Bachelor thesis

Estimating the Parking Capacity in Cities Using Aerial Images

Matouš Dzivjak

Supervisor: Ing. David Fiedler

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
May, 2019

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Dzivjak Matouš** Personal ID number: **469831**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Estimating the Parking Capacity in Cities Using Aerial Images

Bachelor's thesis title in Czech:

Odhad parkovací kapacity ve městech s použitím leteckých snímků

Guidelines:

1. Explore the possibilities of information extraction from aerial images with a focus on the traffic field. Additionally, study the state of the art methods for parking capacity estimation from other data sources.
2. Create a short review of the methods for parking capacity estimation and information extraction from aerial images in similar research areas.
3. Design the method for parking capacity estimation using aerial images.
4. Implement the designed method and test its accuracy on some publicly available dataset.

Bibliography / sources:

[1] H. Tayara, K. Gil Soo and K. T. Chong, "Vehicle Detection and Counting in High-Resolution Aerial Images Using Convolutional Regression Neural Network," in IEEE Access, vol. 6, 2018.
[2] Q. Wu, C. Huang, S. Wang, W. Chiu and T. Chen, "Robust Parking Space Detection Considering Inter-Space Correlation," 2007 IEEE International Conference on Multimedia and Expo, Beijing, 2007.
[3] K. Liu and G. Mattyus, "Fast Multiclass Vehicle Detection on Aerial Images," in IEEE Geoscience and Remote Sensing Letters, vol. 12, no. 9, pp. 1938-1942, Sept. 2015.
[4] Volodymyr Mnih, Geoffrey E Hinton, "Learning to detect roads in high-resolution aerial images" in Computer Vision-ECCV 2010, Springer, pp. 210-223, 2010.
[5] T. Nathan Mundhenk, Goran Konjevod, Wesam A. Sakla, Kofi Boakye: A Large Contextual Dataset for Classification, Detection and Counting of Cars with Deep Learning. CoRR abs/1609.04453 (2016).

Name and workplace of bachelor's thesis supervisor:

Ing. David Fiedler, Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **15.01.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Ing. David Fiedler Supervisor's signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

Prague, date..... ..... signature

Acknowledgements:

I want to thank Ing. David Fiedler for his valuable advice, factual comments, and helpfulness in the bachelor thesis consultations and Martin Koryťák, for his invaluable insight into the process of training neural networks. Furthermore, I would like to thank my family and girlfriend, who supported me throughout my studies and last but not least, the Czech Technical University for the provided education and knowledge.

Abstract

Estimating parking capacity from aerial images can be used for demographic modeling, customer flow analysis, and city planning. This topic has a lot of applications in both the commercial and government sectors. To accurately estimate parking capacity, we need to count both occupied and unoccupied parking spaces on streets and properties as well as in parking lots. We explore previous research that focuses mainly on parking lots capacity estimation and thus does not apply to many European countries where parking places exist alongside roads and in front of houses. We design and implement a method for parking capacity estimation from aerial images using neural networks focusing on the on-street parking. We evaluate this method on data from Brno and Prague, Czech Republic. Our parking capacity estimation method for individual streets achieves accuracy of 6.83%.

Key words: detection in aerial images, deep learning, parking capacity estimation, aerial images, parking places detection

Abstrakt

Většina velkých měst je v dnešná době pokryta satelitními snímky ve vysoké kvalitě, které se dají využít k analýze rozličných vlastností města. Jedna z možností je ze satelitních snímků odhadovat parkovací kapacitu. V této práci nejdříve ve stručnosti ukážeme předchozí výzkum na tomto poli, který pochází především z USA a zabývá se hlavně kapacitou parkovišť, v tomto je charakteristika evropských měst jiná, většina parkovacích míst se nachází na krajích silnic a tak tyto výzkumy lze využít pouze omezeně. Dále navrhneme a naimplementujeme metodu pro odhad parkovací kapacity z leteckých snímků pomocí neuronových sítí a evalujeme tuto metodu na datech z České republiky.

Klíčová slova: detekce ze satelitních snímků, hluboké učení, odhad parkovací kapacity, letecké snímky, detekce parkovacích míst

Contents

List of Figures

List of Tables

Abbreviations:

GSD ..... Ground Sample Distance

YOLO ..... You Only Look Once

COWC ..... Cars Overhead With Context

YOLT ..... You Only Look Twice

SIMRDWN ... Satellite Imagery Multiscale Rapid Detection with Windowed Networks

GIS ..... Geographic Information System

mAP ..... mean Average Precision

FPN ..... Feature Pyramid Network

OSM ..... Open Street Maps

Chapter 1

Introduction

Parking capacity estimation from aerial images has multiple use cases in a humanitarian and commercial domain. Urbanization and city growth is associated with a loss of green space and wildlife habitat.[18] Better knowledge of parking capacity in individual city areas would allow for better planning of city growth and minimization of parking lot areas necessary for smooth transportation and parking. Furthermore, the solution would also make it possible to locate the most crowded places and allow urban planners to address the shortcomings and in the future provide vital information for autonomous fleet routing.

Searching for parking spots also imposes a significant economic burden with drivers in the U.S., Germany, and the U.K. wasting on average 34 hours a year when searching for a parking place at an estimated cost of €136.4 billion.[1] Accurate knowledge of parking capacity thus provides vital information not only for the government but for the commercial sector as well.

Parking capacity detection would allow people to find parking during high traffic hours easily. In less populated areas, cities could easily plan where to build the next parking lot to maximize its usage and minimize the distance from areas that need parking the most. In commercial domain parking lot owners could easily plan new parking lots and adjust their costs to match the demand in the given area.

