from 64.66% to 67.71% (see appendix C for a complete list of all scores). This major advancement is - somewhat unexpectedly - caused by the CRF smoothness kernel, which succeeds in eliminating the few, isolated and wrongly labeled instances of the water class from the results.

Thus, it is no longer necessary to consider the corresponding, class-specific IoU value for the calculation of the mean IoU score, which therefore improves dramatically. The perceived difference in segmentation quality is however not even close to 3

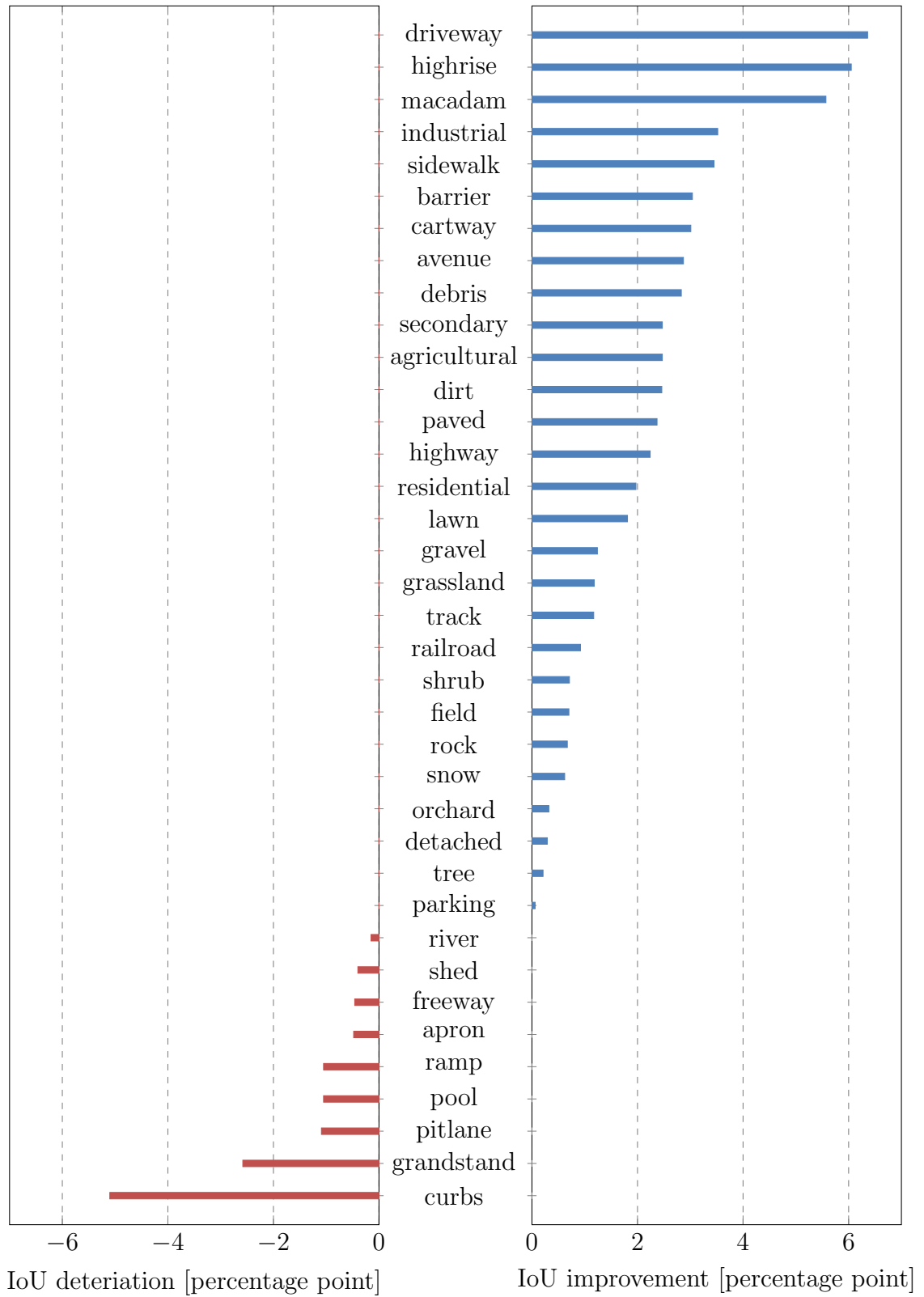


Figure 2.20: IoU change with CRF layer after 10 mean-field iterations.

percentage points, as the original VGG16-8s network would already have achieved a mean IoU value of 66.41% if we were to ignore the water category in the first place.

Figure 2.20 therefore also indicates the differences between all other original, class-specific IoU values and the results of the CRF inference. Although there is a considerable improvement in most categories, some numbers actually deteriorate as a direct result of the refinement process. Upon closer inspection we conclude that this mainly affects classes which regularly exhibit heterogeneous textures. Curbs and open grandstands are prime examples, as a single instance of these classes is usually made up from many small, adjacent patches with different colors. The CRF detects this dissimilarity between neighboring pixels and therefore wrongly infers that the region in question must contain multiple objects of different classes.

Confusion Matrix

Finally, figure 2.21 presents the confusion matrix for the network with integrated CRF refinement as calculated over the validation data set. The improved edge detection performance is visible as reduced confusion between objects and their surroundings. In addition, there are some minor advances regarding ambiguous object classes when compared to figure 2.16. This final confusion matrix for a CRF enhanced network however also indicates that the initial attempt to unambiguously detect our 38 predefined classes has been over-ambitious given the current state of the art. In particular, there still remain some bright spots at the intersections of different building categories as well as for other similar object types.

Supplementary experiments have therefore shown that a separate post-processing step, which reduces the number of different vegetation and building classes, dramatically improves the overall performance values. Dropping just 6 classes by merging them with other, similar categories boosts the per-pixel accuracy from 90.50% to 92.16% while the mean IoU value at the same time increases from 67.59% to 73.60%. We expect that these numbers could be even higher if we were to fully repeat the complete training procedure with the goal of labeling only the remaining 32 classes.

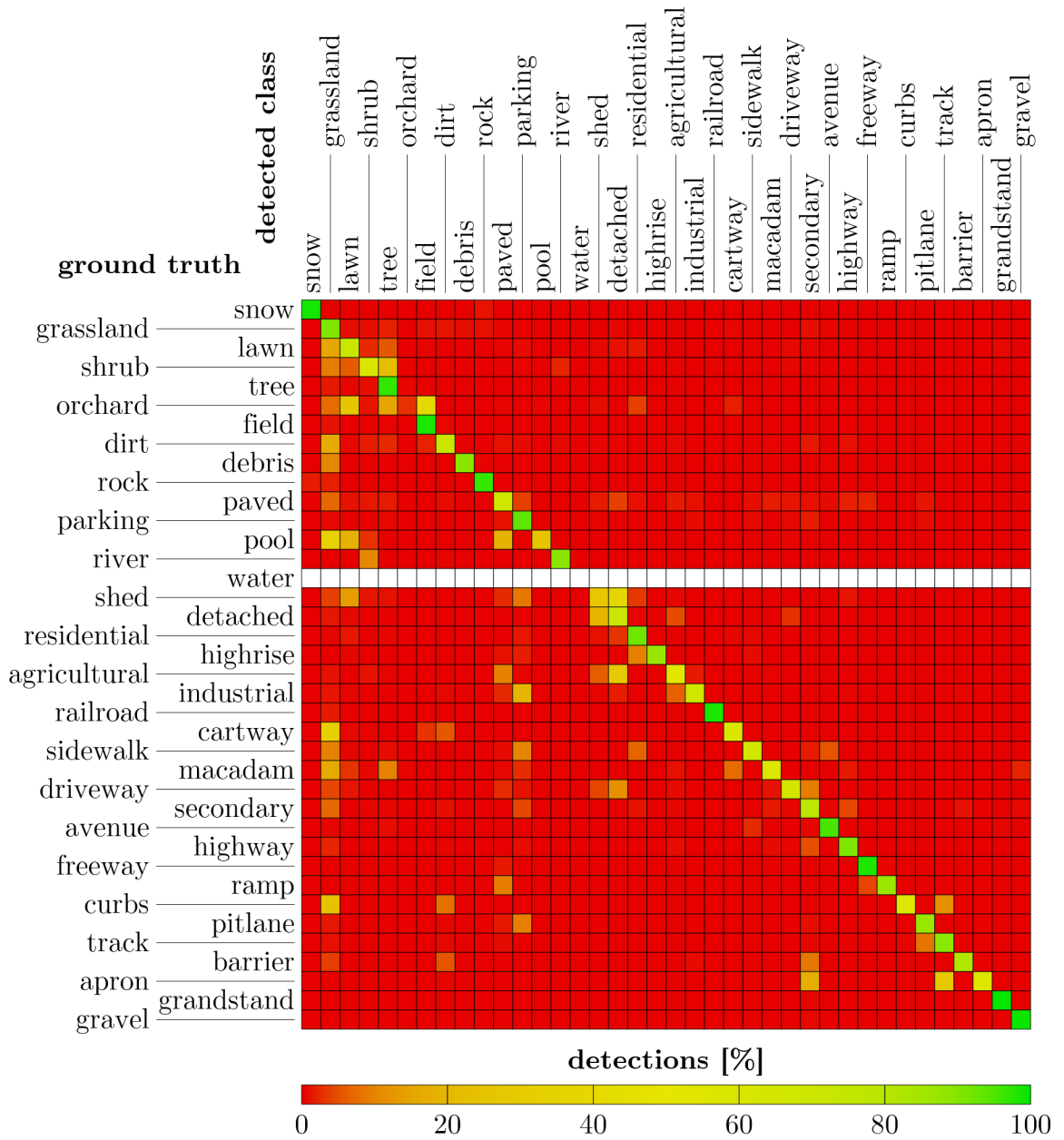


Figure 2.21: Confusion matrix for VGG16-8s-CRF network with 10 mean-field iterations.

2.5 Summary

The realized experiments prove that it is possible to achieve state of the art semantic segmentation results for orthographic aerial images by training a VGG16-based neural network using the Caffe deep learning framework. A fully integrated CRF refinement layer, which is implemented as RNN, can furthermore be employed to improve the segmentation of boundary regions considerably.

Unfortunately however, our small, purpose-built ground truth data collection is not sufficient for automatically optimizing all CRF parameters. We argue that this deficiency is at least partly caused by the fact that the collection of aerial images has been annotated using a comparatively high number of 38 different, pre-defined classes. Our research retrospectively indicates that even with an additional refinement stage, some ambiguities between those classes can not be fully resolved.

However, an exact discrimination of different building and vegetation classes is not even necessary for our application as long as the road information is extracted faithfully. The run-time performance of our algorithms and the quality of the segmentation results are furthermore and by all means adequate for commercial applications on end user systems. Thus, the VGG16-8s+CRF segmentation network described in this chapter is more than capable of providing the input data for the next steps in our workflow and thereby is a viable tool for transforming aerial images into virtual roads.

Chapter 3

Road Map Refinement

The computation of a road network map based on segmented aerial images is an important part within the proposed workflow for creating virtual roads. To this end, the OpenDRIVE[®] specification, which details a file format for virtual driving environments, recommends to define each road segment based on annotations alongside a single reference line (Dupuis, Marius, 2015, p. 28). Although we will be using a simpler description format in order to stay compatible with various real-time platforms, we still aim to create an annotated, graph-based road map in a similar way.

Miao et al. (2013) suggested a pipeline for the extraction of road center lines and intersection information from aerial images. According to their method, a binary semantic segmentation is performed at first and features are extracted from each proposed road segment. This information is then used to fill any holes in the original segmentation based on similarities of the road texture. Finally, multivariate adaptive regression splines (MARS) are utilized to extract a smooth description of each road segment from the segmentation. According to the authors, their approach works well for images with a resolution of approximately 1 meter per pixel (Miao et al., 2013, p. 584). While this method would indeed perfectly harmonize with our already established process, it unfortunately fails to provide an estimate for the road width.

There are however also some other complications which prevent us from directly using the previously generated segmentation results in the first place. Most importantly, a road network may contain segments which are invisible from our airborne point of view. These occlusions can be caused by underground roads or overpasses as well as by nearby tall trees and buildings. A network map built solely upon aerial images would thus almost inadvertently contain interrupted roads and disconnected road fragments. In addition, similar defects would also be caused by any small patch of wrongly classified pixels. Even a failure to correctly label a few pixels in the road shoulder region could have severe consequences, as an unsteady road boundary always causes an unsteady vehicle trajectory when the driver aims to stay in the middle of the lane. Thus, such a road segment would be hardly enjoyable for human or artificial drivers alike.

For these reasons, it is necessary to treat the task at hand as an optimization problem. Our overall objective therefore is to create a consistent road network con-

taining realistic, smooth trajectories based on the previously generated segmentation results. To this end, we however also have to rely on additional data sources which enable us to determine the initial condition for the optimization and to solve defects caused by occlusions. This auxiliary data can be obtained from publicly accessible GPS traces or mapping services. Although the spacial resolution and accuracy of this data is inadequate for the creation of virtual driving environments, it is still an excellent foundation for the optimization process. Thus, the original assignment of creating a map from segmented aerial images can be transferred into a problem of refining an existing map while annotating all road segments with additional shape information and using the semantic segmentation as a secondary condition.

In this chapter we will therefore analyze two different methods for accomplishing this task based on OpenStreetMap¹ data. A state of the art approach by Mattyus et al. uses a combination of different strategies including superpixel classification and edge detection in order to solve the problem by means of inference in a Markov Random Field (MRF) (Mattyus et al., 2015). We will evaluate the applicability of this method using our own test data set and juxtapose the results with those obtained from a straightforward, nonlinear optimization experiment. In addition, we will use the same optimization framework to extend the refined road map to three dimensions by exploiting a DTM.

3.1 Background and Methodology

3.1.1 Map Projections

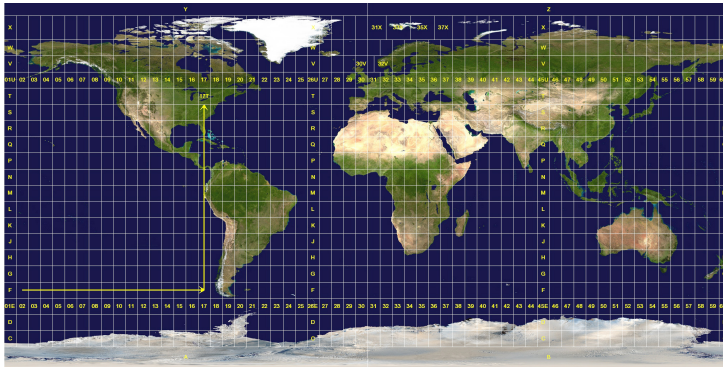
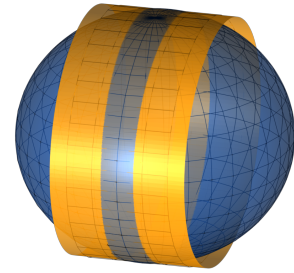
In order to execute the experiments, it is necessary to define a common frame of reference for the original map data and the segmented aerial images. Auxiliary data sources such as terrain models also have to be converted to this same coordinate system. Moreover, the subsequent registration of inertial measurements will enable us to validate our simulation results by comparing them to actual on-road measurements.

The WMTS map projection described in section 2.2.1 is however hardly suitable for these tasks, as the scale of the map varies depending on the geodetic latitude λ . Starting out at $\lambda = 45^\circ\text{N}$ and moving only 10 km further towards the north pole, the scale difference between those two points already amounts to 0.15%. Thus, using this projection, it is not possible to compare the trajectories of real and simulated vehicles in large scale environments with the necessary accuracy.

Consequently, we aim to find a different map projection for converting geodetic road map data into an uniform, rectangular grid which exhibits a constant scale alongside its reference meridian. This projection also has to be conformal, that is the relative local angles about every point on the map have to be shown correctly so that any shape information is preserved (see Snyder, 1987, p. 4).

The most widely accepted implementation of such a projection is the Universal Transverse Mercator (UTM) system as depicted in figure 3.1, which is used for large-

¹<http://www.openstreetmap.org> (Haklay and Weber, 2008)

(a) UTM zones^a (also see Snyder, 1987, p. 62)

(b) UTM map projection

^aThis image is in the public domain in the United States because it is a work prepared by an officer or employee of the United States Government as part of that person's official duties under the terms of Title 17, Chapter 1, Section 105 of the US Code.

Figure 3.1: Universal Transverse Mercator (UTM) system.

scale maps of the entire world (Snyder, 1987, p. 57). To this end, the surface of the earth is divided into 120 regular zones, where each zone covers a 6° wide band of either the northern or southern hemisphere. This approach requires 60 different, elliptic projection cylinders to cover the whole of the earth, and a single cylinder for a single band is visualized in figure 3.1b.

The scale of the center meridian of the underlying transverse Mercator projection is thereby fixed at 0.9996 and thus the projection cylinder intersects the surface of the earth twice. This method mitigates the longitudinal scale variations within the zone, as the projection error is now more equally spread across the entire zone at the expense of a slightly inaccurate scale at the center meridian (Snyder, 1987, p. 57ff).

In order to calculate the projected Cartesian coordinates, it is thus necessary to map geodetic coordinates onto an elliptic cylinder of arbitrary scale. This task is however not trivial, and it is therefore usually solved by a set of series approximations converging at the desired solution (Snyder, 1987, p. 61). The coefficients of the series vary based on the reference ellipsoid which is used to describe the earth. An extensive list of pre-calculated coefficients for many different reference ellipsoids, accurate down to 10^{-6} meters for points within 1000 km of the center meridian of a zone, is however publicly available (see Office of Geomatics, 2014). This level of accuracy is more than sufficient for all practical purposes, and the examples presented in this chapter will therefore use the UTM projection in order to provide a common reference frame for all data sources.