Existent business problems that may benefit from this research include demographic modeling and customer flow analysis, which is useful mainly for those in the retail sector, festival and conference organizers, and governments. Parking places and parked cars detection can be used to monitor peak business hours by counting the number of parked vehicles at a given time and also, if accurate enough, to extrapolate useful customer information such as marital status, income, and even political inclination classifying the types of vehicles the people own.[42][29]

In future GPS navigation could automatically detect free parking places and navigate you there. Finding a free parking spot is a tedious process, especially during working hours. Studies of cruising in busy downtowns have found out that it takes around 7 minutes to find a vacant space, and that 35 percent of the traffic

is cruising for parking which sums up to over 30 extra kilometers each year to find curb parking space.[35] Drivers are also willing to pay more for on-street parking if it is closer to their destination making the subject of parking capacity estimation interesting for a commercial sphere as well.[22]

A few big problems arise when working with satellite images. One is that vehicles in the imagery are usually tiny; around 20px in size, which makes it harder to extract features and abstract on top of low-resolution objects. Next, the objects can be rotated in any way around the unit circle. Input images are huge, ranging in size from tens to thousands of megapixels. Loading them and working with them as a whole in memory is often very complicated and as such other workarounds need to be used, such as splitting images into smaller patches or intentionally lowering their resolution. Last but not least, there is a relative scarcity of data, which makes training neural networks on aerial images hard. Many efforts in recent years try to improve this issue and provide free and open source datasets; example be Imagenet project that provides one of the biggest and well-known datasets for image recognition and classification.

On the other side, satellite images have some real properties as well. The scale of the object (in pixels in meters) is usually known in advance. The observation angle stays mostly the same, and in orthophoto images, there is not an observation angle all. These properties help with the configuration of neural networks and make detection tasks easier.

1.1 Targets of this Thesis

The target of this thesis is to create a complete method for parking capacity estimation over a given area with the main focus on on-street parking. We focus on on-street parking because it is the least researched part of this problem and provides the most prominent space for improvement. We suggest a few different approaches to this problem and discuss their benefits and shortcomings. We show the reasoning behind these methods and how they build on previous work that is described in the same chapter. We describe the state of the art methods for vehicle detection, parking lot parking spaces detection, and capacity estimation and object counting in images. Then we implement a method for parking capacity estimation and finally test our approach on the city of Prague and estimate its accuracy. Throughout this thesis, we introduce the topic of object detection in general, machine learning and methods for working with aerial images.

Chapter 2

Problem Analysis

The problem of object detection and counting in aerial images has gained its popularity in recent years thanks to better availability of aerial images and improvement in their quality. Still, scarcity of well-annotated data sets of objects in aerial scenes complicates the efforts, and this is especially true in our case, where the dataset of parked cars does not exist at all. It is the same when it comes to research conducted in this field. We found many papers dealing with the problem of parking spot detection (Gou Koutaki and Uchimura [14], Young-Woo Seo and Urmson [41], Yu [42]) but all of them assume that the parking places are part of a parking lot. These facts drove our decision in the method design as we explain later in this chapter 2.5.

Due to the lack of research on parking places detection and counting, we decided to divide the task into sub-tasks that were either entirely or partially solved and which could further serve as a basis for our method. We decided that the following methods could serve as building blocks for our method.

2.1 Object Counting Using Neural Networks

First, we looked into object counting in images, which would be the easiest and most straightforward path to take as one method would do all the work.. . .

Santi Seguí and Vitriá [34] explored 2 different approaches to object counting in images in his paper. First, training an object detector, and second, training an object counter. As written in the paper, for the first method to work, we need a broad set of objects examples with correct labels and localization. In the second case, the only thing that is needed is the number of objects instances in the image. They trained CNN to count handwritten digits in the image. Using the dataset of handwritten digits[1] to synthesize the training images, they achieved accuracy of 93.8%. They also tried their method on the task of counting pedestrians in surveillance cameras. On a dataset of 200,000 synthetic images base network achieved 0.74 mean absolute error and 1.12 mean square error. These results sound promising; what made us unsure

[1]MNIST dataset, http://yann.lecun.com/exdb/mnist/

about using this method is the distinction between parked and driving cars. The difference between these two is small and mainly in details and although the network in this paper learned the distinction between odd and even numbers it needed a tremendous amount of data, extensive training, and the training images were several times smaller than the satellite images we will be using.

The process of counting objects using CNNs is also theme of work by Wang et al. [38] and Boominathan, Kruthiventi, and Babu [3]. Both teams developed methods for densely packed objects counting; in both cases, people in the crowd. Accuracy in both papers was low, and the task of counting densely packed objects proved to be harder than expected. This method could be used for estimating the count of parked cars in parking lots where the vehicles are densely packed. However, the accuracy in the mentioned papers was low, and we believe that there is a little chance of significant improvement. Furthermore, this would solve only part of the problem of parking capacity estimation, and we would need to use more methods to count on-street parking cars and parking places as well.

2.2 Car Detection

Splitting the task into individual sub-tasks brings us to the problem of car detection in general. Many new methods for processing of large images emerged in recent years, and new hardware makes this processing fast and affordable. Modern detection frameworks work at an astonishing and can detect objects on an area of one square kilometer per second[9] compared to approximately 1 minute for a square kilometer with lower resolution and accuracy in 2001.[10][28] These speeds could be used for near real-time surveillance and monitoring as well as make any reconnaissance faster and cheaper thus more lucrative for commercial use. The current state of the art CNNs can not only classify the images as a whole but also find multiple objects and their bounding boxes. We discuss a few of them here, namely RetinaNet 3.2.2.2 and YOLOv3 3.2.2.4.

Before the emergence of deep learning and neural networks, hand-crafted features combined with a classifier were the mostly adopted ideas to detect cars in aerial images. Even so, the hand-crafted features cannot generalize, and the resulting classifiers need to be modified to adapt to new features.[40] Methods using hand-crafted features can be divided into two groups, appearance-based and feature-based.

Appearance-based methods for object detection use example images (templates) of the object to perform recognition and detection. As the object in different images might have different lighting, viewing angle and size and shape, one template usually is not enough and the set of templates that the detection framework checks against quickly grows. In appearance-based methods belong methods such as edge matching, divide-and-conquer search, histograms of receptive field responses, etc.[37]

Feature-based object detection includes methods that compute abstraction of the image and make a local decision at every point of the image, whether there is a feature or not. Features can be edges, corner (interest points), and blobs (regions of

interest points). Once the features are detected, they are extracted to the so-called feature vector. Resulting feature vector is then compared to the feature vector of the template and if a certain threshold of matching features is met, a positive match is returned.

Feature-based methods were used in the first attempts to detect cars in aerial images, which come from the start of this millennium when images were only in greyscale, and the resolution was low. Zhao and Nevatia [43] firstly did a psychological experiment on human subjects to determine what features we use to detect cars in aerial images. They found out that the most important is car boundary, windshield, and the shadow of the car. They used these features and tried to detect them in various positions in the image to identify a car. As mentioned before, this approach worked but needed to be modified to new features and tasks.[28]

In recent years, the detection slowly evolved to the usage of neural networks. Yang et al. [40] worked with more modern approach. They modified R-CNN (described in 3.2.2.3), a particular type of neural network, using ResNet architecture as the backbone of their method (4.2.1.2). Their network learned on images with 10 cm GSD and achieved a recall rate of 89.44%, precision rate 64.61% and F1 score 0.75.

Paper by Etten [10] presents The Satellite Imagery Multiscale Rapid Detection with Windowed Networks (SIMRDWN), the current state of the art detection network. This network is an upgrade of the YOLO detection framework (3.2.2.4) tuned for aerial and satellite images. It follows the Tensorflow object detection API and builds on top of Darknet[2]. SIMRDWN further handles large splitting images into smaller chips (416 pixels by default) as well as combining these chips back together after detecting target objects.

One article which explores the problem of object detection from aerial images from all the perspectives was recently released by Alouini [2]. They participated in the Kaggle's Airbus Challenge where the task was to detect ships in aerial images.

Etten [8] also has a great series of articles about aerial object detection, data preprocessing for the detection frameworks and also about the YOLO and SIMRDWN frameworks 3. These articles first describe the detection of boats and airplanes, then detection of cars, and finally analyze in detail the effect of resolution on deep neural network image classification accuracy. Codes from these articles are also publicly available on GitHub[3].[8]

2.3 Parking Spots and Parking Lots Detection

While there are many papers focused on detecting and tracking occupation of parking places, most of them work with images from cameras on parking lots (Jordan Cazamias [19]) or can be used on parking lots only. That is

[2]Darknet is an open source neural network framework written in C and CUDA. https://pjreddie.com/darknet/

[3]https://github.com/avanetten/yolt

understandable for countries like the USA where most vehicles actually park in car parks, but not for European countries where cars often park along the roads or on parking places unmarked by lines. Our method will need to detect parking spots in parking lots as well as alongside the road or in the driveway of a house. Throughout this paper, we will focus mainly on the on-street parking since the problem of parking lot detection has already solutions with sufficient quality.[1]

An interesting approach is shown in Gou Koutaki and Uchimura [14], and although it aims mainly for parking lots, many facts and steps can be applied to our problem as well. Their team took a closer look at the geometric properties of parking lots. They created a pipeline in which they pre-process the data to make the cars better visible, detect cars and parking spaces, and then merge this data. Parking columns and parking rows are then estimated based on the various vehicle and parking space candidates extracted in the previous step of the pipeline. They are then grouped based on the geometric parking lot model, and the whole schema of the parking lot is created. Even if methods from this paper could not be used for onstreet parking places detection, they can prove useful for parking capacity estimation.

Gou Koutaki and Uchimura [14] use a DSM (Digital Surface Model). DSMs can be obtained through UAVs or drones with infrared cameras[4] or laser scanners. For vehicle detection, they presume a fixed scale of all the images and use a Haar-like detector and AdaBoost. A Haar-like detector considers adjacent regions at a specific area in a detection window, sums up the pixel in each region and calculates the difference between these sums; this creates *feature* of the given region. AdaBoost then takes care of classifying and detecting cars based on these features by assigning each feature its weight in the process of training; the result of the classification is a weighted sum of the features. For parking space detection, they used template matching (2.2) with manually created template models for white and yellow lines. They further modified the templates by artificially adding shadow to the templates to adjust for parking spaces with missing bounding lines. In the last step, the grouping of previously detected parking spaces and cars is applied. They group parking places based on the distance of their centroids and add a margin to account for imperfect detection methods; other methods are used as well to handle the imperfections in detections. This complicated method achieved outstanding results and would be a perfect fit for parking lot parking capacity estimation.

Young-Woo Seo and Urmson [41] describes another approach that uses multiple image processing steps such as line extraction, line clustering, and (parking) block prediction. Then parking extrapolation and interpolation were used to extend the detected areas to the whole parking lot. This approach again works with the assumption that the parking places are clearly marked by lines, although with worse results then Gou Koutaki and Uchimura [14].

All the mentioned papers and research achieved good results and accuracy, but they all work only with parking lots and omit the parking places on streets alongside

[4]They used eBee drone by senseFly

roads. This is not a big deal in the U.S.A. where the majority of the parking places are in parking lots, but in Europe, where the parking is primarily on streets and properties these methods would not work.

2.4 Data

Data are as important as the methods that we use.[4] For our research, aerial images and street annotations and properties are the most important data we need to obtain. We obtain geographical data of the road network over areas of interest from *OpenStreetMap* (OSM)[5], a collaborative editable map of the world. OpenStreetMap works similarly to Wikipedia but for maps, it supports public contributions, and the content is freely available. This data contains the whole street network and street types and names.