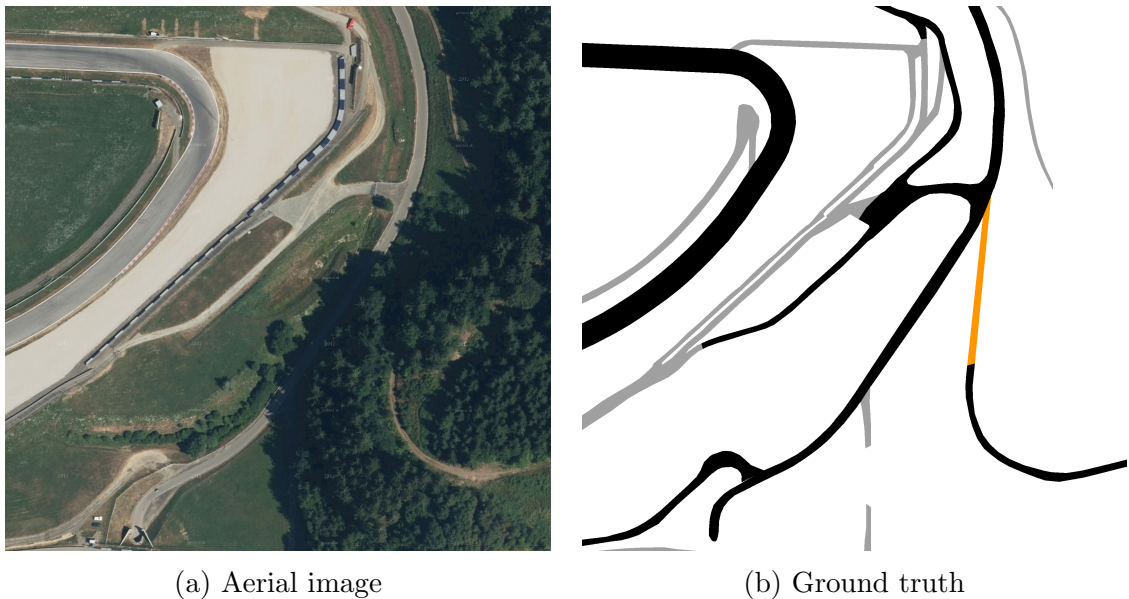


Figure 3.2: Central region of the validation data set. The OSM database does not include matching entries for light gray paths which are therefore excluded from evaluation. Orange paths are not visible in the aerial image as a result of natural occlusion, but they should be evaluated nonetheless.

3.1.2 Validation Data

The evaluation of the two road map refinement approaches will be conducted using a specialized ground truth data set. To this end, the original orthographic aerial images have been slightly enlarged to a scale of 13 cm / pixel - which is exactly the scale Mattyus et al. used and thus enables us to immediately apply their method without further modifications. Out of this overall data set, a 4096×4096 pixel region (0.28 km^2) has been manually labeled and will be used for validation. Figure 3.2 visualizes the central region of this validation data set.

The validation region has been carefully chosen so that it contains many diverse race track sections, secondary and primary roads and crossings in close proximity, which will enable us to predict an algorithms performance with respect to the application at hand. In order to achieve this goal using a rectangular region, it was however necessary to accept a tiny overlap with the original data set used for CNN training. Thus, 3.5% percent of the data points have already been used as examples for the segmentation network and will therefore be excluded from the evaluation in order to avoid a bias towards our method.

Furthermore, the OSM database does not contain valid seed data for some minor roads and footpaths. These paths will also be excluded from the analysis as demonstrated in figure 3.2b. With the help of additional GPS traces, an optimization of said paths would however become possible using either of the proposed methods.

3.2 Refinement by Inference

Mattyus et al. have proposed an efficient method for enhancing freely available road maps by using a contextual model. Their algorithm returns only topographically correct roads and also provides estimations for the width of each road segment (see Mattyus et al., 2015). In this section, we will therefore present a compact overview of this method, discuss possible shortcomings and enhancements and finally evaluate the refinement results on our custom validation data set.

3.2.1 Markov Random Fields

The authors suggest a road map refinement algorithm which is based on inference in a Markov Random Field (MRF). Prince (2012) defines a MRF as a graphical model which ties together individual observations to form a solution that makes global sense. Thereby, a MRF is composed of

1. a set of sites which often correspond to physical or image locations
2. a set of random variables associated with each site
3. a set of neighbors (other sites) which are connected to each site

In addition, random variables which are not directly connected to each other have to be conditionally independent in order to form a valid MRF (also see Prince, 2012, p. 218). Cost functions - known as potentials - are then used to describe the **joint probability** of a specific distribution of random variable states for a connected subset of sites. Consequently, the overall energy of a MRF can be defined as sum of these potentials. The process of inference in a MRF thus computes a distribution of states which minimizes the field's energy². (Prince, 2012, pp. 227-283)

In order to solve the road map refinement problem, Mattyus et al. present a MRF which directly uses the offset of each pixel with respect to the original OSM data and the width of each road segment as random variables (Mattyus et al., 2015, p. 1). Pairwise potentials between neighboring segments are used to ensure the smoothness of the inferred road surface. In addition, the authors also suggest to include a wide range of other, image-based potentials, which will be briefly described over the next paragraphs.

3.2.2 Image-based Features

Superpixel Segmentation and Random Forest Classification

The most significant potential for the refinement of road maps can be derived from a semantic segmentation of aerial images. In this context, a classification of image

²While this may sound similar to the description of a CRF in section 2.1.6, there is a fundamental difference: A CRF models the conditional probability $P(Y|X)$ i.e. it is conditioned on an input X and thus is only designed to handle straightforward prediction tasks. A MRF on the other hand models the joint probability $P(X, Y)$ which allows for a wider range of applications. (see Lafferty et al., 2001, p. 286)

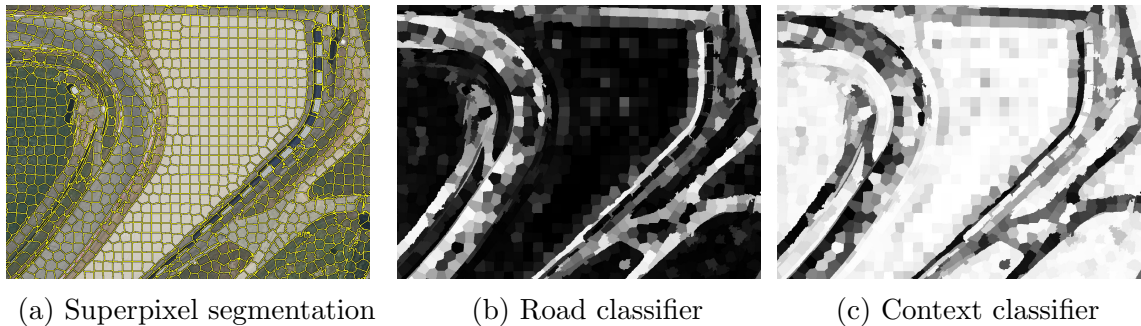


Figure 3.3: Random forest classification of superpixels. Each binary classifier predicts the probability for a given label on a scale from 0% (black) to 100% (white).

pixels by a CNN with an integrated CRF as described in chapter 2 is the state of the art method, which however is computationally expensive. Mattyus et al. suggest to create a much simpler binary segmentation by labeling pixels as either road or context with the help of a superpixel-based random forest classifier. This method is orders of magnitude faster than the CNN approach, but the quality of the resulting segmentation is also significantly worse.

Nonetheless, superpixels and random forest classifiers have been successfully used for the segmentation of aerial images. Kluckner et al. achieved an accuracy of 95.3% for the detection of road segments by using this method with an additional CRF refinement step (Kluckner et al., 2010). Mattyus et al. however do not directly apply any costly post-processing, but they instead compensate for imprecise superpixel boundaries by introducing other image-based features.

Moreover, Mattyus et al. make use of even faster simple linear iterative clustering (SLIC) superpixels, which further increases the performance difference when compared to the CNN approach. The SLIC method is an alteration of the k-means algorithm, and thereby also clusters observations so that each observation belongs to the cluster with the nearest mean value. The number of calculations is however optimized by restricting the search space for each pixel’s cluster to a region which is proportional to the desired - and parameterized - superpixel size. (Achanta et al., 2012)

Figure 3.3a visualizes the results of the SLIC algorithm on our custom validation data set. Using this data, two separate random forest classifiers are trained in order to calculate the probability for an input being either a road or a context pixel (see figures 3.3b and 3.3c). To this end, Mattyus et al. use the random tree implementation provided by OpenCV³. The final cost function is then implemented by adding up the probabilities for all non-road pixels within the proposed road segment and likewise the sum of all road pixel probabilities within a fixed context region next to the proposed road segment. (Mattyus et al., 2015, p.2)

³<http://opencv.org/> (Bradski, 2000)

Edge Detection

A superpixel segmentation without additional post-processing is however not sufficient to detect the exact boundaries of objects. For this reason, Mattyus et al. argue that the borders of the detected road segments should also align with visible edges of the aerial image. This observation is always true for primary roads with visible lane markers, but even for secondary roads there often is a clearly visible boundary between the road surface and the surrounding environment. Thus, the distance to the nearest image edge can be used as an additional feature for the road width estimation (see Mattyus et al., 2015, p. 4).

Overlapping Potentials

Furthermore, Mattyus et al. propose the inclusion of potentials in order to avoid overlapping road segments. They implement this feature as a hard constraint, and any two roads which are located close to each other and whose orientation is within 20 degrees must not overlap. (Mattyus et al., 2015, p. 5)

Additional Features

Mattyus et al. also suggest to exploit three other features in order to enhance the stability of the refinement algorithm. According to their research, the homogeneity of the road surface and appearance differences with respect to the contextual regions can be used to correct errors when the road classifier produces unreliable results. However, with a fully trained road classifier, these features result in a comparatively minor, combined IoU improvement of just over 0.5 percentage points on average (see Mattyus et al., 2015, p. 4).

In addition, the authors also propose to implement an object detector for cars in order to determine the most likely direction of the road based on the car orientation (Mattyus et al., 2015, p. 4). Such a feature however requires high resolution aerial images and could still cause incorrect results when applied to race tracks, because the orientation of a race car is not necessarily aligned with the direction of the road center line.

For these reasons, we decided to apply the refinement method by using only the classification, edge detection and overlap potentials. The relative weights of these enabled features have been retrieved from the original work by Mattyus et al. and are based on their *Bavaria* data set.

3.2.3 Evaluation of Results

Mattyus et al. obtained their results by using an image edge detector component which has been optimized for real-time performance and produces state of the art results (see Dollár and Zitnick, 2013). Unfortunately, the license terms for this specific edge detector explicitly prohibit any commercial use and any reproduction of the associated documentation, which makes it impractical to employ this method for our application.

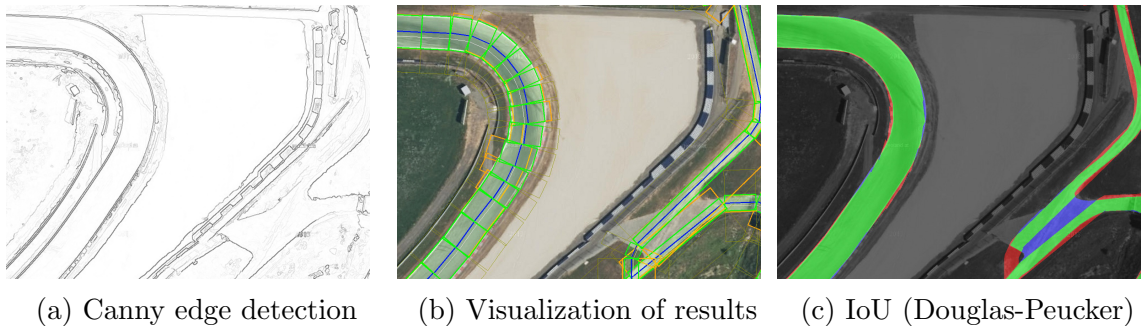


Figure 3.4: Refinement results using our Canny edge detection method.

Consequently, we created a custom edge detector based on the well-known work by Canny as implemented in the OpenCV package (see Canny, 1986). This custom edge detector applies the Canny algorithm multiple times with different thresholds, and merges the outputs to create an image as visualized in figure 3.4a. Our algorithm is significantly slower than the method proposed by Dollár and Zitnick, but the results are even better as demonstrated in table 3.1.

Edge Detector	Polygon Algorithm	
	Douglas-Peucker [%]	Smooth Roads [%]
Dollár and Zitnick	76.5161	76.8237
Canny	76.7887	77.2340

Table 3.1: IoU inference vs. ground truth on custom validation set.

In addition, we found that the calculation of the IoU value based on the refined rectangle segments as proposed by Mattyus et al. does not yield the best possible results with respect to our validation data set. The authors calculate the final road surface polygon by using the Douglas-Peucker algorithm on the detected rectangle segments. However, in our experiments, this results in an unnecessarily high number of errors as shown in figure 3.4c, which uses the color red to highlight false positives and blue to visualize false negatives.

For this reason, table 3.1 also includes the refinement results for a different method of expanding the rectangle segments into polygons. This algorithm estimates the corner points of the road surface polygons by averaging the orientation and width of neighboring rectangle segments. Thus, we are able to create smoother roads which result in fewer false positives and a better overall IoU value.

Mattyus et al. have published a maximum IoU score of 77.6% for their method. This value has been obtained on their *aerial KITTI* data set and is relative to the best achievable score with respect to their model hypothesis (Mattyus et al., 2015, p.4). In conclusion, we were able to reproduce this excellent refinement result on our custom validation data set even though it was not possible to use all of the suggested feature potentials.

3.3 Refinement by Nonlinear Optimization

Using the refinement method proposed by Mattyus et al., we find that the mean offset error of the road center lines as computed over our validation data set is just 0.847m. Therefore, we conclude that the OSM data is already an excellent prior for the real road center lines. This observation however also gives rise to the idea that state of the art results can be achieved without a global inference of all roads in the network.

Thus, we propose an alternative algorithm for the refinement of road maps, which is based on the formulation of a nonlinear optimization problem. For this reason, we briefly present an overview of the theoretical foundations of our method before analyzing the associated cost functions and the obtained refinement results in greater detail.

3.3.1 Nonlinear Least-Square Optimization Problems

Our goal is to formulate a model for the road surface which is based on parameters that can be optimized in order to closely match our observations and additional secondary constraints. The overall approach is therefore quite similar to the MRF method proposed by Mattyus et al. However, we interpret the task at hand as a prototypical example of a straightforward data fitting problem. Consequently, we also aim to find a parameter vector \vec{x} for our model which minimizes the error with respect to the measured data points by means of a nonlinear optimization algorithm.

An optimization problem is based on a set of one or more cost functions. Each cost function thereby describes one aspect of the quality of the fit with respect to a subset of the given parameter vector \vec{x} as well as some of the observations and constraints. In a typical application, there are at least as many instances of a cost function as there are data points, because each cost function instance is responsible for analyzing the fitting quality for one measured data point. A single output of a cost function instance - which is also known as residual - is thus only representative for a small part of the overall model fitting error. The total error for a given parameter vector \vec{x} is then calculated as the sum of all squared residuals. In order to optimize the model parameters, we thus have to find the minimum value for this sum of squares, hence we are dealing with a least-square optimization problem.

This task becomes a nonlinear optimization problem when a single cost function or constraint exhibits nonlinear behavior. In such a case, it is no longer possible to compute the global optimum solution, because there is no obvious connection between the residual for a cost function and the associated parameter values:

All methods for non-linear optimization are iterative: From a starting point \vec{x}_0 the method produces a series of vectors $\vec{x}_1, \vec{x}_2, \dots$ which (hopefully) converges to \vec{x} , a local minimizer for the given function. (Madsen et al., 2004, p. 6)

We are especially interested in algorithms where the calculation of the second derivative of the cost functions is not necessary. One example of this class is the

Gauss–Newton method, which is based on a linear approximation to the components of the cost function in the neighborhood of \vec{x} . Consequently, the problem can be solved by employing the Jacobian matrix, which contains the first-order partial derivatives of the cost function in this neighborhood. This procedure is rarely used directly and without alterations, but it is the foundation of some very efficient methods for the solution of nonlinear least squares problems. (Madsen et al., 2004, p. 20)

In practice, we will therefore use the **Levenberg-Marquardt** algorithm to solve our system, which extends the classical Gauss–Newton method through the introduction of an additional damping parameter. This damping parameter influences both the step size and the step direction, thereby producing either a short step in the direction of the gradient descent or - during the final iterations, when the current position is already close to the local minimum - a quadratic convergence behavior. (Madsen et al., 2004, pp. 24-29) (also see Press et al., 1992, p. 671ff)

In addition, we will make use of a **loss function** which reduces the importance of large residuals for the fit. This step is necessary to prevent a small number of outliers, that is objective errors in the measured data points, from heavily influencing the overall results. Consequently, a loss function improves the robustness of our algorithm. To this end, only the smaller residuals will be actually squared while the function should approximate a more linear behavior for large residuals. One example of such a function is the Cauchy loss as defined by Agarwal and Mierle (Agarwal and Mierle, 2016):

$$\rho(z) = \ln(1 + z) \quad (3.1)$$

For practical applications, Agarwal and Mierle however also introduce an additional scaling parameter C to set the threshold between outliers and valid measurement data, which yields the equation for the squared residuals $\hat{\rho}(r^2)$:

$$\hat{\rho}(r^2) = C^2 \rho\left(\left(\frac{r}{C}\right)^2\right) \quad (3.2)$$

As explained in chapter 2.3, we however do not aim to implement all the evaluated methods ourselves. Thus, we will use the excellent nonlinear least squares package CERES to build or model, formulate the constraints and solve the problem with the Levenberg-Marquardt algorithm. The CERES framework has one additional property which greatly facilitates this procedure: It is not necessary to explicitly implement the first order derivatives of our cost functions, because CERES supports two automatic ways of calculating the Jacobian matrices. For unsteady functions, the partial first order derivatives can be calculated numerically. For steady cost functions, CERES however also supports automatic differentiation by using the standard differentiation rules on the mathematical operations which form the cost function based on the functions actual source code (Agarwal and Mierle, 2016). By exploiting these features, we can fully concentrate on devising the cost functions which best describe our problem without thinking about possible mathematical implications.