The images that we work with are mostly adjusted to being orthographic. *Orthophoto* or *orthographic images* is satellite imagery that is geometrically corrected so that the scale of objects in the image is uniform. The exact distance can be measured in such images, and there is no perspective in these pictures. Difference between orthographic images and image with perspective is shown in figure 2.1.

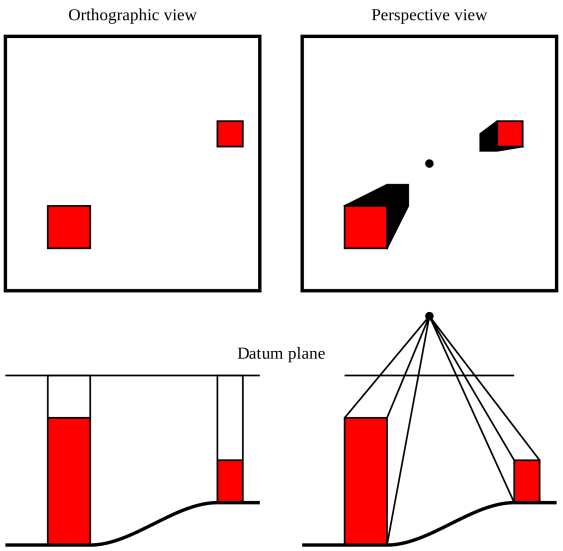


Figure 2.1: Orthographic views project at a right angle to the datum plane. Perspective views project from the surface onto the datum plane from a fixed location.

Before we describe datasets of satellite and aerial images, it is important to explain the GSD and OpenAerialMap platform. *GSD* (Ground Sample Distance) is

[5]https://www.openstreetmap.org/#map=18/49.57747/14.53958

the distance between pixels measured on the ground level. For example, images with 30cm GSD which we use in this paper mean, that the adjacent pixels in the image are 30cm apart on the ground. *OpenAerialMap* is a service that provides open access to a commons of openly licensed map layers and imagery. It is open to the public, and anyone can contribute and access data on this service.

One very recent and popular dataset is a dataset of Zanzibar satellite images collected via the Zanzibar Mapping Initiative[6]. It is available on OpenAerialMap with the resolution of 7.5 GSD. This dataset has no annotations for cars, only for buildings and roads. These labels could be manually created to obtain a high-resolution dataset of a large area with many cars. Although the higher resolution might help with car detection and would be more precise, it is proven that the difference is insignificant for our needs. F1 score degrades only by 5% as the resolution changes from 0.15m to 0.60m GSD. We will have this dataset as a backup option to either visually estimate the accuracy of our method or to add annotations and use it to extend our training dataset.[11]

Another large dataset is DOTA (A Large-scale Dataset for Object Detection in Aerial Images). Xia et al. [39] describes the DOTA dataset, which is a large dataset of aerial images with annotated objects such as cars, ships, planes, soccer fields, and more (15 categories in total). This dataset has not only bounding boxes but also their angle from the horizontal direction of the standard bounding box which proves to be more difficult to detect (they compare results with the YOLOv2 framework and achieve half the accuracy on cars compared to bounding boxes without angle). In contrast with COWC it contains 2 806 large images (800-4000px) and 188,282 bounding box instances. These images were collected from various platforms with multiple resolutions. As this dataset was released only two years ago, there are still not many articles about the performance of individual detection frameworks on this dataset except the comparison in the paper itself, which is the reason why we went with the COWC dataset instead.

DigitalGlobe, a yearly organizer of SpaceNet challenge, provides a lot of freely accessible data as well. These data are collected and provided by DigitalGlobe partners, among theme are big companies like Mapbox. Previous year the challenge was to detect road network; this year the task was to discover off-nadir buildings. Sadly, no dataset with annotated cars was provided yet, but at least these images might be used to test and estimate the quality of our detection method on different kinds of data. Russakovsky et al. [33] describe more in-depth (worthy 43 pages) the origin od ImageNet dataset and its contribution to the research of classification and detection of objects in aerial images. Methods for collecting and annotating datasets of such size are thoroughly described.[36]

The dataset that will be used the most throughout this thesis is COWC (Cars Overhead With Context)[7], which was assembled by members of the Computer Vision Group within the Computation Engineering Division at Lawrence Livermore

[6] http://www.zanzibarmapping.com/

[7] https://gdo152.llnl.gov/cowc/

National Laboratory. It contains aerial images at 15 cm per pixel resolution (15 cm GSD). This dataset contains images from six locations: Selwyn New Zealand, Potsdam, and Vaihingen (Germany), Toronto (Canada), Columbus and Utah (United States). In total, there are 32 716 annotated cars. The data is collected from other aerial platforms then satellites but resembles satellite imagery in its properties (nadir view angle, orthorectified). Data labels are images of the same size as the satellite image, all black with white pixels denoting centroids of annotated cars.

Even though all of these datasets provide a strong foundation for training classifiers, they miss for us crucial class of objects which is parked cars. These data sets can be used to train car detection framework which we will do in 3, but for the task of detecting parked cars only, we will have to search further or create our dataset.

2.4.1 Tools

Visual data are challenging to work with without specialized tools. For manual analysis and to visually examine the data we use *QGIS*, a free and open-source desktop GIS application.[8] *GIS* (Geographic information system) is a system for satellite imagery manipulation, analysis, editing, viewing, and storing. GIS can relate multiple data sources using location as the key index and in general, makes the work with aerial imagery easier and quicker.

2.5 Our Approach

As we have shown several methods were used before to detect parking places or parked cars, and although these methods provide insight into what might and what might not work, they mostly work with parking lots and vehicles on parking lots.