3.3.2 Problem Formulation

Our method is partially based on the work by Mattyus et al. as it also aims to create a map of a road network by refining existing OSM data. This data is in our experience already quite close to the optimum solution and thus makes for an excellent initial condition which enables us to utilize a local optimization method in the first place. While Mattyus et al. however calculate the width and offset per road segment, our model of the road map graph is parameterized in terms of node properties. More precisely, our model estimates the offset and width at each position which joins two adjacent road segments and in a direction which is normal to the trajectory of the associated path.



Figure 3.5: Road graph refinement. OSM data is shown as white polyline, the optimized road surface area is enclosed by smoothly connected orange quads.

This design requires slightly more complicated calculations, but it also inherently ensures topographically correct, smooth road surfaces: As we are optimizing only node parameters, the properties at the end point of one segment and the start point of the next segment are calculated simultaneously and adjacent segments thus always maintain a smooth connection. As a result, the offset and width may however be different at the start and end of a single road segment and the segments consequently have the shape of arbitrary quads. Figure 3.5 visualizes this effect by showing the state of the model at the end of the nonlinear optimization process.

Classification Errors

Similar to Mattyus et al., we also use a semantic segmentation of aerial images to define our primary cost function for the detection of road segments. There are however some fundamental differences between the two approaches: Our method directly uses the output of the CNN+CRF network described in section 2.4 to compute the

residuals. This network already infers the exact boundaries of segmented objects by using a fully integrated CRF, and thus it is not necessary to add a separate post processing step or an additional edge detector component. Furthermore, the CRF annotates each pixel with the most likely class out of the 38 different pre-defined labels while the information about the original relative probabilities of the classes is ultimately discarded.

Despite of these differences, our cost function for the computation of classification errors is still heavily influenced by the work of Mattyus et al. Most notably, we also use one instance of the cost function per road segment and we analyze both the segment itself and the contextual region to compute residuals. To this end, our cost function is based on two adjacent nodes and thus receives the offset and width at each node in the form of four independent parameters. This enables us to calculate the resulting arbitrary road segment quad which spans the distance between the two nodes.

Subsequently, we define four different regions based on this geometry: Two trapezoids are located immediately outside of the left and right border of the original quad. These areas both have a fixed width of 16 pixels (1.25 meters) and form the contextual region for the road segment. In addition, two other trapezoids are located just inside of the quad’s right and left border. In our implementation, these areas also have a fixed width of 16 pixels and mark the outer regions of the actual road segment. Figure 3.5 visualizes contextual areas in blue and boundary regions in red.

Label	False Pos. [%]	False Neg. [%]
cartway, road, racetrack, pitlane	100.0	0.0
apron	0.0	0.0
paved, parking	0.0	20.0
buildings, vegetation, other	0.0	100.0

Table 3.2: Class-specific cost factors for mapping classification errors to residuals.

The actual residuals are then calculated as weighted sums of all classification errors in the context and boundary regions. To this end, table 3.2 details the conversion factors from labels to error terms. Thereby, false positives denote pixels which are classified as road surface, but are currently located in one of the two contextual regions. False negatives on the other hand are pixels which are classified as non-road, but are at the moment located within the boundary regions of the road surface. The distinction between road and non-road is however not always unambiguous: Some classes like paved surfaces, parking lots and aprons may or may not be part of an actual road surface. For this reason, the conversion factors for these classes have been manually chosen based on this author’s experience. As a result, these cost factors are significantly lower than for classes with definitive mappings.

Furthermore, all classification errors are also weighted based on their actual distance from the border of the road segment quad as visualized in figure 3.6. This additional function has been instigated because misclassified pixels which are just a

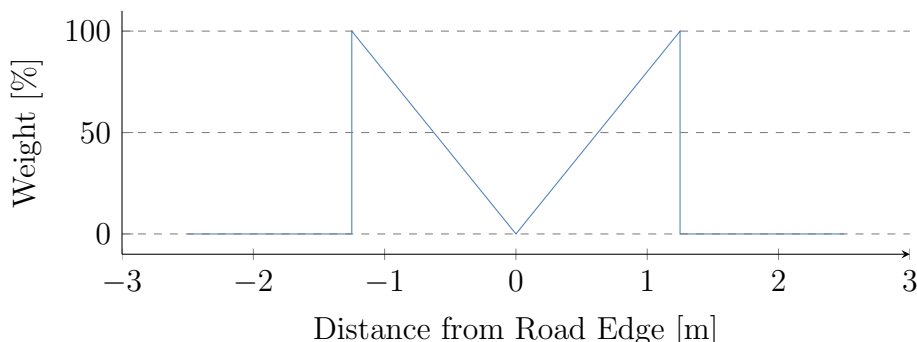


Figure 3.6: Distance-specific cost for classification errors.

few centimeters away from the estimated perimeter should not cause severe alterations. On the other hand however, classification errors which are located far from the segment’s border should have greater influence on the optimization results.

The product of the classification error cost c according to table 3.2 and the distance factor d is then calculated for each pixel p within the four significant regions of the road segment. These terms are summed up and normalized based on the total number of evaluated pixels. In addition, we introduce a global weight factor w_c which determines the relative importance of the classification error cost function in the context of our model. This finally yields the complete equation for the residuals r_c :

$$r_c = \frac{w_c}{\sum_p d(p)} \cdot \sum_p c(p) \cdot d(p) \quad (3.3)$$

Equation 3.3 is evaluated separately for context and road boundary regions, that is twice per road segment. This creates a total of $2 \cdot (n - 1)$ residuals for a path with n nodes. The cost function for the computation of residuals from classification errors is however not differentiable. Therefore, we have to rely on the numeric differentiation method in order to perform a nonlinear optimization.

Additional Cost Functions

While the calculation of classification errors is a substantial part of our model, we also implement three other, albeit much simpler, cost functions. These favor the creation of smooth realistic road surfaces by computing residuals for certain undesirable properties.

First and foremost, we identify the magnitude of the orientation difference between two road segments as such an adverse feature. Consequently, our model prefers the creation of topographies with larger corner radii whenever possible. In addition, this also corrects irregular variations of the original path by distributing isolated curvature peaks across multiple connected segments.

To this end, we have to perform a numerically stable calculation of the angle θ between two adjacent road segments v_1 and v_2 . The classical formula based on the inverse cosine of the dot product however requires the normalization of the input

vectors, which is computationally expensive due to two separate square root function calls. For this reason, we derive equation 3.8 to compute θ in a more efficient way. This leads to a speed-up of more than 25% without losing any significant digits (also see Kahan, 2006, p. 46):

$$\tan(\theta) = \frac{\sin(\theta)}{\cos(\theta)} \quad (3.4)$$

$$\theta = \text{atan2}(\sin(\theta), \cos(\theta)) \quad (3.5)$$

$$\|v_1 \times v_2\| = \|v_1\| \|v_2\| \sin(\theta) \quad (3.6)$$

$$v_1 \cdot v_2 = \|v_1\| \|v_2\| \cos(\theta) \quad (3.7)$$

$$r_a = w_a \cdot \theta = w_a \cdot \text{atan2}(\|v_1 \times v_2\|, v_1 \cdot v_2) \quad (3.8)$$

The resulting angle is then also multiplied by a global weight factor w_a to yield the desired residual r_a . Equation 3.8 is evaluated for any two connected road segments and a path made up from n nodes thus produces a total number of $n - 2$ residuals.

Finally, we also define two cost functions for the difference between the width and offset variables of two connected nodes in order to punish strong variations of these properties within a single road segment. The functions are directly derived from the parameters of our model and evaluated once per road segment:

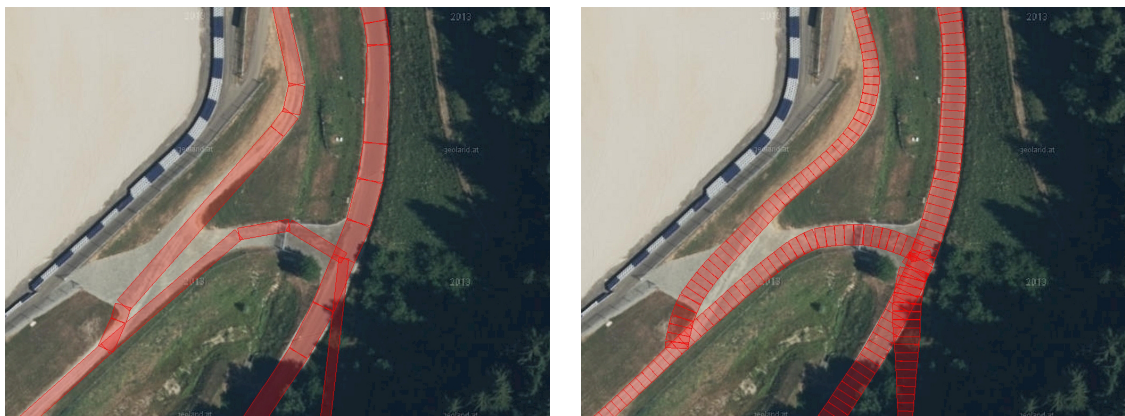
$$r_w = w_w \cdot (\text{width}_n - \text{width}_{n-1}) \quad (3.9)$$

$$r_o = w_o \cdot (\text{offset}_n - \text{offset}_{n-1}) \quad (3.10)$$

Two separate, global scaling factors w_w and w_o furthermore determine the relative importance of width and offset variations for solving the system.

3.3.3 Experiments and Results

A model which is built using the previously described cost functions can create state of the art refinement results for OSM data straight away. However, our initial experiments have shown that an insufficient spatial resolution of roads in the OSM data,



(a) Original OSM data

(b) Interpolated OSM data

Figure 3.7: Road graph interpolation. Refinement results using NLO.

as is demonstrated in figure 3.7a, often prevents us from utilizing our cost functions to their full potential. For this reason, we modify the original OSM data by first performing a linear interpolation of all paths so that the average distance between two connected nodes is reduced down to 2 meters. All of our results in this section have been obtained from using such an interpolated data set, and an example output of the entire algorithm is visualized in figure 3.7b.

Feature	Weight Factors	
	Set 1	Set 2
Classification w_c	10.0	10.0
Road Segment Orientation w_a	1.0	20.0
Width Variation w_w	0.2	10.0
Offset Variation w_o	0.2	0.0

Table 3.3: Refinement weights for nonlinear optimization.

In addition, we also found that the quality of the refinement results mainly depends on the application-specific relative weights of the four cost functions. Table 3.3 details two different sets of weight coefficients, which were manually selected based on a limited number of supervised experiments: The first set focuses on the classification error cost and thus features comparatively lower weights for the smoothness residuals. These weights are however significantly increased for the second set of optimization parameters, which consequently favors smoothness over classification accuracy. An automatic search for these parameter values could definitely lead to even better refinement results, but any such experiment would also require a much larger ground truth data set and a cross-validation approach to detect over-fitting issues.

Weights	Max. Iterations	IoU	Avg. Width σ^2
Set 1	1×20	80.09%	0.378m
	1×40	80.19%	0.370m
	2×20	80.40%	0.410m
Set 2	1×20	65.97%	0.091m
	1×40	65.89%	0.084m
	2×20	66.38%	0.069m
Set 1+2	20 + 20	80.32%	0.179m

Table 3.4: Refinement Results for nonlinear optimization on validation data set.

Table 3.4 lists the refinement results for our algorithm and both parameter sets in terms of the corresponding IoU values. Moreover, we also publish the average variance σ^2 of the refined road segments' width as a measure for the smoothness of

the obtained road boundaries:

$$\sigma^2 = \frac{\sum (X - \mu)^2}{N} \quad (3.11)$$

The IoU values for parameter set 1 thereby confirm our initial assumption that a nonlinear optimization can be used to create state of the art road map refinement results. However, the smoothness of the optimized road boundaries is not quite acceptable for our application, and it even degrades with the number of optimization iterations. Parameter set 2 on the other hand creates a very smooth road network at the expense of the IoU values. While both properties continue to improve slightly due to added iterations and multiple consecutive executions, it is obvious that the relative influence of the semantic segmentation is not sufficient to reach high IoU values.

As a result of these observations, we suggest that the quality of the refinement can be improved by a two-step algorithm: During the first run, we optimize the road map to closely match the semantic segmentation by applying the weight coefficients from set 1 for 20 iterations. Subsequently, we use this data as initial condition for a second run of another 20 iterations, but this time we set the weights according to parameter set 2. This approach enables us to strike a good balance between accurate results and an acceptable smoothness of the refined road segments: An IoU value of 80.32% after 40 iterations exceeds our reference value of 77.23% significantly, while the average variance of the road width for a given path is less than 0.18 meters or 1.4 pixels.

Supplementary experiments have shown that any further increase of the maximum number of iterations only has comparatively minor effects: A total of 80 iterations, split equally across the two runs, results in an IoU value of 80.31% and a corresponding road width variation of 0.17 meters. We therefore conclude that 20 iterations per pass are already sufficient to achieve excellent results. In order to speed up the calculations, it could even be advisable to relax the convergence criteria slightly.

Performance Comparison

Using our method, we are able to optimize the OSM road data in an area of 7.81 km² within 37 minutes. During this time, our algorithm refines the positions of 17714 nodes by analyzing the pre-calculated semantic segmentation for 400 million pixels. By comparison, the method proposed by Mattyus et al. solves the same problem in less than a third of the time, so our approach is significantly slower. However, the results of our method are more suitable for the generation of virtual road networks, as the smoothness of the resulting road surface is inherently enforced.



Figure 3.8: Example output of our algorithm: A road graph for a region of 0.26 km^2 after the NLO-based refinement. All connections between road segments are preserved.

3.4 Three-dimensional Road Maps

In order to create real world driving scenarios, we also have to ensure that the simulated driving resistance is calculated based on a realistic DEM. To this end, we enhance our refined, two-dimensional road network by also adding annotations for the elevation of each node. We thereby aim to seamlessly integrate the resulting, three-dimensional road surfaces with the surrounding terrain while also ensuring a smooth height profile alongside each path in the network.

Due to the lack of high resolution measurement data, we propose to create the required annotations by exploiting publicly available terrain and elevation models which are defined as uniform grids in the UTM coordinate system. The SRTM project⁴ provides a complete digital elevation model of the earth, sampled at a 30 m interval, but local governments often distribute data of higher quality. Our experiments in this section will thus be conducted using a DTM published by Land Steiermark⁵, which is available for UTM zone 33N at a grid resolution of 10 meters. Thus, we can directly perform a bilinear interpolation or a cubic convolution interpolation (see Keys, 1981) on this grid to smoothly sample the elevation data.

3.4.1 Problem Formulation

As a result of the comparatively low resolution of the available DTM, the interpolated values may however still be flawed. Road elevations next to steep faces and height annotations for bridges and tunnels are both unreliable. Consequently, a post-processing step is required to detect and filter these deficiencies in order to create a smooth road gradient. For this reason, we introduce two contradicting measures for the refined road profiles: We assess the deviation Δ_h from the original DTM as well as the energy of the resulting road gradient ($\tan(\alpha_n)$) and aim to achieve a balance between those two qualities. To this end, we introduce equations 3.12 and 3.13 in order to calculate the normalized error terms E_h and E_{RG} for each path with N nodes.

$$E_h = \frac{1}{N} \sum_{n=1}^N \Delta_{h,n}^2 \quad (3.12) \quad E_{RG} = \frac{1}{N} \sum_{n=1}^N (\tan(\alpha_n))^2 \quad (3.13)$$

While the energy of the road gradient can be reduced efficiently by calculating a moving average of the elevation profile, this simplistic method also causes significant alterations of the absolute height data. For this reason, we propose a different method for optimizing the road profile by reusing the nonlinear optimization framework described in section 3.3.1. To this end, we perform a model-based optimization of the road gradient's energy which also takes the original DTM values into account. Furthermore, we demonstrate that our approach can also be used to correct measurement errors in the DTM data with the help of a suitable loss function.

⁴United States Geological Survey, 2016. *Shuttle Radar Topography Mission*. <http://srtm.usgs.gov>

⁵*Digitales Geländemodell - 10m - Steiermark* (Land Steiermark, 2016a)

3.4.2 Optimization Method

As described previously, a model for a nonlinear least squares optimization problem is made up from a set of weighted functions which each calculate the cost of an undesirable property. Our method is based on defining costs for both the road gradient and the distance between the road surface and the elevation model. Consequently, the proposed model is parameterized in terms of the absolute height h of the road center points, and we use the original values of the DTM as initial condition.

Road Gradient Cost

The trajectory for real roads is chosen so that steep gradients are avoided whenever possible - even if this requires the construction of bypasses, bridges or tunnels. For this reason, it is safe to assume that the gradient of our refined trajectory also should be minimal and we thus identify the absolute value of the road gradient itself as an adverse property. This gradient is calculated as the quotient of the height difference between the segment's end points (rise) and the length of the segment as projected in the XY plane (run). Furthermore, the residual r_g is calculated separately for each road segment by multiplying this value with a weight factor w_g :

$$\text{gradient} = \tan \alpha = \frac{\text{rise}}{\text{run}} \quad (3.14)$$

$$r_g = w_g \cdot \text{gradient} \quad (3.15)$$

Road Gradient Variation Cost

In addition, we also define a cost function for the difference between the road gradients of two connected segments in order to punish strong road gradient variations regardless of the length of the involved segments and thus ensure a smooth road profile. Once more, we introduce an additional weight factor w_v in order to set the relative importance of this property and calculate the residuals for any pair of connected segments:

$$r_v = w_v \cdot (\text{gradient}_n - \text{gradient}_{n-1}) \quad (3.16)$$

Elevation Difference Cost

Moreover, the scaled difference between the current parameterization h_n of the model and the original DTM elevation data h_{DTM} is used to implement a final cost function, which is evaluated for each node of the path.

$$r_e = w_e \cdot \Delta_{h,n} = w_e \cdot (h_n - h_{\text{DTM}}) \quad (3.17)$$

In this case however, we also have to deal with errors in the DTM data. For this reason, we also add a Cauchy loss function with a normalization value of 0.5 in order to reduce the influence of larger residuals.

3.4.3 Experiments and Results

We assess the algorithms twice by applying both a bilinear and a cubic convolution interpolation on the DTM data and calculating the previously defined quality measures for each refinement method. The results are moreover also compared to an improved elevation model, which has been manually modified to correct for some obvious errors in the original data set such as missing bridges. The scores for the NLO algorithm are thereby obtained by using empirically determined cost function weights as listed in table 3.5.

Feature	Weight Factor
Road Gradient Cost w_g	1.2
Road Gradient Variation Cost w_v	12.0
Elevation Difference Cost w_e	0.2

Table 3.5: Refinement weights for nonlinear optimization of road height.

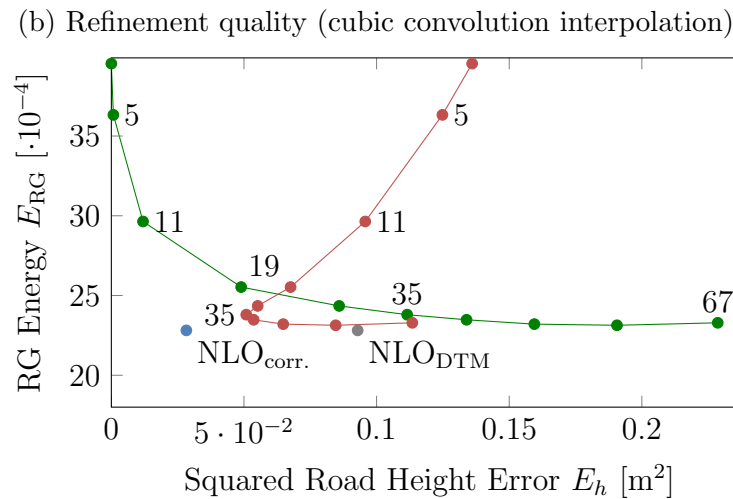
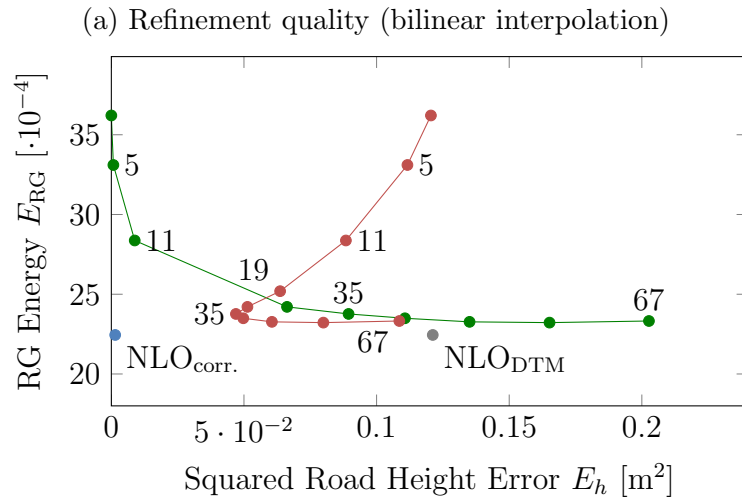
Selected results are given in numerical form in table 3.6. Without additional smoothing, the evaluation of the DTM with a bilinear interpolation results in an average road gradient energy of $36.208 \cdot 10^{-4}$. As is to be expected, this quantity can be improved considerably by applying moving average filters with large window sizes, which however also cause significant deviations from the original DTM data.

Interpolation	Smoothing (Filter Size)	$E_{\text{RG}} [\cdot 10^{-4}]$	$E_h [\text{m}^2]$	
			DTM	DTM _{corr.}
Bilinear	None	36.208	0.000	0.120
	Moving Avg. (11)	28.371	0.009	0.088
	Moving Avg. (35)	23.765	0.089	0.047
	Moving Avg. (67)	23.322	0.203	0.109
	NLO	22.447	0.121	0.001
Cubic Conv.	None	39.542	0.000	0.136
	Moving Avg. (11)	29.641	0.012	0.096
	Moving Avg. (35)	23.798	0.112	0.051
	Moving Avg. (67)	23.289	0.229	0.113
	NLO	22.814	0.093	0.028

Table 3.6: Refinement results for road height estimation.

When we further analyze the results for the moving average filters and compare them to the manually corrected DTM, we find that a window size of 35 samples is a good compromise for both interpolation algorithms: At this size, errors in the DTM are partially corrected due to the clearly visible smoothing effect while the absolute height difference is hardly noticeable. **Our NLO-based method however consistently outperforms the moving average filters** by a significant margin: The

energy of the road gradient is reduced to an absolute minimum while the elevation values are kept close to the DTM. In addition, the introduction of a loss function enables us to efficiently correct minor errors in the data set, as unrealistic road gradients are automatically corrected by isolated modifications to the elevation model. These findings are also visualized in figure 3.9.



	DTM	corr. DTM
Moving average filter vs.	█	█
Nonlinear optimization vs.	█	█

Figure 3.9: Graphical evaluation of refinement quality. Road gradient energy E_{RG} [-] vs. squared height error E_h [m²]. An ideal compromise between these two properties would have to be located in the lower left corner of each plot.

The practical applicability of our method is finally demonstrated by a visualization of the interpolated and optimized road heights at the Red Bull Ring circuit. Both the correction of a major error in the published DTM (figure 3.10a) and the reduced noise in the road gradient signal (figure 3.10b) are thereby clearly visible.

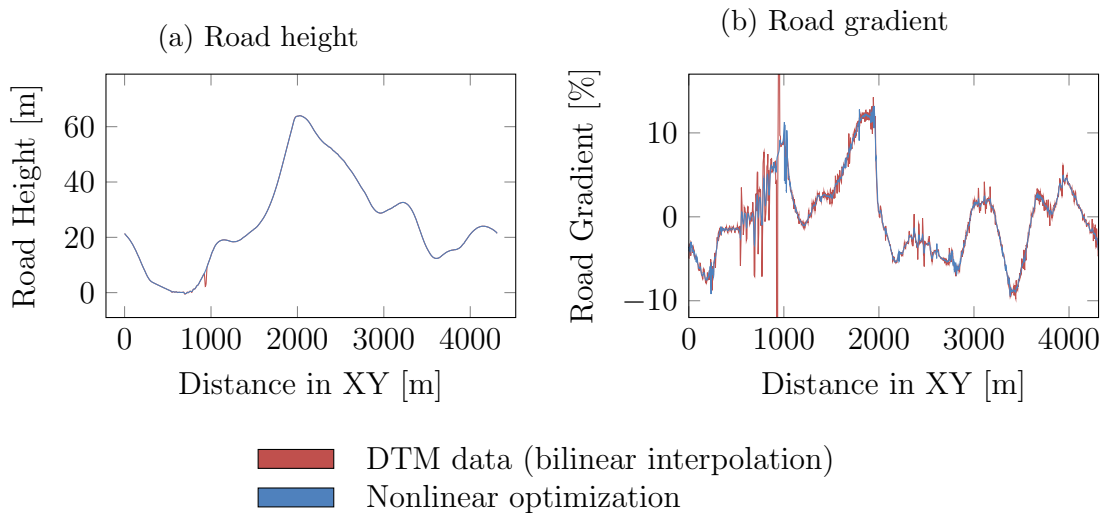


Figure 3.10: Road height and gradient over projected distance.

3.5 Summary

This chapter has established that a nonlinear optimization framework can be used to generate state of the art refinement results for road networks based on a semantic segmentation of aerial images. In addition, the same set of tools can also be exploited to create smooth, three-dimensional road surfaces which are based on existing terrain models. Moreover, these models can be textured using both the orthographic aerial images or the corresponding semantic segmentation, as all these data sources have been converted to the same, UTM-based coordinate system and thus are already aligned with each other.

There are however still some limitations which will have to be addressed in future works. For once, our implementation of the road map refinement is not yet able to detect individual road lanes and thus fails to create reliable information for crossings and other irregular road sections. In addition, the algorithm would have to be improved in order to fully support the detection of tunnels and overpasses. Finally, the low resolution of the evaluated terrain models unfortunately prevents us from calculating the banking of road surfaces, which makes it difficult to perform a correlation between vehicle simulations and actual, on-road measurements. Despite of these issues, the data collected so far already enables us to create virtual, three-dimensional driving environments as will be demonstrated in the next chapter.

Chapter 4

Visualization and Practical Applications

Several outputs of the earlier pipeline stages have to be combined in order to form virtual driving environments which enable us to implement dynamic vehicle simulations and visualize their results. This chapter primarily describes the generation of three-dimensional meshes based on the previously extracted road and terrain descriptions and includes step-by-step instructions for this process. The resulting assets could be used with any visualization tool, but all screenshots hereinafter have been created using the AVLViewer program, which is bundled with several AVL software products. The features and architecture of this render engine will however only be discussed briefly, as the core of this application has been developed by third parties some years prior to this thesis.

4.1 Render Engine

The AVLViewer is built upon the Object-Oriented Graphics Rendering Engine OGRE¹. It has been originally developed as a tool for AVL's vehicle simulation model VSM² in order to allow for a visual analysis of simulation results. To this end, it supports the visualization of arbitrary roads by computing a generic landscape based on a three-dimensional definition of a reference line.

Each road definition is thereby expanded into a point cloud and a two-dimensional Constraint Delaunay Triangulation (CDT) is calculated from this data (see Shewchuk, 2002) (also see Ruppert, 1993; Chew, 1993). The enclosing terrain is generated in a way which minimizes the vertical tension between the road vertices and their surroundings. Moreover, procedurally created objects such as curbs and lane markings can be added to the scene. An example output of this algorithm is presented in figure 4.1. The resulting meshes are perfectly aligned with the original road definition and therefore can be used for real-time visualizations as well as the post-processing of

¹The OGRE Team, 2016. *O-O Graphics Rendering Engine (OGRE)*. <http://www.ogre3d.org/>

²AVL List GmbH, 2016. *AVL VSMTM*. <https://www.avl.com/-/avl-vsm-vehicle-simulation>



Figure 4.1: AVLViewer screenshot. Generic landscape.

simulation results. In order to provide for these applications, the AVLViewer program contains a wide range of purpose-built features, including specialized presentation elements for many properties of the simulated vehicle.

The underlying render engine has been gradually improved over time and currently supports vertex and pixel shaders as well as multiple light sources and camera effects. These features are however mainly used for the visualization of highly detailed car models, while the procedurally created landscapes still lack visual appeal.

4.2 Mesh Generation

The most pressing issue with respect to the automatic environment generation is however a lack of scalability, as the generic terrain is created as one single mesh. We aim to address this particular shortcoming by offering a new, tile-based mesh synthesis algorithm based on aerial images and a DTM as an alternative to the existing landscape generation method.

The entire scene is thereby split into smaller quadratic tiles, each spanning an area of 480×480 meters. This size has been chosen so that the resulting image resolution for a standard texture size of 2048×2048 pixels is a good match for the aerial photographs. Contrary to the original method, our mesh generation algorithm thus does not create the terrain based on the road descriptions, but rather intends to produce road meshes which are compatible with these predefined terrain tiles.

During the first phase of the mesh generation, all road definitions are therefore expanded into two-dimensional polygons, which are then intersected with and subtracted from the quadratic tile areas. This process often results in a significant number of independent track and terrain objects for a single tile, as each tile can contain mul-

multiple road segments and the terrain thus may be split several times. For this reason, it is necessary to compute all polygon operations in a numerically stable way, and we use the NetTopologySuite³ to this end.

Each terrain polygon is furthermore rasterized at a resolution of 10 meters. The exact location of points which are not part of the object's boundary is moreover slightly randomized in order to avoid visual artifacts. We once more use the NetTopologySuite in order to triangulate the resulting two-dimensional point cloud, as this package also contains algorithms for the calculation of a CDT based on Shewchuk's method. Likewise, a two-dimensional CDT is computed independently for each of the obtained road polygons.

In order to expand the resulting polygons into three-dimensional meshes, the z-coordinates have to be determined in a consistent way for both terrain and road objects. To this end, two different data sources have to be evaluated: For all locations which are covered by or very close to a road surface polygon, the interpolated height of the refined road's optimized definition is being used. This also affects inner vertices of terrain objects, as long as they are within a predefined distance from a neighboring road polygon. The z-coordinate for all other vertices is however directly determined by a bilinear interpolation of the original DTM.

In addition, all meshes have to be generated in a way which avoids visible artifacts both at the tile borders and at the boundaries between road surfaces and the terrain objects. The normal vectors for boundary vertices thus have to be manually calculated based on the DTM and the road definitions in order to enable consistent lighting across the entire scene. Moreover, aerial images and prefabricated high resolution textures may also span multiple objects and the integrity of the generated UV coordinates therefore has to be ensured.

4.3 Application Example

The next sections provide step-by-step instructions for the generation and utilization of a virtual driving environment. We cover a region of approximately 20 km² enclosing the Red Bull Ring race track and provide visual examples for all intermediate results of the suggested workflow. The region's location in the Austrian mountains suits this example perfectly, as it enables us to demonstrate the full potential of the terrain generation system.

4.3.1 Generation of a Virtual Driving Environment

As a necessary prerequisite for our approach, aerial images have to be retrieved from OpenGIS compatible data sources so that they can be stored in a local WMTS database. In addition, a suitable terrain model and priors for the location of road

³NetTopologySuite - Team, 2016. *NetTopologySuite: A .NET GIS solution that is fast and reliable for the .NET platform.* <https://www.nuget.org/packages/NetTopologySuite/>

segments have to be provided. This example once more uses data from basemap.at⁴, Land Steiermark⁵ and the OpenStreetMap⁶ project.

Semantic Segmentation

The semantic segmentation of aerial images has been described in detail in chapter 2. In order to ensure seamless results, each original 256×256 pixel WMTS tile is padded with a 97-pixel wide margin made up from the pixels of neighboring tiles. The output of this process is visualized in figure 4.2a, which shows a projection of the semantic segmentation on the shaded DTM surface.

Road Map Refinement

Likewise, the refined road network is displayed as an additional layer on top of the original aerial images in figure 4.2b. The refinement method described in chapter 3 has however been slightly altered for this example in order to create even smoother road networks. To this end, an additional optimization step has been added which further reduces the energy of the roads' curvature by applying a model-based filtering similar to the process described in section 3.4. Moreover, the algorithm for refining the coordinates of road intersections has been modified, as high level roads should have more influence on their exact location than secondary paths. These two enhancements not only improved the visual appearance of the refined road network, but also resulted in a minor increase of the overall IoU value from 80.32% to 80.69% as measured on our validation data set.

Constrained Delaunay Triangulation

Figure 4.2c presents the CDT for this scenario, which has been calculated as described in section 4.2. In this case however, only the main parts of the race track have been subtracted from the terrain tiles, as the various foot paths and service roads will not be used by the actual vehicle simulation. The resulting tile boundaries are clearly visible as irregularities of the triangulated road surface.

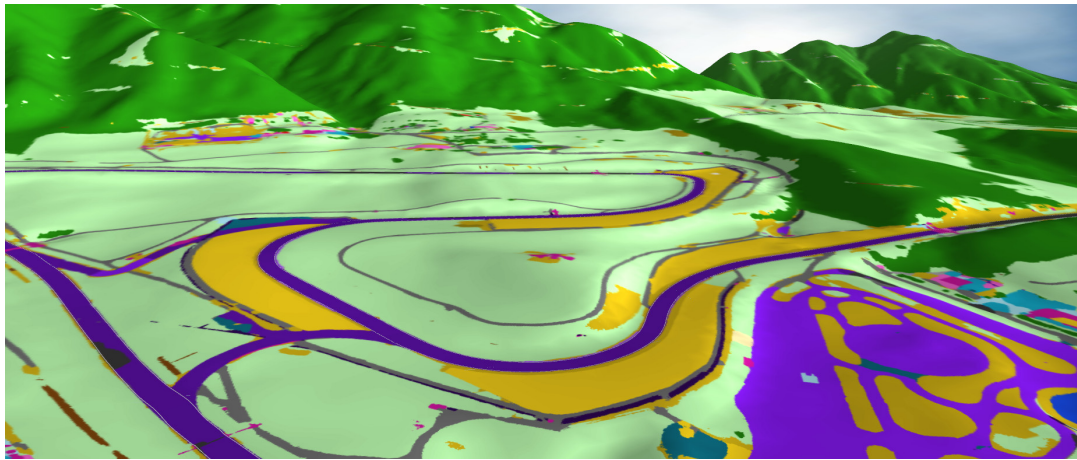
Texture Generation and UV Mapping

The calculation of textures from the aerial images is a resource intensive process, as the data has to be reprojected from the WMTS tiles to the UTM coordinate system. This task cannot be solved by a simple matrix transformation, but requires a lot of computations which have to be executed independently for each pixel. As such, our experiments have shown that the texture projection for an area of 1 km^2 takes up to 40 seconds, which is more than twice of the time required by all other steps of the mesh generation process combined.