The first method that we propose and that will be implemented and throughout tested further in this thesis is multi-step pipeline which first detects cars then classifies these cars based on the broader context (their surroundings) and then estimates the number of parking places in the given area either using statistical methods and the number of detected parked cars or by extrapolating parking spaces along the roads. This method was chosen because it separates the problem into individual steps that can be further optimized and that were already solved before with sufficient quality and accuracy or at least similar research already exist in the given topic. For car detection, we will use the YOLO framework (3.2.2.4), which achieves excellent results on cars. For classification, multiple image classification methods will be used and compared (4).[30][8]

The second method builds on the assumption that parking type is usually the same on the whole street segment; for example, there's perpendicular parking alongside the entire street. We can then classify the whole street segment and

[8] https://www.qgis.org/en/site/

determine the parking style, then use statistical methods to estimate the parking capacity as multiple of parking spots per meter for given street parking type and length of the segments in meters. This method is shorter and more comfortable to implement but introduces few caveats. For example, the length of street segments varies a lot, and we would have to train the neural network to either classify the parts regardless of its size, or to classify longer sections on more places, and that combine these classifications.[5]

Other methods could be attempted as well. For example, detecting empty parking spots using the lines around them. This method would need higher resolution images than that are currently available as the lines have lousy visibility on the most commonly available resolutions of satellite images and most are absent on the streets altogether. Other than that, we could try detecting parked cars right in the detection neural network in the first proposed method.[14]

2.5.1 Car Detection and Parking Capacity Estimation

Here we in detail describe individual steps of the method that we proposed as our primary objective.

2.5.1.1 Object Detection With Deep Learning

In our proposed method, we build on top of car detection in aerial images. As this is the first step on which the rest of the pipeline depends, it is essential to achieve sufficient quality and provide excellent and accurate results for the following classifier. We can build on top of the work already done by many teams in competitions and research.[24][36]

2.5.1.2 Image Classification

With detected cars and thanks to data provided by Czech Geodetic and Cadastral Office[9], we can easily split recognized cars into two classes - parked and driving, which can be easily done. We classify the car as parked if it layers within any of the parking zones. With this dataset, we train the classifier.

2.5.1.3 Parking Places Counting

Counting parked cars would not be enough to accurately predict the parking capacity of the given area as it leaves empty parking places out. Empty parking places are hard to detect hence it is easier to classify the type of parking for given street segments and then use it to estimate it is parking capacity or uses available statistics to count the number of parking places from the number of parked cars or even the number of vehicles detected in a given area.

[9] https://www.cuzk.cz/

2.5.2 Street classification

Another method we proposed and that we will examine further is the classification of street segments. Street segments can have parking either on one or on both sides (or no parking at all). And the parking can be parallel or perpendicular. We decided to keep this method as our second option as it is harder to create a dataset for the training of the neural network.

Chapter 3

Car Detection

The first step in our pipeline is car detection. The detection of vehicles in aerial images has caught increasing attention in both academic and industrial fields and is widely applied in many applications, e.g., vehicle tracking, traffic monitoring, parking lot analysis, and planning, etc. We decided to use state of the art YOLO framework (3.2.2.4) for the detection and large COWC dataset (2.4) to train it on.

3.1 Dataset

The AI can be only as good as the data.Canter [4] Data will be base of all our methods whichever we decide to use. Following datasets are publicly available and can be used for tasks related to transportation analysis.

For this part of the problem, we used the COWC data set that we described in 2.4. With one slight change; Vaihingen and Columbus images are in greyscale, and as such, we will not use them for our research.

COWC dataset proved to be a good basis for car detection and counting in numerous articles. Etten [8] showcases YOLT2 detection framework on COWC dataset and achieved F1[1] score of 0.91 ± 0.09. He trained YOLT2 convolutional neural network for four days; he discarded greyscale images from the dataset and downsampled the images to 30 m GSD which is justified by current best available satellite images being of 30 m GSD resolution as well. For training, all the locations except Utah were used, which is 13 303 cars for training and 19 807 vehicles for testing. The lower number of cars for training then for testing makes sense because YOLT2 demonstrated consistent results even with small training sets.

Douillard [7] used COWC dataset in NATO innovation challenge. This challenge was about making decisions in a landlocked country where a contagious infection is spreading. The approach was left to the solvers, but one part of the solution was car detection in satellite images that they used for the prediction of traffic congestion. Instead of YOLT2, they used RetinaNet architecture which was

[1] 3.2.3

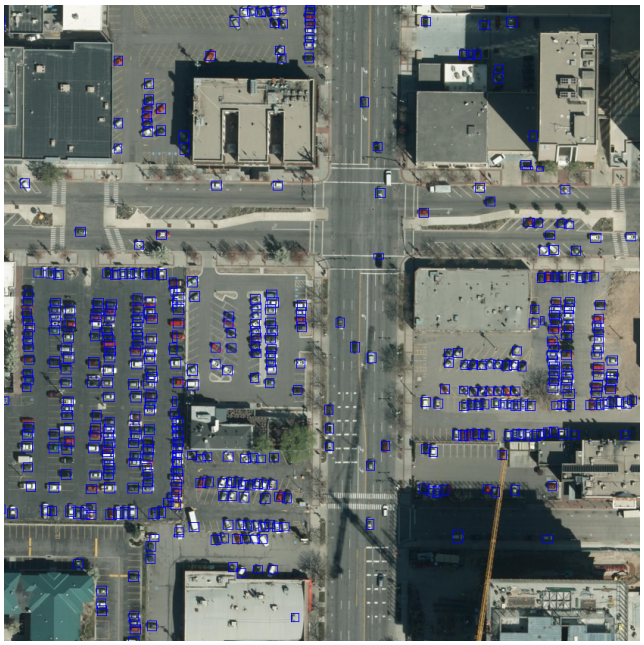


Figure 3.1: Showcase of our trained YOLT model showing detected cars over Salt Lake City, Utah, USA.

published by Facebook FAIR[2] in 2017 and the paper behind this architecture[26] won the Best Student Paper of ICCV 2017 (International Conference on Computer Vision).