⁴Stadt Wien und Österreichische Länder bzw. Ämter der Landesregierung, 2015

⁵*Digitales Geländemodell - 10m - Steiermark*. (Land Steiermark, 2016a).

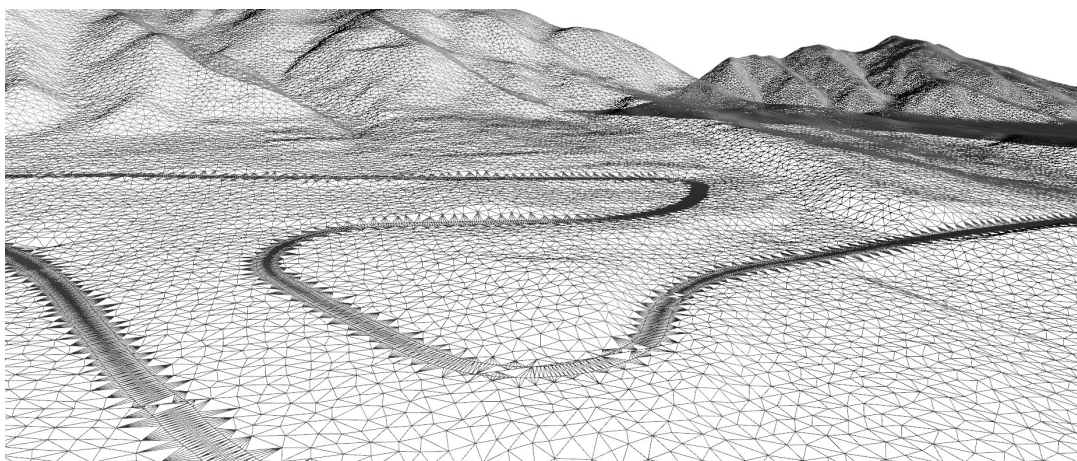
⁶Haklay and Weber, 2008



(a) Semantic segmentation



(b) Road map refinement

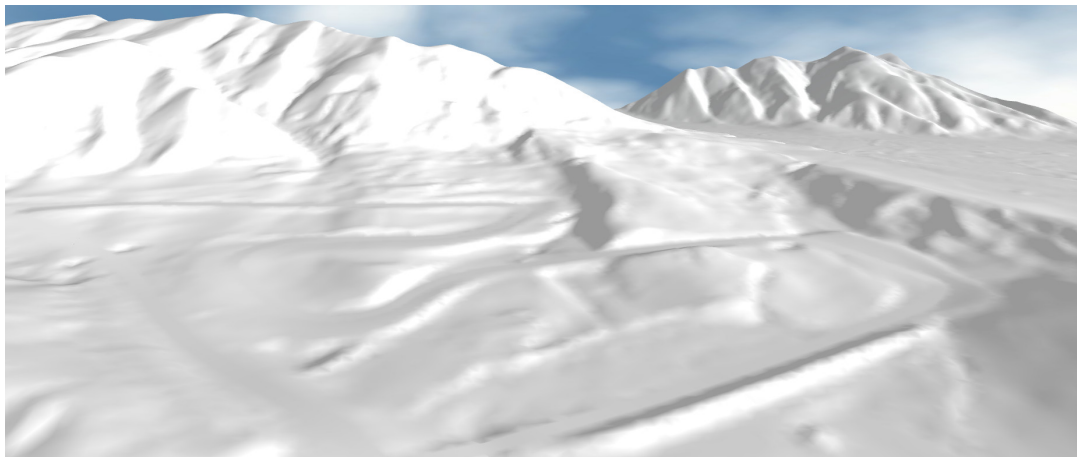


(c) Constrained Delaunay Triangulation (CDT)

Figure 4.2: Application example part I.



(a) Texture generation and UV mapping



(b) Shaded terrain model



(c) Virtual driving environment

Figure 4.3: Application example part II.

Figure 4.3a demonstrates the transition between aerial images and prefabricated high resolution textures which are used for the road surface. The blending is achieved by using alpha maps in combination with a second set of UV coordinates, which is oriented alongside the road surface and spans multiple tiles.

Shaded Terrain Model

A shaded terrain model for our scenario is presented in figure 4.3b. The boundaries between the individual tiles are not visible at all, as the normal vectors have been calculated consistently across the entire scene. However, a distinction between road surfaces and the surrounding terrain is clearly noticeable. This is a result of two different calculation procedures for the normal vectors: The road normals are calculated based on the gradient and banking of the road definition, while the terrain normals are taken straight out of the DTM. Consequently, road meshes appear as smooth surfaces in this demonstration, whereas the normal vectors of the enclosing terrain vary significantly and thus convey a certain degree of roughness.

Virtual Driving Environment

Figure 4.3c visualizes the finished virtual driving environment. To this end, aerial images and high resolution textures are mapped onto the shaded terrain. The scene also contains additional objects such as lane markings and curbs, which have been procedurally generated based on the road definition.

4.3.2 Simulation Results

At last, a vehicle simulation is conducted in the previously generated virtual driving environment to provide conclusive evidence for the suitability of the proposed workflow and the topographically correct extraction of road surfaces. For this reason, we import the road definition into the simulation software AVL VSM 4.1⁷. This application contains a range of tools for the generation and post-processing of track descriptions as well as for the computation of ideal racing lines. However, we deliberately do not use any of those features, as we intend to analyze the simulation results for the raw, unmodified road center lines as extracted from the semantic segmentation and the DTM.

To this end, a generic vehicle parameterization for sport cars, which is distributed as part of the AVL VSM installation, is furthermore used. Figure 4.4 presents the simulation results for one complete lap with this car at the virtual Red Bull Ring race track. The velocity and lateral acceleration signals thereby perfectly fit our expectations, as the influence of the individual corners of the circuit is clearly visible. Moreover, the shape of the vehicle speed signal is different for the full load accelerations towards the second and third corner of the track, which indicates that the road gradient is correctly evaluated during the simulation of this mountain course.

⁷AVL List GmbH, 2016. *AVL VSMTM*. <https://www.avl.com/-/avl-vsm-vehicle-simulation>

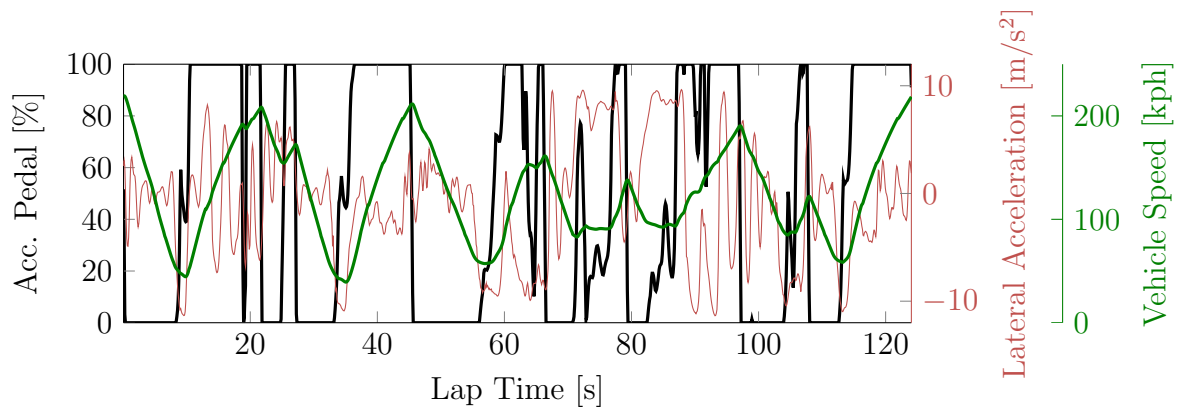


Figure 4.4: Simulation results.

A full comparison of the simulation results and actual, on-road measurements has however not been carried out yet, as this would not only require a perfectly matched parameterization for the simulated vehicle, but also an exact reproduction of the racing line within the boundaries set by the extracted road surfaces.⁸

Despite of this limitation, the application example presented herein successfully proves that the extracted road definitions are of sufficient quality to support the vehicle development process by facilitating the creation of realistic virtual driving environments. Figure 4.5 finally demonstrates the visualization of the obtained simulation results by means of the previously described render engine.

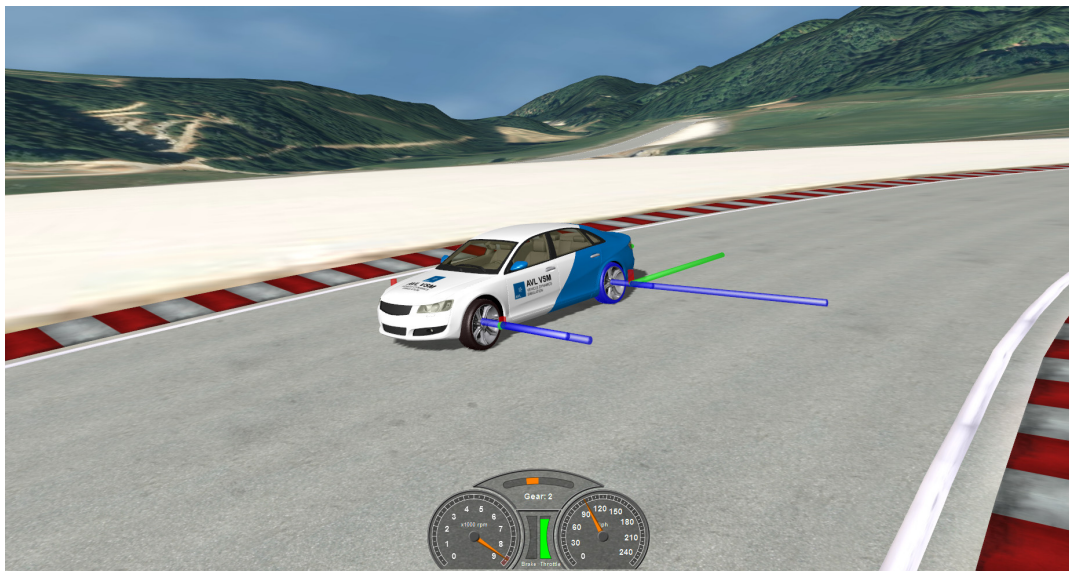


Figure 4.5: Visualization of simulation results.

⁸Existing data which fits these requirements is unfortunately subject to confidentiality agreements which prevent the use as part of this work, and the creation of a new reference data set clearly exceeds the scope of this thesis.

Chapter 5

Conclusion

This thesis suggested a workflow for the creation of large, virtual driving environments based on aerial images and other, publicly available, data sources. To this end, a semantic segmentation of orthographic pictures, computed by a purpose-built FCN, has been used to refine and annotate existing road maps. The obtained refinement results improve upon the state of the art when evaluated against a new ground truth data set that has been generated as part of this work.

Moreover, a topographically correct, three-dimensional description of road networks has been calculated based on a low resolution DTM and by means of a nonlinear optimization algorithm. The resulting surfaces are a suitable input for real-time vehicle simulations. In addition, the alignment between aerial images, extracted road definitions and the DTM facilitates the visualization of simulation results as part of a virtual landscape.

5.1 Interpretation of Results

These findings verify the central hypothesis that a FCN-based, semantic segmentation of aerial images is an adequate first step in a workflow for creating realistic virtual worlds. A procedural generation of driving environments based on the predefined pixel classes is definitely possible, and the use of nonlinear optimization tools enables the extraction of road surfaces while striking an optimum balance between accuracy and smoothness.

The algorithms which are described in this work have been implemented as set of independent command line tools and are already used successfully for commercial applications. For this reason, we conclude that the proposed workflow is highly suitable for the generation of virtual driving environments, and that the primary objective of this thesis thus has been accomplished.

5.2 Limitations

Due to the limited availability of ground truth data, the results of both the segmentation and road map refinement stages do not have much significance in a global context. Even though the performance measures for both methods exceed the respective state of the art when tested against the custom validation data sets, additional experiments would have to be conducted in order to reach a final verdict.

Furthermore, complicated road segments such as overpasses, tunnels and elaborate intersections have been dealt with only in a superficial and ultimately inadequate way. The detection of these structures could without a doubt be improved by exploiting the terrain model in earlier stages of the workflow. In addition, the suggested process for the generation of three-dimensional meshes from geometry definitions by using a CDT currently does not support multi-level intersections at all and thus would have to be enhanced as well. Such complicated structures are however comparatively rare and as such, they do not have any significant impact on the quality of the overall refinement results presented herein. Thus, it may be more efficient to resolve any remaining ambiguities with respect to the elevation model during a manual post-processing step whenever necessary.

5.3 Outlook and Future Work

This thesis only scratched the surface of many related, worthwhile topics. As such, there is a wide range of ideas for rewarding subsequent work. First and foremost, the integration of inertial measurement data into the workflow is of particular interest, as the exact shape of road surfaces could also be inferred from such sources. Moreover, this would enable exact correlations with on-road vehicle measurements, as roughness and banking signals could be incorporated in the virtual model based on the IMU data and additional ride height sensors. In addition, the road map refinement itself could be enhanced to detected multiple road lanes, which would be beneficial for ADAS development.

Furthermore, the scalability of the real-time visualization should be improved. The current process already supports the generation of LOD meshes, which helps to increase the render performance. However, the third-party render engine itself prevents us from loading meshes and textures in background threads, which is a severe obstacle for the visualization of larger virtual landscapes. Once this issue is resolved, the visualization of landscapes covering hundreds of square kilometers should become possible.

Finally, the procedural generation of animated environments should be addressed. The existing render engine is however showing its advancing age, as the underlying object model is outdated and many features have to be configured on a low level. For these reasons, it may be advisable to recreate the entire visualization module using a modern game engine while also relying on prefabricated assets to create a cutting edge experience for interactive driving simulators.

Bibliography

- Achanta, Radhakrishna, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk (2012). “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 34.11, pp. 2274–2282. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2012.120. URL: <http://dx.doi.org/10.1109/TPAMI.2012.120>.
- Bradski, G. (2000). “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools*.
- Campos, Carlos, João Miguel Leitão, and António Fernando Coelho (2015). “Integrated Modeling of Road Environments for Driving Simulation”. In: *Proceedings of the 10th International Conference on Computer Graphics Theory and Applications (VISIGRAPP 2015)*, pp. 70–80. ISBN: 978-989-758-087-1. DOI: 10.5220/0005308600700080.
- Canny, John (1986). “A computational approach to edge detection”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6, pp. 679–698.
- Cao, Lili and John Krumm (2009). “From GPS traces to a routable road map”. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, pp. 3–12.
- Charaniya, Amin P., Roberto Manduchi, and Suresh K. Lodha (2004). “Supervised parametric classification of aerial lidar data”. In: *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW’04. Conference on*. IEEE, pp. 30–30.
- Chew, L. Paul (1993). “Guaranteed-quality mesh generation for curved surfaces”. In: *Proceedings of the ninth annual symposium on Computational geometry*. ACM, pp. 274–280.
- Clode, Simon, Peter J. Kootsookos, and Franz Rottensteiner (2004). “The automatic extraction of roads from LIDAR data”. In: *The International Society for Photogrammetry and Remote Sensing’s Twentieth Annual Congress*. Vol. 35. ISPRS, pp. 231–236.
- Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, pp. 248–255.
- Dollár, Piotr and C. Zitnick (2013). “Structured forests for fast edge detection”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1841–1848.

- Everingham, Mark, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman (2015). “The pascal visual object classes challenge: A retrospective”. In: *International Journal of Computer Vision* 111.1, pp. 98–136.
- Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik (2014). “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.
- Haklay, Mordechai and Patrick Weber (2008). “OpenStreetMap: User-Generated Street Maps”. In: *IEEE Pervasive Computing* 7.4, pp. 12–18. ISSN: 1536-1268. DOI: 10.1109/MPRV.2008.80. URL: <http://dx.doi.org/10.1109/MPRV.2008.80>.
- Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov (2012). “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580*.
- Hinz, Stefan and Albert Baumgartner (2003). “Automatic extraction of urban road networks from multi-view aerial imagery”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 58.1, pp. 83–98.
- Hu, Xiangyun, C. Vincent Tao, and Yong Hu (2004). “Automatic road extraction from dense urban area by integrated processing of high resolution imagery and lidar data”. In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*. Istanbul, Turkey, pp. 320–324.
- Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell (2014). “Caffe: Convolutional architecture for fast feature embedding”. In: *Proceedings of the ACM International Conference on Multimedia*. ACM, pp. 675–678.
- Kahan, William (2006). “How Futile are Mindless Assessments of Roundoff in Floating-Point Computation?” In: *Web. Work In Progress*. URL: <http://www.cs.berkeley.edu/~wkahan/Mindless.pdf>.
- Keys, Robert (1981). “Cubic convolution interpolation for digital image processing”. In: *IEEE transactions on acoustics, speech, and signal processing* 29.6, pp. 1153–1160.
- Kluckner, Stefan, Michael Donoser, and Horst Bischof (2010). “Super-pixel class segmentation in large-scale aerial imagery”. In: *Proceedings of Annual Workshop of the Austrian Association for Pattern Recognition*.
- Kluckner, Stefan, Thomas Mauthner, Peter M. Roth, and Horst Bischof (2009). “Semantic classification in aerial imagery by integrating appearance and height information”. In: *Computer Vision-ACCV 2009*. Springer, pp. 477–488.
- Krähenbühl, Philipp and Vladlen Koltun (2012). “Efficient inference in fully connected crfs with gaussian edge potentials”. In: *arXiv preprint arXiv:1210.5644*.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105. URL: [http:](http://)

- [//papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf](http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf).
- Lafferty, John, Andrew McCallum, and Fernando Pereira (2001). “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In: *Proceedings of the eighteenth international conference on machine learning, ICML*. Vol. 1, pp. 282–289.
- LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.
- Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2014). “Fully Convolutional Networks for Semantic Segmentation”. In: *CoRR* abs/1411.4038. URL: <http://arxiv.org/abs/1411.4038>.
- Madsen, Kaj, Hans Bruun Nielsen, and Ole Tingleff (2004). “Methods for non-linear least squares problems”. In:
- Mattyus, Gellert, Shenlong Wang, Sanja Fidler, and Raquel Urtasun (2015). “Enhancing Road Maps by Parsing Aerial Images Around the World”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1689–1697.
- McCulloch, Warren S. and Walter Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.
- Miao, Zelang, Wenzhong Shi, Hua Zhang, and Xinxin Wang (2013). “Road centerline extraction from high-resolution imagery based on shape features and multivariate adaptive regression splines”. In: *IEEE geoscience and remote sensing letters* 10.3, pp. 583–587.
- Mitchell, Tom M. (1997). *Machine Learning*. WCB/McGraw-Hill.
- Mordvintsev, Alexander, Christopher Olah, and Mike Tyka (2015). “Inceptionism: Going deeper into neural networks”. In: *Google Research Blog*. Retrieved June 20.
- Müller, Sönke and Daniel Wilhelm Zaum (2005). “Robust building detection in aerial images”. In: *International Archives of Photogrammetry and Remote Sensing* 36.B2 / W24, pp. 143–148.
- Pérez, Fernando and Brian E. Granger (2007). “IPython: a System for Interactive Scientific Computing”. In: *Computing in Science and Engineering* 9.3, pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: <http://ipython.org>.
- Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (1992). *Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. New York, NY, USA: Cambridge University Press. ISBN: 0-521-43108-5.
- Prince, S.J.D. (2012). *Computer Vision: Models Learning and Inference*. Cambridge University Press.
- Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: *NATURE* 323, p. 9.
- Ruppert, Jim (1993). “A New and Simple Algorithm for Quality 2-dimensional Mesh Generation”. In: *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '93. Austin, Texas, USA: Society for Industrial and

- Applied Mathematics, pp. 83–92. ISBN: 0-89871-313-7. URL: <http://dl.acm.org/citation.cfm?id=313559.313615>.
- Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. (2015). “Imagenet large scale visual recognition challenge”. In: *International Journal of Computer Vision* 115.3, pp. 211–252.
- Russell, Stuart and Peter Norvig (2010). *Artificial Intelligence. A Modern Approach*. 3rd ed. Pearson Education Inc.
- Sampath, Aparajithan and Jie Shan (2010). “Segmentation and reconstruction of polyhedral building roofs from aerial lidar point clouds”. In: *IEEE Transactions on geoscience and remote sensing* 48.3, pp. 1554–1567.
- Shewchuk, Jonathan Richard (2002). “Delaunay refinement algorithms for triangular mesh generation”. In: *Computational geometry* 22.1, pp. 21–74.
- Simonyan, Karen and Andrew Zisserman (2014). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556. URL: <http://arxiv.org/abs/1409.1556>.
- Snyder, John P. (1987). *Map Projections - A Working Manual*. U.S. Geological Survey professional paper; 1395; GA110.S577. Washington, DC 20402: United States Government Printing Office.
- Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). “Going deeper with convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.
- Van Zyl, Jakob J. (2001). “The Shuttle Radar Topography Mission (SRTM): a breakthrough in remote sensing of topography”. In: *Acta Astronautica* 48.5, pp. 559–565.
- Yang, Bisheng, Lina Fang, and Jonathan Li (2013). “Semi-automated extraction and delineation of 3D roads of street scene from mobile laser scanning point clouds”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 79, pp. 80–93.
- Zheng, Shuai, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr (2015). “Conditional random fields as recurrent neural networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1529–1537.

Miscellaneous Resources

- Agarwal, Sameer, Keir Mierle, et al. (2016). *Ceres Solver*. <http://ceres-solver.org>. Accessed: 2016-01-04.
- AVL List GmbH (2016). *AVL VSMTM*. URL: <https://www.avl.com/-/avl-vsm-vehicle-simulation>. Accessed: 2016-03-08.
- Dupuis, Marius, ed. (2015). *OpenDRIVE[®] Format Specification*. 1.4. VI2014.106. Accessed: 2016-09-25. VIRES Simulationstechnologie GmbH. URL: <http://www.opendrive.org/docs/OpenDRIVEFormatSpecRev1.4H.pdf>.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). “Deep Learning”. Book in preparation for MIT Press. URL: <http://www.deeplearningbook.org>.
- IPG Automotive GmbH (2016). *CarMaker[®]*. URL: <http://ipg.de/simulationso-lutions/carmaker>. Accessed: 2016-03-08.
- Klokan Technologies; Petr Pridal (2015). *Tiles à la Google Maps: Coordinates, Tile Bounds and Projection - conversion to EPSG:900913 (EPSG:3785) and EPSG:4326 (WGS84)*. URL: <http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/>. Accessed: 2015-11-04.
- Klokan Technologies; Petr Pridal, Tomas Pohanka and Radim Kacer (2015). *WGS 84 / Pseudo-Mercator - Spherical Mercator, Google Maps, OpenStreetMap, Bing, ArcGIS, ESRI - EPSG:3857*. URL: <https://epsg.io/3857>. Accessed: 2015-11-03.
- Land Steiermark (2016a). *Digitales Geländemodell - 10m - Steiermark*. URL: <http://data.steiermark.gv.at>. Accessed: 2016-03-10.
- (2016b). *Preise für Geodaten und Kartenerstellung*. URL: http://gis.stmk.gv.at/content/dokumente/Geomarket/Preisliste_Geodaten.pdf. Accessed: 2016-03-15.
- Masó, Joan, ed. (2014). *OGC Web Map Tile Service (WMTS) Simple Profile*. 1.0. OGC 13-082r2. Open Geospatial Consortium Inc. URL: <http://docs.opengeospatial.org/is/13-082r2/13-082r2.html>.
- Masó, Joan, Keith Pomakis, and Núria Julià, eds. (2010). *OpenGIS Web Map Tile Service Implementation Standard*. 1.0.0. OGC 07-057r7. Open Geospatial Consortium Inc. URL: http://portal.opengeospatial.org/files/?artifact_id=35326.
- Mechanical Simulation (2016). *CarSim[®]*. URL: <https://www.carsim.com/products/carsim>. Accessed: 2016-03-08.
- NetTopologySuite - Team (2016). *NetTopologySuite: A .NET GIS solution that is fast and reliable for the .NET platform*. URL: <https://www.nuget.org/packages/NetTopologySuite/>. Accessed: 2016-12-31.
- New South Wales Government (2016). *Land and Property Information Web Services*. URL: http://www.lpi.nsw.gov.au/mapping_and_imagery/lpi_web_services. Accessed: 2016-03-15.
- Office of Geomatics, ed. (2014). *The Universal Grids and the Transverse Mercator and Polar Stereographic Map Projections*. 2.0.0. NGA.SIG.0012.2.0.0-UTMUPS.

- National Geospatial Intelligence Agency. URL: http://http://earth-info.nga.mil/GandG/publications/NGA_SIG_0012_2_0_0_UTMUPS/NGA.SIG.0012_2.0.0_UTMUPS.pdf.
- Pomakis, Keith, ed. (2009). *OWS-6 DSS Engineering Report - SOAP/XML and REST in WMTS*. 0.3.0. OGC 09-006. Open Geospatial Consortium Inc. URL: http://portal.opengeospatial.org/files/?artifact_id=33269.
- Schweizerische Eidgenossenschaft (2016). *Image strips swisstopo*. URL: <https://opendata.swiss/en/dataset/luftbildstreifen-swisstopo>. Accessed: 2016-03-15.
- Stadt Wien und Österreichische Länder bzw. Ämter der Landesregierung (2015). *basemap.at – Verwaltungsgrundkarte Österreichs*. URL: <http://www.basemap.at/>. Accessed: 2015-11-04.
- The OGRE Team (2016). *O-O Graphics Rendering Engine (OGRE)*. URL: <http://www.ogre3d.org/>. Accessed: 2016-12-31.
- United States Geological Survey (2016). *Shuttle Radar Topography Mission*. URL: <http://srtm.usgs.gov/>. Accessed: 2016-03-15.

Appendix A

Ground Truth Data

A.1 Pixel Classes

The tables in this section list all the pixel classes which have been defined as part of this thesis to facilitate a detailed semantic segmentation of orthographic aerial images. The first column of each table indicates the position of a color in the palette when storing the classification results as indexed bitmaps.
















#	Color	Class	Description	
0		#FFFFFF	snow	snow covered terrain
1		#ACE6A1	grassland	grassland, meadows
2		#73E65C	lawn	grass courts, public parks, cultivated turf and lawn
3		#26990F	shrub	shrubs, bushes, ornamental plants
4		#1C730B	tree	leaf trees, conifers, dense forests
5		#73610B	orchard	plantations, orchards, vineyards
6		#99820F	field	cultivated (plowed) fields
7		#C0A313	dirt	dirt, soil, earth
8		#E6DAA1	debris	coarse gravel, boulders
9		#E6E6E6	rock	rocks and mountains
10		#A1DAE6	paved	cobbled areas, concrete, tamped earth
11		#13A3C0	parking	parking lots
12		#5C7EE6	pool	swimming pools
13		#174AE6	river	rivers and streams
14		#133EC0	water	natural pools, lakes, oceans

Table A.1: Pixel classes for various terrains.







#	Color	Class	Description	
15		#E65C5C	shed	sheds, annexes, garage buildings
16		#E61717	detached	detached houses, family homes
17		#C01313	residential	multi-story commercial and residential buildings, town houses
18		#990F0F	highrise	highrise buildings
19		#E65CC3	agricultural	stables, storage silos
20		#C01394	industrial	factories, hangars and other man-made structures

Table A.2: Pixel classes for buildings.











#	Color	Class	Description	
21		#99540F	railroad	rails and embankments
22		#733F0B	cartway	narrow farm and forest roads
23		#B3B3B3	sidewalk	for pedestrians and/or cyclists
24		#999999	macadam	gravel and macadam roads
25		#808080	driveway	private, paved driveways
26		#666666	secondary	paved secondary roads, optional lane and side markings
27		#4C4C4C	avenue	multi-lane inner city streets
28		#333333	highway	one lane per direction, compulsory lane and side markings
29		#E67E17	freeway	multi-lane, limited access roads
30		#E6A15C	ramp	single lane, oneway access ramps

Table A.3: Pixel classes for transportation.








#	Color	Class	Description	
31		#A15CE6	curbs	curbs next to track corners
32		#7E17E6	pitlane	pit lanes and proving grounds
33		#540F99	track	main circuit, wide race track
34		#2A074C	barrier	tyre barriers (acting as track borders)
35		#0B6173	apron	paved aprons and run-off areas
36		#730B59	grandstand	grandstands next to the track
37		#E6C317	gravel	used for sand and/or gravel traps

Table A.4: Auxiliary pixel classes for race tracks.



Figure A.1: Examples of labeled ground truth data (II). The first column contains the original aerial images^a, the second column the manually created segmentation. An overlay of both images is shown in the third column. Top row (a-c): Section of the Red Bull Ring race track. Middle row (d-f): City center of Graz, Austria. Bottom row (g-i): Rural area, Styria, Austria.

^aDatenquelle / Source of images: basemap.at; License: CC-BY 3.0 AT.

A.2 Ground Truth Images

Figure A.1 shows additional examples of labeled ground truth images. 16 such images - each with a size of 1024×1024 pixels (or approximately 208×208 meters) - were created manually. In addition, each of these larger images has been divided into 16 tiles, resulting in a total number of 256 ground truth tiles with a size of 256×256 pixels.

A.3 Distribution of Pixel Classes

These original 256 ground truth tiles have been partitioned into separate training and validation image sets according to a 3:1 ratio. The number of tiles was however not sufficient to achieve equal distributions of the 38 pixel classes in both data sets. Table A.5 and diagram A.2 therefore give the exact spread of pixel classes which was used during the training and validation stages.

Class	Train. [%]	Val. [%]	Class	Train. [%]	Val. [%]
snow	2.37	4.22	agricultural	0.35	0.52
grassland	26.35	20.67	industrial	1.40	0.68
lawn	2.08	2.71	railroad	0.23	0.32
shrub	1.06	1.18	cartway	0.55	0.46
tree	14.75	21.10	sidewalk	0.41	0.70
orchard	0.93	0.43	macadam	0.34	0.28
field	18.55	15.12	driveway	0.23	0.53
dirt	2.50	2.39	secondary	1.89	2.13
debris	0.72	1.18	avenue	0.90	2.58
rock	4.83	4.25	highway	1.20	1.57
paved	2.62	1.92	freeway	1.01	0.79
parking	4.28	4.79	ramp	0.56	0.13
pool	0.02	0.02	curbs	0.05	0.05
river	0.12	0.32	pitlane	1.96	1.15
water	0.14	0.00	track	1.43	1.80
shed	0.55	0.43	barrier	0.11	0.19
detached	0.46	0.63	apron	0.19	0.17
residential	2.43	1.91	grandstand	0.15	0.16
highrise	0.44	0.06	gravel	1.88	2.47

Table A.5: Pixel class distribution.

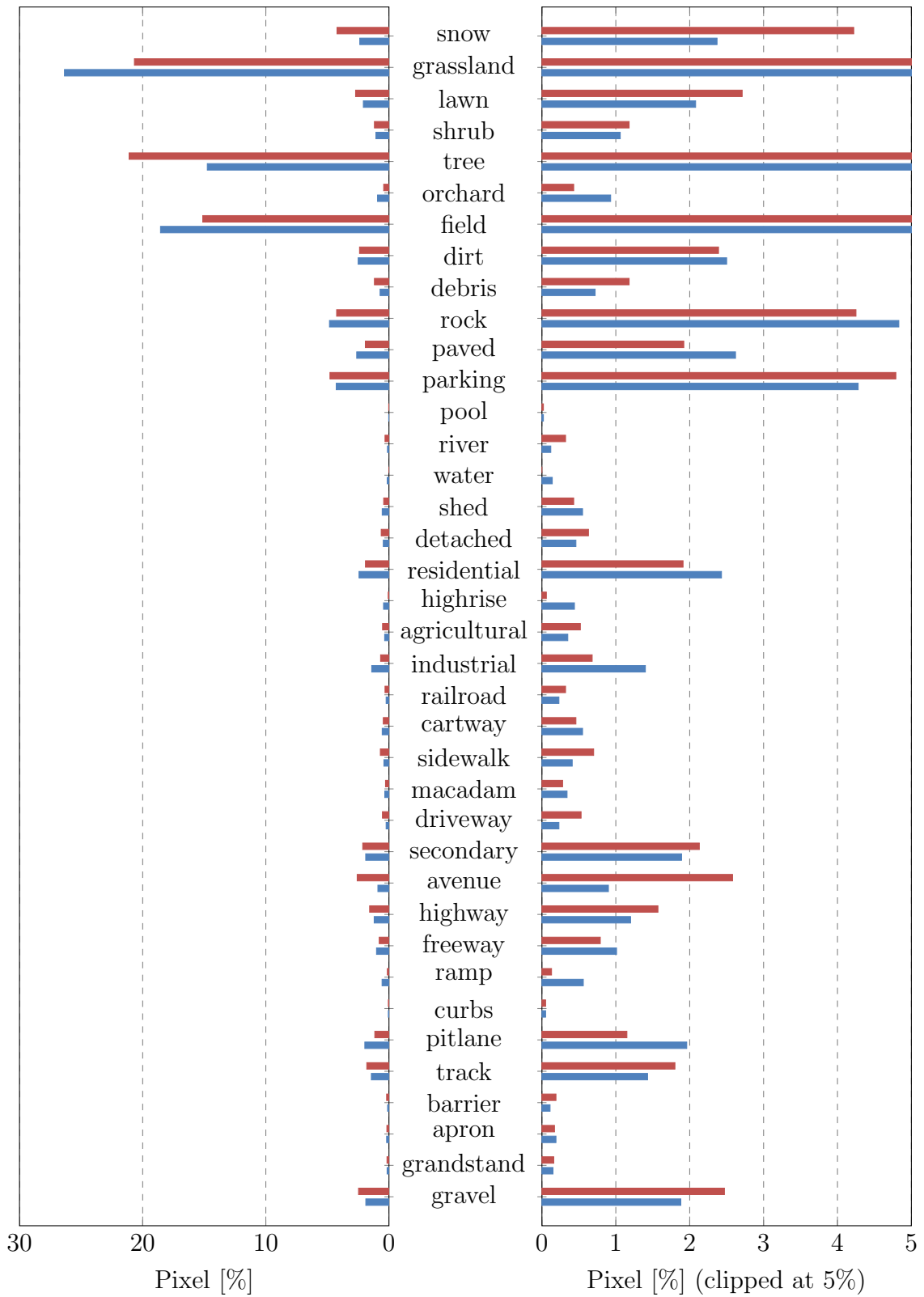


Figure A.2: Distribution of pixel classes for training (red) and validation (blue) data.