3.2 Neural Networks and Machine Learning

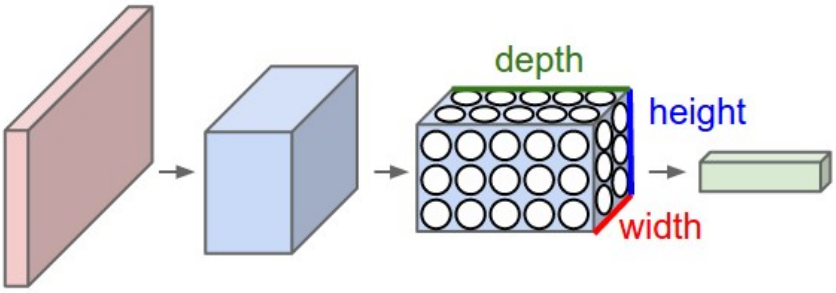
As we decided to use neural networks for car detection, we will provide a brief introduction to them, especially to convolutional neural networks that are mostly used for images.

3.2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs/ConvNets) is a type of deep neural network commonly used on images and photos. They are very similar to ordinary Neural Networks. Neural networks are inspired by the human brain; they are made of neurons. Each neuron has learnable weights and biases, it is, in fact, a function that receives multiple inputs (numbers), performs a dot product, adds bias, optionally uses non-linearity and outputs a number that is used further in the network. The base idea is that we can train the neuron in the sense of setting weights to individual inputs such that the function provides useful information that can be used further in the network. It can be things such as detecting edges or shapes.

[2]https://research.fb.com/category/facebook-ai-research/

Figure 3.2: Dimension transformation of input image in convolutional neural networks.[20].



3.2.1.1 Regular Neural Nets

Regular Neural Networks receive a single vector a through series of hidden layers transform it. Each hidden layer is made out of individual neurons that are fully connected with all neurons in the previous layer. Neurons in any single layer function independently; they do not share weights or connections. The last layer called the "output layer" and represents class scores.

Regular Neural Networks scale badly to full images. For example for image of size $64 \times 64 \times 3$ (64 pixels in height and width and 3 channels for colors - red, green, blue) neuron in first hidden layer would have $64 \times 64 \times 3 = 12\,288$ weights which does not seem like much, but the problem of scaling to images of bigger sizes can be seen. Image of commonly used size $1\,920 \times 1\,080 \times 3$ would cause the neurons to have $1\,920 \times 1\,080 \times 3 = 6\,220\,800$ weights. Furthermore, we would use more of these neurons, and the weights would add up quickly. We can see that the full connectivity is wasteful, for single layer processing full HD image we have 6 220 800 weights and neural networks can have multiple layers with this size, and that a large number of parameters would cause overfitting.

3.2.1.2 CNNs

Convolutional Neural Networks build on the fact that the input is an image. Unlike in Regular Neural Networks, the layers in CNNs have neurons arranged in 3 dimensions - width, height, and depth[3].

Convolution neural network consists of 3 types of layers: Convolutional layers, Pooling layers and Fully-connected layers as in standard multilayer neural network. The benefit of CNNs is that they are easier to train and have fewer parameters than fully connected networks whiles keeping the number of hidden units the same.

[3]depth is the number of channels/colors that the image has

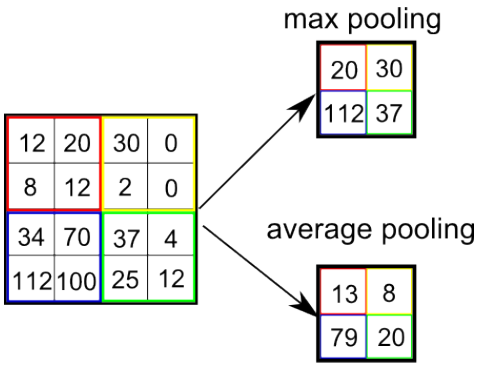
3.2.1.3 Convolutional Layer

The convolutional layer, as the name suggests, is the core building block of convolutional neural networks. The convolutional layer consists of a set of filters. Every filter has a small spatial size[4] but is applied through the full depth of the input channels. A typical filter can, for example, have a size $5 \times 5 \times 3$ (5 pixels width, 5 pixels height and 3 for all three color channels). The filter is then slid (convolved) across the whole image and computes the dot product between input at any position and filters weights. As we convolve the filter we get a two-dimensional map with activations of the filter (response of the filter to the input at every spatial position). Intuitively the filters will be trained to detect some features such as color, edge, or orientation on the first layer and gradually more sophisticated features such as wheel-like or eye-like patterns on higher layers in the network. We will have multiple filters in each layer, and each filter will produce its map. We will stack these maps along the depth dimension and create new output volume.

3.2.1.4 Pooling Layer

The pooling layer is usually periodically inserted in-between convolutional layers to reduce the number of parameters, to progressively reduce the volume of the representation, and therefore to also prevent or control overfitting. The pooling layer works on all depth slices independently and reduces their size spatially using either MAX or MEAN operation (the latter being used rarely). The most commonly used pooling filter is of size 2×2 and is applied with a stride of 2. The filter downsamples every depth slice to half width and half height, thus reducing the size to 25% (in other words, discarding 3/4 activations). MAX operation would take a maximum of the four activations (2×2) and MEAN operation would take the mean. Pooling leaves the depth dimension unchanged.

Figure 3.3: Maximum pooling vs average pooling



[4] small size compared to the size of the input

3.2.1.5 Fully-connected Layer

As in regular neural networks, neurons of the fully connected layer have connections to all activations in the previous layer. Hence the activations are computed using matrix multiplication and adding the bias.

3.2.2 Network Architecture

As described before, convolutional neural networks are made up of three types of layers: convolutional, pooling, and fully-connected. These layers follow some patterns when being stack together to form whole CNNs. The most common CNNs stack a few convolutional layers followed by the pooling layer, repeat this pattern until the spatial size is small enough. Then the network transitions to fully-connected layers and the last of fully-connected layers hold the output such as class scores. Object detection architectures are split into two categories: single-stage and two-stage.