Appendix B

Network Architecture

The next sections detail the final architecture of the semantic segmentation networks analyzed in section 2.3 by means of the Caffe network description format.

B.1 VGG16-8s

```
1 name: "FCN"
  input: "data"
  input_dim: 1
  input_dim: 3
  input_dim: 450
6 layer{name: "conv1_1" type: "Convolution" bottom: "data" top: "conv1_1"
  convolution_param{num_output: 64 pad: 0 kernel_size: 3}}
  layer{name: "relu1_1" type: "ReLU" bottom: "conv1_1" top: "conv1_1"}
  layer{name: "conv1_2" type: "Convolution" bottom: "conv1_1" top: "conv1_2"
11 convolution_param{num_output: 64 pad: 1 kernel_size: 3}}
  layer{name: "relu1_2" type: "ReLU" bottom: "conv1_2" top: "conv1_2"}
  layer{name: "pool1" type: "Pooling" bottom: "conv1_2" top: "pool1"
  pooling_param{pool: MAX kernel_size: 2 stride: 2}}
16 layer{name: "conv2_1" type: "Convolution" bottom: "pool1" top: "conv2_1"
  convolution_param{num_output: 128 pad: 1 kernel_size: 3}}
  layer{name: "relu2_1" type: "ReLU" bottom: "conv2_1" top: "conv2_1"}
  layer{name: "conv2_2" type: "Convolution" bottom: "conv2_1" top: "conv2_2"
  convolution_param{num_output: 128 pad: 1 kernel_size: 3}}
  layer{name: "relu2_2" type: "ReLU" bottom: "conv2_2" top: "conv2_2"}
21 layer{name: "pool2" type: "Pooling" bottom: "conv2_2" top: "pool2"
  pooling_param{pool: MAX kernel_size: 2 stride: 2}}
  layer{name: "conv3_1" type: "Convolution" bottom: "pool2" top: "conv3_1"
  convolution_param{num_output: 256 pad: 1 kernel_size: 3}}
  layer{name: "relu3_1" type: "ReLU" bottom: "conv3_1" top: "conv3_1"}
26 layer{name: "conv3_2" type: "Convolution" bottom: "conv3_1" top: "conv3_2"
  convolution_param{num_output: 256 pad: 1 kernel_size: 3}}
  layer{name: "relu3_2" type: "ReLU" bottom: "conv3_2" top: "conv3_2"}
  layer{name: "conv3_3" type: "Convolution" bottom: "conv3_2" top: "conv3_3"
  convolution_param{num_output: 256 pad: 1 kernel_size: 3}}
31 layer{name: "relu3_3" type: "ReLU" bottom: "conv3_3" top: "conv3_3"}
  layer{name: "pool3" type: "Pooling" bottom: "conv3_3" top: "pool3"
  pooling_param{pool: MAX kernel_size: 2 stride: 2}}
  layer{name: "conv4_1" type: "Convolution" bottom: "pool3" top: "conv4_1"
  convolution_param{num_output: 512 pad: 1 kernel_size: 3}}
36 layer{name: "relu4_1" type: "ReLU" bottom: "conv4_1" top: "conv4_1"}
  layer{name: "conv4_2" type: "Convolution" bottom: "conv4_1" top: "conv4_2"
  convolution_param{num_output: 512 pad: 1 kernel_size: 3}}
```



```

layer{name: "relu4_2" type: "ReLU" bottom: "conv4_2" top: "conv4_2"}
layer{name: "conv4_3" type: "Convolution" bottom: "conv4_2" top: "conv4_3"}
41 convolution_param{num_output: 512 pad: 1 kernel_size: 3}}
layer{name: "relu4_3" type: "ReLU" bottom: "conv4_3" top: "conv4_3"}
layer{name: "pool4" type: "Pooling" bottom: "conv4_3" top: "pool4"}
pooling_param{pool: MAX kernel_size: 2 stride: 2}}
layer{name: "conv5_1" type: "Convolution" bottom: "pool4" top: "conv5_1"}
46 convolution_param{num_output: 512 pad: 1 kernel_size: 3}}
layer{name: "relu5_1" type: "ReLU" bottom: "conv5_1" top: "conv5_1"}
layer{name: "conv5_2" type: "Convolution" bottom: "conv5_1" top: "conv5_2"}
convolution_param{num_output: 512 pad: 1 kernel_size: 3}}
layer{name: "relu5_2" type: "ReLU" bottom: "conv5_2" top: "conv5_2"}
51 layer{name: "conv5_3" type: "Convolution" bottom: "conv5_2" top: "conv5_3"}
convolution_param{num_output: 512 pad: 1 kernel_size: 3}}
layer{name: "relu5_3" type: "ReLU" bottom: "conv5_3" top: "conv5_3"}
layer{name: "pool5" type: "Pooling" bottom: "conv5_3" top: "pool5"}
pooling_param{pool: MAX kernel_size: 2 stride: 2}}
56 layer{name: "fc6-conv" type: "Convolution" bottom: "pool5" top: "fc6"}
convolution_param{num_output: 4096 kernel_size: 7}}
layer{name: "relu6" type: "ReLU" bottom: "fc6" top: "fc6"}
layer{name: "drop6" type: "Dropout" bottom: "fc6" top: "fc6"}
dropout_param{dropout_ratio: 0.5}}
61 layer{name: "fc7-conv" type: "Convolution" bottom: "fc6" top: "fc7"}
convolution_param{num_output: 4096 kernel_size: 1}}
layer{name: "relu7" type: "ReLU" bottom: "fc7" top: "fc7"}
layer{name: "drop7" type: "Dropout" bottom: "fc7" top: "fc7"}
dropout_param{dropout_ratio: 0.5}}
66 layer{name: "score37" type: "Convolution" bottom: "fc7" top: "score37"}
convolution_param{num_output: 38 kernel_size: 1}}
layer{name: "upscore2" type: "Deconvolution" bottom: "score37" top: "upscore2"}
convolution_param{num_output: 38 kernel_size: 4 stride: 2}}
71 layer{name: "score-pool4" type: "Convolution" bottom: "pool4" top: "score-pool4"}
convolution_param{num_output: 38 kernel_size: 1}}
layer{type: 'Crop' name: 'crop'
bottom: 'score-pool4' bottom: 'upscore2' top: 'score-pool4c'}
layer{name: "fuse" type: "Eltwise" bottom: "upscore2" bottom: "score-pool4c"}
top: "score-fused" eltwise_param{operation: SUM}}
76 layer{name: "us-fused-16" type: "Deconvolution" bottom: "score-fused" top: "score4"}
convolution_param{num_output: 38 kernel_size: 4 stride: 2}}
layer{name: "score-p3" type: "Convolution" bottom: "pool3" top: "score-p3"}
convolution_param{num_output: 38 kernel_size: 1}}
layer{type: 'Crop' name: 'crop' bottom: 'score-p3' bottom: 'score4' top: 'score-p3c'}
81 layer{name: "fuse" type: "Eltwise" bottom: "score4" bottom: "score-p3c"}
top: "score-final" eltwise_param{operation: SUM}}
layer{name: "upsample" type: "Deconvolution" bottom: "score-final" top: "bigscore"}
convolution_param{num_output: 38 kernel_size: 16 stride: 8}}
86 layer{name: 'crop' type: 'CenterCrop' bottom: 'bigscore' top: 'score'}
transform_param{crop_size: 256}}

```

Network/vgg16-8s.prototxt

B.2 GoogLeNet-8s

```

name: "GoogLeNet"
input: "data"
input_shape{dim: 1 dim: 3 dim: 450 dim: 450}
4 layer{name: "c1/7x7_s2" type: "Convolution" bottom: "data" top: "c1/7x7_s2"
  convolution_param {num_output: 64 pad: 3 kernel_size: 7 stride: 2}}
layer{name: "c1/relu_7x7" type: "ReLU" bottom: "c1/7x7_s2" top: "c1/7x7_s2"}
layer{name: "pool1/3x3_s2" type: "Pooling" bottom: "c1/7x7_s2" top: "pool1/3x3_s2"
  pooling_param {pool: MAX kernel_size: 3 stride: 2}}
9 layer{name: "pool1/norm1" type: "LRN" bottom: "pool1/3x3_s2" top: "pool1/norm1"
  lrn_param {local_size: 5 alpha: 0.0001 beta: 0.75}}
layer{name: "c2/3x3_red" type: "Convolution" bottom: "pool1/norm1" top: "c2/3x3_red"
  convolution_param {num_output: 64 kernel_size: 1}}
layer{name: "c2/relu_3x3_red" type: "ReLU" bottom: "c2/3x3_red" top: "c2/3x3_red"}
14 layer{name: "c2/3x3" type: "Convolution" bottom: "c2/3x3_red" top: "c2/3x3"
  convolution_param {num_output: 192 pad: 1 kernel_size: 3}}
layer{name: "c2/relu_3x3" type: "ReLU" bottom: "c2/3x3" top: "c2/3x3"}
layer{name: "c2/norm2" type: "LRN" bottom: "c2/3x3" top: "c2/norm2"
  lrn_param {local_size: 5 alpha: 0.0001 beta: 0.75}}
19 layer{name: "pool2/3x3_s2" type: "Pooling" bottom: "c2/norm2" top: "pool2/3x3_s2"
  pooling_param {pool: MAX kernel_size: 3 stride: 2}}
layer{name: "i3a/1x1" type: "Convolution" bottom: "pool2/3x3_s2" top: "i3a/1x1"
  convolution_param {num_output: 64 kernel_size: 1}}
layer{name: "i3a/relu_1x1" type: "ReLU" bottom: "i3a/1x1" top: "i3a/1x1"}
24 layer{name: "i3a/3x3_red" type: "Convolution" bottom: "pool2/3x3_s2"
  top: "i3a/3x3_red" convolution_param {num_output: 96 kernel_size: 1}}
layer{name: "i3a/relu_3x3_red" type: "ReLU" bottom: "i3a/3x3_red" top: "i3a/3x3_red"}
layer{name: "i3a/3x3" type: "Convolution" bottom: "i3a/3x3_red" top: "i3a/3x3"
  convolution_param {num_output: 128 pad: 1 kernel_size: 3}}
29 layer{name: "i3a/relu_3x3" type: "ReLU" bottom: "i3a/3x3" top: "i3a/3x3"}
layer{name: "i3a/5x5_red" type: "Convolution" bottom: "pool2/3x3_s2" top:
  "i3a/5x5_red" convolution_param {num_output: 16 kernel_size: 1}}
layer{name: "i3a/relu_5x5_red" type: "ReLU" bottom: "i3a/5x5_red" top: "i3a/5x5_red"}
layer{name: "i3a/5x5" type: "Convolution" bottom: "i3a/5x5_red" top: "i3a/5x5"
  convolution_param {num_output: 32 pad: 2 kernel_size: 5}}
34 layer{name: "i3a/relu_5x5" type: "ReLU" bottom: "i3a/5x5" top: "i3a/5x5"}
layer{name: "i3a/pool" type: "Pooling" bottom: "pool2/3x3_s2" top: "i3a/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i3a/proj" type: "Convolution" bottom: "i3a/pool" top: "i3a/proj"
  convolution_param {num_output: 32 kernel_size: 1}}
39 layer{name: "i3a/relu_proj" type: "ReLU" bottom: "i3a/proj" top: "i3a/proj"}
layer{name: "i3a/output" type: "Concat"
  bottom: "i3a/1x1" bottom: "i3a/3x3" bottom: "i3a/5x5" bottom: "i3a/proj"
  top: "i3a/output"}
44 layer{name: "i3b/1x1" type: "Convolution" bottom: "i3a/output" top: "i3b/1x1"
  convolution_param {num_output: 128 kernel_size: 1}}
layer{name: "i3b/relu_1x1" type: "ReLU" bottom: "i3b/1x1" top: "i3b/1x1"}
layer{name: "i3b/3x3_red" type: "Convolution" bottom: "i3a/output" top: "i3b/3x3_red"
  convolution_param {num_output: 128 kernel_size: 1}}
49 layer{name: "i3b/relu_3x3_red" type: "ReLU" bottom: "i3b/3x3_red" top: "i3b/3x3_red"}
layer{name: "i3b/3x3" type: "Convolution" bottom: "i3b/3x3_red" top: "i3b/3x3"
  convolution_param {num_output: 192 pad: 1 kernel_size: 3}}
layer{name: "i3b/relu_3x3" type: "ReLU" bottom: "i3b/3x3" top: "i3b/3x3"}
layer{name: "i3b/5x5_red" type: "Convolution" bottom: "i3a/output" top: "i3b/5x5_red"
  convolution_param {num_output: 32 kernel_size: 1}}
54 layer{name: "i3b/relu_5x5_red" type: "ReLU" bottom: "i3b/5x5_red" top: "i3b/5x5_red"}
layer{name: "i3b/5x5" type: "Convolution" bottom: "i3b/5x5_red" top: "i3b/5x5"
  convolution_param {num_output: 96 pad: 2 kernel_size: 5}}
layer{name: "i3b/relu_5x5" type: "ReLU" bottom: "i3b/5x5" top: "i3b/5x5"}
59 layer{name: "i3b/pool" type: "Pooling" bottom: "i3a/output" top: "i3b/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i3b/proj" type: "Convolution" bottom: "i3b/pool" top: "i3b/proj"
  convolution_param {num_output: 64 kernel_size: 1}}
layer{name: "i3b/relu_proj" type: "ReLU" bottom: "i3b/proj" top: "i3b/proj"}

```

```

64 layer{name: "i3b/output" type: "Concat"
  bottom: "i3b/1x1" bottom: "i3b/3x3" bottom: "i3b/5x5" bottom: "i3b/proj"
  top: "i3b/output"}
layer{name: "pool3/3x3_s2" type: "Pooling" bottom: "i3b/output" top: "pool3/3x3_s2"
  pooling_param {pool: MAX kernel_size: 3 stride: 2}}
69 layer{name: "i4a/1x1" type: "Convolution" bottom: "pool3/3x3_s2" top: "i4a/1x1"
  convolution_param {num_output: 192 kernel_size: 1}}
layer{name: "i4a/relu_1x1" type: "ReLU" bottom: "i4a/1x1" top: "i4a/1x1"}
layer{name: "i4a/3x3_red" type: "Convolution" bottom: "pool3/3x3_s2"
  top: "i4a/3x3_red" convolution_param {num_output: 96 kernel_size: 1}}
74 layer{name: "i4a/relu_3x3_red" type: "ReLU" bottom: "i4a/3x3_red" top: "i4a/3x3_red"}
layer{name: "i4a/3x3" type: "Convolution" bottom: "i4a/3x3_red" top: "i4a/3x3"
  convolution_param {num_output: 208 pad: 1 kernel_size: 3}}
layer{name: "i4a/relu_3x3" type: "ReLU" bottom: "i4a/3x3" top: "i4a/3x3"}
layer{name: "i4a/5x5_red" type: "Convolution" bottom: "pool3/3x3_s2"
  top: "i4a/5x5_red" convolution_param {num_output: 16 kernel_size: 1}}
79 layer{name: "i4a/relu_5x5_red" type: "ReLU" bottom: "i4a/5x5_red" top: "i4a/5x5_red"}
layer{name: "i4a/5x5" type: "Convolution" bottom: "i4a/5x5_red" top: "i4a/5x5"
  convolution_param {num_output: 48 pad: 2 kernel_size: 5}}
layer{name: "i4a/relu_5x5" type: "ReLU" bottom: "i4a/5x5" top: "i4a/5x5"}
84 layer{name: "i4a/pool" type: "Pooling" bottom: "pool3/3x3_s2" top: "i4a/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i4a/proj" type: "Convolution" bottom: "i4a/pool" top: "i4a/proj"
  convolution_param {num_output: 64 kernel_size: 1}}
layer{name: "i4a/relu_proj" type: "ReLU" bottom: "i4a/proj" top: "i4a/proj"}
89 layer{name: "i4a/output" type: "Concat"
  bottom: "i4a/1x1" bottom: "i4a/3x3" bottom: "i4a/5x5" bottom: "i4a/proj"
  top: "i4a/output"}
layer{name: "i4b/1x1" type: "Convolution" bottom: "i4a/output" top: "i4b/1x1"
  convolution_param {num_output: 160 kernel_size: 1}}
94 layer{name: "i4b/relu_1x1" type: "ReLU" bottom: "i4b/1x1" top: "i4b/1x1"}
layer{name: "i4b/3x3_red" type: "Convolution" bottom: "i4a/output" top: "i4b/3x3_red"
  convolution_param {num_output: 112 kernel_size: 1}}
layer{name: "i4b/relu_3x3_red" type: "ReLU" bottom: "i4b/3x3_red" top: "i4b/3x3_red"}
layer{name: "i4b/3x3" type: "Convolution" bottom: "i4b/3x3_red" top: "i4b/3x3"
  convolution_param {num_output: 224 pad: 1 kernel_size: 3}}
99 layer{name: "i4b/relu_3x3" type: "ReLU" bottom: "i4b/3x3" top: "i4b/3x3"}
layer{name: "i4b/5x5_red" type: "Convolution" bottom: "i4a/output" top: "i4b/5x5_red"
  convolution_param {num_output: 24 kernel_size: 1}}
layer{name: "i4b/relu_5x5_red" type: "ReLU" bottom: "i4b/5x5_red" top: "i4b/5x5_red"}
104 layer{name: "i4b/5x5" type: "Convolution" bottom: "i4b/5x5_red" top: "i4b/5x5"
  convolution_param {num_output: 64 pad: 2 kernel_size: 5}}
layer{name: "i4b/relu_5x5" type: "ReLU" bottom: "i4b/5x5" top: "i4b/5x5"}
layer{name: "i4b/pool" type: "Pooling" bottom: "i4a/output" top: "i4b/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
109 layer{name: "i4b/proj" type: "Convolution" bottom: "i4b/pool" top: "i4b/proj"
  convolution_param {num_output: 64 kernel_size: 1}}
layer{name: "i4b/relu_proj" type: "ReLU" bottom: "i4b/proj" top: "i4b/proj"}
layer{name: "i4b/output" type: "Concat"
  bottom: "i4b/1x1" bottom: "i4b/3x3" bottom: "i4b/5x5" bottom: "i4b/proj"
  top: "i4b/output"}
114 layer{name: "i4c/1x1" type: "Convolution" bottom: "i4b/output" top: "i4c/1x1"
  convolution_param {num_output: 128 kernel_size: 1}}
layer{name: "i4c/relu_1x1" type: "ReLU" bottom: "i4c/1x1" top: "i4c/1x1"}
layer{name: "i4c/3x3_red" type: "Convolution" bottom: "i4b/output" top: "i4c/3x3_red"
  convolution_param {num_output: 128 kernel_size: 1}}
119 layer{name: "i4c/relu_3x3_red" type: "ReLU" bottom: "i4c/3x3_red" top: "i4c/3x3_red"}
layer{name: "i4c/3x3" type: "Convolution" bottom: "i4c/3x3_red" top: "i4c/3x3"
  convolution_param {num_output: 256 pad: 1 kernel_size: 3}}
layer{name: "i4c/relu_3x3" type: "ReLU" bottom: "i4c/3x3" top: "i4c/3x3"}
124 layer{name: "i4c/5x5_red" type: "Convolution" bottom: "i4b/output" top: "i4c/5x5_red"
  convolution_param {num_output: 24 kernel_size: 1}}
layer{name: "i4c/relu_5x5_red" type: "ReLU" bottom: "i4c/5x5_red" top: "i4c/5x5_red"}
layer{name: "i4c/5x5" type: "Convolution" bottom: "i4c/5x5_red" top: "i4c/5x5"
  convolution_param {num_output: 64 pad: 2 kernel_size: 5}}
129 layer{name: "i4c/relu_5x5" type: "ReLU" bottom: "i4c/5x5" top: "i4c/5x5"}