Two-stage architectures first classify potential target objects into two categories: Foreground and background. Then all the foreground's potential targets are classified into the desired classes, such as cat, dog, airplane, etc. Two-stage architecture is thus slower but more accurate. The most famous two-stage state of the art architecture is Faster-RCNN.

Single-stage architecture classifies objects in the first pass. It is, in general, faster but less accurate nevertheless with the advancement and further research, both architectures currently are on par and achieve similar results in a similar time. RetinaNet and YOLO object detection frameworks belong here.[15][30]

3.2.2.1 Resnet

ResNet is an architecture of a convolutional deep neural network released in the year 2015. As we will use this architecture in step two (4.2.1.2) we will discuss it further there.[15]

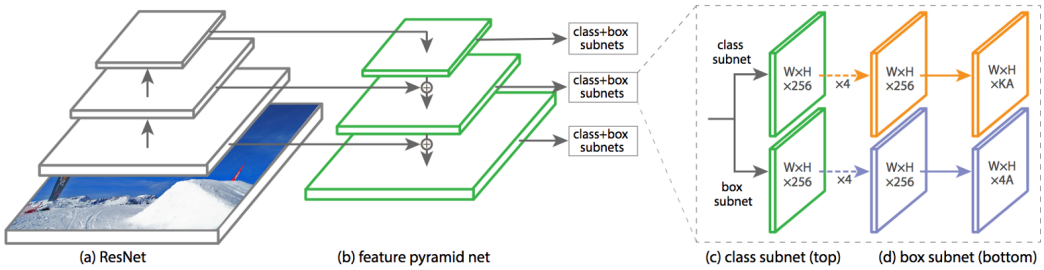
3.2.2.2 RetinaNet

RetinaNet is the first neural network architecture we considered for the training of our detector. It uses Resnet at its core but instead of taking ResNet's last feature maps (of shape $7 \times seven \times 2048$), using average pooling and feeding the result into a Fully-connected layer, it adds Feature Pyramid Network. RetinaNet picks feature maps from different layers inside ResNet and uses them as rich and multi-scale features. As the smaller feature maps are too crude, RetinaNet first upsamples them to the size of bigger feature maps and then takes their sum. Each of the FPN encodes at a different scale different information hance all the feature maps contribute to the final object detection. FPN takes the output of the third (512 channels), fourth (1024 channels), and fifth (2048 channels) layers of ResNet.[26][25][7]

At each level of FPN, several anchors are moved around the feature maps. Anchors are base frames in which the object is being detected. RetinaNet uses five sizes and three different ratios for the anchors, thus using 15 unique anchors scaled accordingly to the spatial dimensions of the FPN level where they are being used.[26][7]

All the previous features have been used before. What RetinaNet introduced that made it state of the art detection framework was *Focal Loss*.[26] Traditional single-stage architectures are usually overwhelmed by the background objects that are often selected as a potential target object. Focal Loss deals with this by giving well-classified examples lower weight hence forcing the model to learn on harder examples.[26][27]

Figure 3.4: Architecture of RetinaNet.



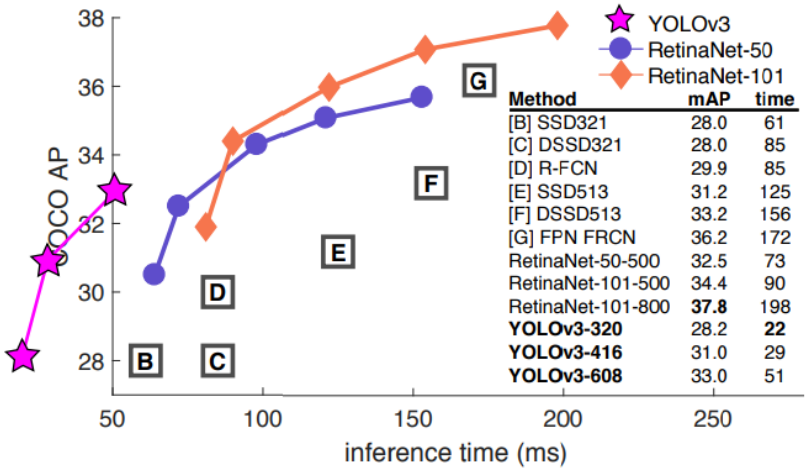
3.2.2.3 R-CNN

R-CNN is another detection system that works in two steps. In the first step, regions of interest are proposed using features-based detection method 2.2. Its second step it classifies these regions. R-CNN creates region proposals using Selective search - it looks on the image through windows of different scales and sizes and for each window tries to group adjacent pixels by color, texture, or intensity. Once proposals are created they are warped to square size and passed to AlexNet, classification neural network.[12][32]

3.2.2.4 YOLOv3

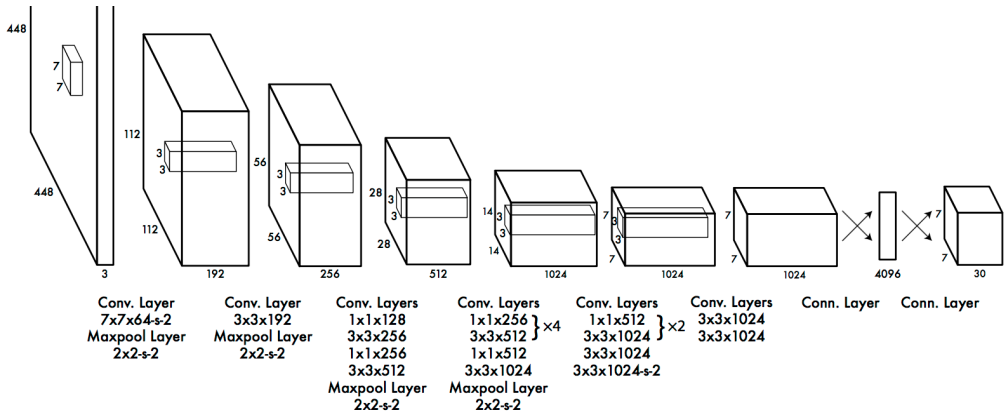
YOLOv3 (You Only Look Once version 3) is another state of the art, real-time, object detection system. Instead of applying classifier or localizer on multiple scales and locations on the image, YOLOv3 applies a single neural network to the full image. The network itself divides the image into smaller regions and predicts probabilities and bounding boxes for each region. Schema of YOLOv3 architecture is in figure 3.5. This approach has one significant advantage, and that is that the network looks at the whole image and thus can create predictions that are informed by the global context in the image. Furthermore, unlike some other systems such as R-CNN, this network is evaluated just once over the whole image. Evaluating just once