```

```

layer{name: "i4c/pool" type: "Pooling" bottom: "i4b/output" top: "i4c/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i4c/proj" type: "Convolution" bottom: "i4c/pool" top: "i4c/proj"
  convolution_param {num_output: 64 kernel_size: 1}}
134 layer{name: "i4c/relu_proj" type: "ReLU" bottom: "i4c/proj" top: "i4c/proj"}
layer{name: "i4c/output" type: "Concat"
  bottom: "i4c/1x1" bottom: "i4c/3x3" bottom: "i4c/5x5" bottom: "i4c/proj"
  top: "i4c/output"}
layer{name: "i4d/1x1" type: "Convolution" bottom: "i4c/output" top: "i4d/1x1"
139 convolution_param {num_output: 112 kernel_size: 1}}
layer{name: "i4d/relu_1x1" type: "ReLU" bottom: "i4d/1x1" top: "i4d/1x1"}
layer{name: "i4d/3x3_red" type: "Convolution" bottom: "i4c/output" top: "i4d/3x3_red"
  convolution_param {num_output: 144 kernel_size: 1}}
144 layer{name: "i4d/relu_3x3_red" type: "ReLU" bottom: "i4d/3x3_red" top: "i4d/3x3_red"}
layer{name: "i4d/3x3" type: "Convolution" bottom: "i4d/3x3_red" top: "i4d/3x3"
  convolution_param {num_output: 288 pad: 1 kernel_size: 3}}
layer{name: "i4d/relu_3x3" type: "ReLU" bottom: "i4d/3x3" top: "i4d/3x3"}
layer{name: "i4d/5x5_red" type: "Convolution" bottom: "i4c/output" top: "i4d/5x5_red"
  convolution_param {num_output: 32 kernel_size: 1}}
149 layer{name: "i4d/relu_5x5_red" type: "ReLU" bottom: "i4d/5x5_red" top: "i4d/5x5_red"}
layer{name: "i4d/5x5" type: "Convolution" bottom: "i4d/5x5_red" top: "i4d/5x5"
  convolution_param {num_output: 64 pad: 2 kernel_size: 5}}
layer{name: "i4d/relu_5x5" type: "ReLU" bottom: "i4d/5x5" top: "i4d/5x5"}
154 layer{name: "i4d/pool" type: "Pooling" bottom: "i4c/output" top: "i4d/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i4d/proj" type: "Convolution" bottom: "i4d/pool" top: "i4d/proj"
  convolution_param {num_output: 64 kernel_size: 1}}
layer{name: "i4d/relu_proj" type: "ReLU" bottom: "i4d/proj" top: "i4d/proj"}
159 layer{name: "i4d/output" type: "Concat"
  bottom: "i4d/1x1" bottom: "i4d/3x3" bottom: "i4d/5x5" bottom: "i4d/proj"
  top: "i4d/output"}
layer{name: "i4e/1x1" type: "Convolution" bottom: "i4d/output" top: "i4e/1x1"
  convolution_param {num_output: 256 kernel_size: 1}}
layer{name: "i4e/relu_1x1" type: "ReLU" bottom: "i4e/1x1" top: "i4e/1x1"}
164 layer{name: "i4e/3x3_red" type: "Convolution" bottom: "i4d/output" top: "i4e/3x3_red"
  convolution_param {num_output: 160 kernel_size: 1}}
layer{name: "i4e/relu_3x3_red" type: "ReLU" bottom: "i4e/3x3_red" top: "i4e/3x3_red"}
layer{name: "i4e/3x3" type: "Convolution" bottom: "i4e/3x3_red" top: "i4e/3x3"
  convolution_param {num_output: 320 pad: 1 kernel_size: 3}}
169 layer{name: "i4e/relu_3x3" type: "ReLU" bottom: "i4e/3x3" top: "i4e/3x3"}
layer{name: "i4e/5x5_red" type: "Convolution" bottom: "i4d/output" top: "i4e/5x5_red"
  convolution_param {num_output: 32 kernel_size: 1}}
layer{name: "i4e/relu_5x5_red" type: "ReLU" bottom: "i4e/5x5_red" top: "i4e/5x5_red"}
174 layer{name: "i4e/5x5" type: "Convolution" bottom: "i4e/5x5_red" top: "i4e/5x5"
  convolution_param {num_output: 128 pad: 2 kernel_size: 5}}
layer{name: "i4e/relu_5x5" type: "ReLU" bottom: "i4e/5x5" top: "i4e/5x5"}
layer{name: "i4e/pool" type: "Pooling" bottom: "i4d/output" top: "i4e/pool"
  pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
179 layer{name: "i4e/proj" type: "Convolution" bottom: "i4e/pool" top: "i4e/proj"
  convolution_param {num_output: 128 kernel_size: 1}}
layer{name: "i4e/relu_proj" type: "ReLU" bottom: "i4e/proj" top: "i4e/proj"}
layer{name: "i4e/output" type: "Concat"
  bottom: "i4e/1x1" bottom: "i4e/3x3" bottom: "i4e/5x5" bottom: "i4e/proj"
  top: "i4e/output"}
184 layer{name: "pool4/3x3_s2" type: "Pooling" bottom: "i4e/output" top: "pool4/3x3_s2"
  pooling_param {pool: MAX kernel_size: 3 stride: 2}}
layer{name: "i5a/1x1" type: "Convolution" bottom: "pool4/3x3_s2" top: "i5a/1x1"
  convolution_param {num_output: 256 kernel_size: 1}}
layer{name: "i5a/relu_1x1" type: "ReLU" bottom: "i5a/1x1" top: "i5a/1x1"}
189 layer{name: "i5a/3x3_red" type: "Convolution" bottom: "pool4/3x3_s2"
  top: "i5a/3x3_red" convolution_param {num_output: 160 kernel_size: 1}}
layer{name: "i5a/relu_3x3_red" type: "ReLU" bottom: "i5a/3x3_red" top: "i5a/3x3_red"}
layer{name: "i5a/3x3" type: "Convolution" bottom: "i5a/3x3_red" top: "i5a/3x3"
  convolution_param {num_output: 320 pad: 1 kernel_size: 3}}
194 layer{name: "i5a/relu_3x3" type: "ReLU" bottom: "i5a/3x3" top: "i5a/3x3"}
layer{name: "i5a/5x5_red" type: "Convolution" bottom: "pool4/3x3_s2"

```

```

    top: "i5a/5x5_red" convolution_param {num_output: 32 kernel_size: 1}}
layer{name: "i5a/relu_5x5_red" type: "ReLU" bottom: "i5a/5x5_red" top: "i5a/5x5_red"}
layer{name: "i5a/5x5" type: "Convolution" bottom: "i5a/5x5_red" top: "i5a/5x5"}
199 convolution_param {num_output: 128 pad: 2 kernel_size: 5}}
layer{name: "i5a/relu_5x5" type: "ReLU" bottom: "i5a/5x5" top: "i5a/5x5"}
layer{name: "i5a/pool" type: "Pooling" bottom: "pool4/3x3_s2" top: "i5a/pool"}
    pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i5a/proj" type: "Convolution" bottom: "i5a/pool" top: "i5a/proj"}
204 convolution_param {num_output: 128 kernel_size: 1}}
layer{name: "i5a/relu_proj" type: "ReLU" bottom: "i5a/proj" top: "i5a/proj"}
layer{name: "i5a/output" type: "Concat" bottom: "i5a/1x1" bottom: "i5a/3x3"
    bottom: "i5a/5x5" bottom: "i5a/proj" top: "i5a/output"}
layer{name: "i5b/1x1" type: "Convolution" bottom: "i5a/output" top: "i5b/1x1"}
209 convolution_param {num_output: 384 kernel_size: 1}}
layer{name: "i5b/relu_1x1" type: "ReLU" bottom: "i5b/1x1" top: "i5b/1x1"}
layer{name: "i5b/3x3_red" type: "Convolution" bottom: "i5a/output" top: "i5b/3x3_red"}
    convolution_param {num_output: 192 kernel_size: 1}}
layer{name: "i5b/relu_3x3_red" type: "ReLU" bottom: "i5b/3x3_red" top: "i5b/3x3_red"}
214 layer{name: "i5b/3x3" type: "Convolution" bottom: "i5b/3x3_red" top: "i5b/3x3"}
    convolution_param {num_output: 384 pad: 1 kernel_size: 3}}
layer{name: "i5b/relu_3x3" type: "ReLU" bottom: "i5b/3x3" top: "i5b/3x3"}
layer{name: "i5b/5x5_red" type: "Convolution" bottom: "i5a/output" top: "i5b/5x5_red"}
    convolution_param {num_output: 48 kernel_size: 1}}
219 layer{name: "i5b/relu_5x5_red" type: "ReLU" bottom: "i5b/5x5_red" top: "i5b/5x5_red"}
layer{name: "i5b/5x5" type: "Convolution" bottom: "i5b/5x5_red" top: "i5b/5x5"}
    convolution_param {num_output: 128 pad: 2 kernel_size: 5}}
layer{name: "i5b/relu_5x5" type: "ReLU" bottom: "i5b/5x5" top: "i5b/5x5"}
layer{name: "i5b/pool" type: "Pooling" bottom: "i5a/output" top: "i5b/pool"}
224 pooling_param {pool: MAX kernel_size: 3 stride: 1 pad: 1}}
layer{name: "i5b/proj" type: "Convolution" bottom: "i5b/pool" top: "i5b/proj"}
    convolution_param {num_output: 128 kernel_size: 1}}
layer{name: "i5b/relu_proj" type: "ReLU" bottom: "i5b/proj" top: "i5b/proj"}
layer{name: "i5b/output" type: "Concat" bottom: "i5b/1x1" bottom: "i5b/3x3"
229 bottom: "i5b/5x5" bottom: "i5b/proj" top: "i5b/output"}
layer{name: "score37" type: "Convolution" bottom: "i5b/output" top: "score37"}
    convolution_param {num_output: 38 kernel_size: 1 stride: 1 pad: 0}}
layer{name: "upscore2" type: "Deconvolution" bottom: "score37" top: "upscore2"}
    convolution_param {num_output: 38 kernel_size: 4 stride: 2 pad: 1}}
234 layer{name: "score-p3" type: "Convolution" bottom: "pool3/3x3_s2" top: "score-p3"}
    convolution_param {num_output: 38 kernel_size: 1}}
layer{name: "fuse" type: "Eltwise" bottom: "upscore2" bottom: "score-p3"
    top: "score-fused" eltwise_param {operation: SUM}}
layer{name: "us-fused-16" type: "Deconvolution" bottom: "score-fused" top: "score3"}
239 convolution_param {num_output: 38 kernel_size: 4 stride: 2 pad: 1}}
layer{name: "score-p2" type: "Convolution" bottom: "pool2/3x3_s2" top: "score-p2"}
    convolution_param {num_output: 38 kernel_size: 1}}
layer{name: "fuse" type: "Eltwise" bottom: "score3" bottom: "score-p2"
244 top: "score-final" eltwise_param {operation: SUM}}
layer{name: "upsample" type: "Deconvolution" bottom: "score-final" top: "bigscore"}
    convolution_param {num_output: 38 kernel_size: 16 stride: 8}}
layer{name: 'crop' type: 'CenterCrop' bottom: 'bigscore' top: 'score'}
    transform_param {crop-size: 256}}

```

Network/googlenet-8s.prototxt

Appendix C

Segmentation Results

Class	Goog- LeNet	VGG16	VGG16 +CRF	Class	Goog- LeNet	VGG16	VGG16 +CRF
snow	96.11	96.89	97.51	agricultural	30.62	38.31	40.77
grassland	74.49	80.32	81.51	industrial	43.27	58.91	62.43
lawn	40.50	57.95	59.75	railroad	87.50	93.48	94.40
shrub	39.51	41.49	42.20	cartway	27.42	42.88	45.88
tree	92.39	92.53	92.74	sidewalk	43.44	52.91	56.36
orchard	1.14	2.14	2.46	macadam	34.19	34.83	40.40
field	91.89	94.93	95.63	driveway	35.44	48.13	54.49
dirt	58.80	59.37	61.83	secondary	59.17	60.85	63.32
debris	73.55	75.73	78.56	avenue	85.99	88.72	91.59
rock	88.93	91.43	92.10	highway	76.73	78.81	81.05
paved	43.67	56.29	58.66	freeway	89.33	93.68	93.22
parking	81.68	83.35	83.42	ramp	76.99	86.47	85.42
pool	28.70	25.92	24.86	curbs	49.03	57.08	51.98
river	80.79	80.67	80.52	pitlane	64.48	74.55	73.47
water	0.00	0.00	—	track	83.94	85.00	86.17
shed	20.44	16.95	16.55	barrier	52.44	64.49	67.54
detached	41.54	35.57	35.85	apron	71.44	56.77	56.29
residential	81.06	82.54	84.51	grandstand	94.34	91.05	88.47
highrise	87.14	78.73	84.78	gravel	97.56	97.29	98.52
mean IoU	61.20	64.66	67.71				

Table C.1: Segmentation performance (IoU [%]) of networks with a stride of 8 pixels as measured on the validation data set. VGG16+CRF contains an additional - albeit untrained - CRF layer but is otherwise identical to VGG16.

Appendix D

Third Party Software

This thesis has been made possible by the availability of open source machine learning and computer vision software. Table D.1 shows a non-exclusive list of third-party software components which were used to create the tools that are a part of this work.

Software	Version	Url
Anaconda	2.7	https://www.continuum.io/downloads
Boost	1.56.0	http://www.boost.org/
Caffe	customized	uses code from:
	- Caffe-future	https://github.com/longjon/caffe
	- Windows port	https://github.com/willyd/caffe
	- CRFasRNN	https://github.com/torrvision/crfasrnn.git
Ceres	1.11.0	http://ceres-solver.org/
CUDA	7.5	http://www.nvidia.com
CUDNN	3.0	http://www.nvidia.com
DenseCRF	-	http://graphics.stanford.edu/projects/drfs/
Eigen	b30b87236a1b	http://eigen.tuxfamily.org
gflags	9db828953a	https://github.com/gflags/gflags.git
glib	2.42.2	http://www.gtk.org/
glog	f46e0745a8	https://github.com/google/glog.git
HDF5	1.8.16	http://hdf5.net/
IPython	-	https://ipython.org/
LevelDB	915d663292	https://github.com/google/leveldb
LMDB	58ad1dd757	https://github.com/LMDB/lmdb.git
OpenBLAS	0.2.14	http://www.openblas.net/
OpenCV	2.4.9 & 3.1.0	http://opencv.org/
protobuf	2.6.1	https://github.com/google/protobuf
snappy	1.1.1.8	https://snappy.angeloflogic.com/

Table D.1: Third party software components.