Figure 3.6: Comparison of different detection frameworks.[31]



makes YOLOv3 extremely fast, which makes both the training and detection faster. Comparsion of accuracy and inference speed of YOLOv3 with RetinNet is shown in figure 3.6.[31][30]

Figure 3.5: YOLO detection network has 24 Convolutional layers followed by 2 Fully-connected layers. Image taken from[31]



3.2.3 Neural Nework Accuracy

It is important to know the accuracy of a neural network on a given task, but when it comes to measuring the quality of the neural network or detection framework

accuracy is not the only measure, many methods and formulas exist. *Precision* precision in detection frameworks means how many of our positive predictions are correct 3.1. *Recall* measures how accurate we are finding the positives 3.2. *F1* score combines precision and recall into one statistic. It is the harmonic average of the recall and precision 3.3. *IoU* (Intersection over Union), also known as the Jaccard index, measures how much our predicted region and the ground truth region overlap 3.7.[5] *mAP* (Mean Average Precision) is a metric for the measure of the accuracy of object detectors. The mAP is the average of the maximum precisions at different recall values.[17]

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

$$\mathrm{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.3}$$

Where TP, FP, and FN denotate *true positive*, *false positive*, and *false negative* respectively.

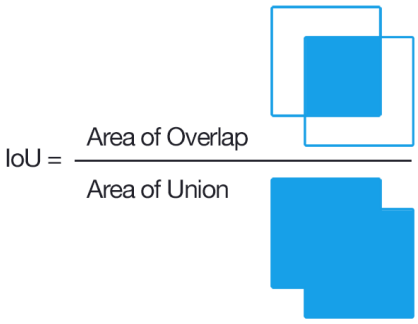


Figure 3.7: Intersection over Union (IoU, or Jaccard Index) visualy explained.

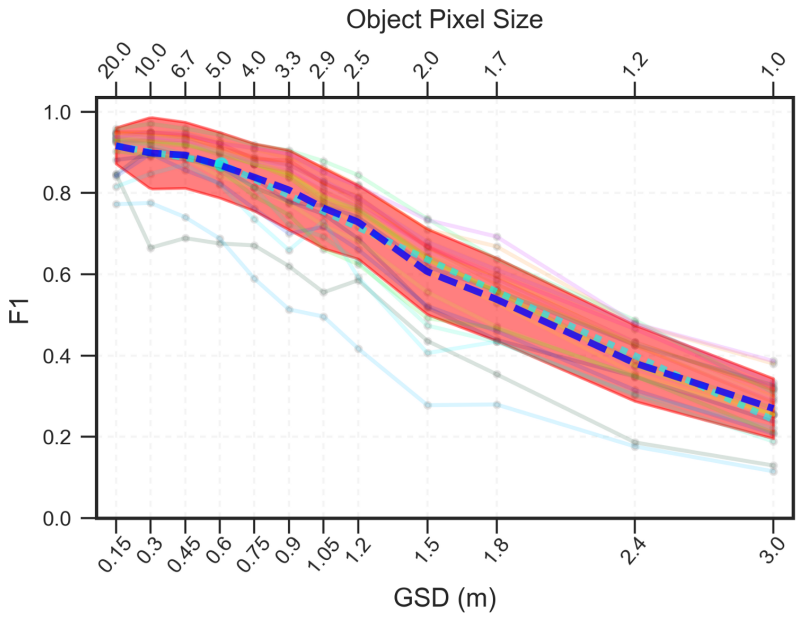
3.3 Our Approach

We used the YOLO framework for car detection. We trained it using the COWC dataset with resolution reduced from 15m GSD to 30m GSD hence reducing the overall size of the dataset by 75% (this step is justifiable because the F1 score and accuracy change negligibly as shown in the figure 3.8) which made handling of the data easier and training of the detector faster. Following article by Etten [8], we trained the detector for four days on Cloud Engine (Google Cloud Platform) VM with 4 cores, 14GB RAM and 1 NVidia tesla p100 GPU. We taught the network for

[5] https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

2 200 epochs, but the accuracy did not change a lot from the 1000th epoch further. Although they improved, slightly overall performance did not vary by more than 20%. We then tested our network on the aerial images of Utah from the COWC dataset achieving an F1 score of 0.91. We also tested the network manually on images of the city of Prague. Overall performance on clearly visible cars was satisfactory but streets of the Prague are more narrow with taller buildings than most of the images in COWC dataset and as such it introduces new problem of some cars being in shadow (partly or wholly) or being for example partially covered by the adjacent building as not all the images available were fully orthographic and had some perspective artifacts. Because of these conditions we detected 572 cars out of 697, about 82 % of vehicles. The false-positive rate was low overall, and we identified only 20 non-car objects like cars. We calculated this accuracy over ten randomly selected images from Prague. Accuracy over the city of Brno was on par with Prague; 0.81 over ten randomly selected images.

Figure 3.8: Dependence of F1 score on GSD.[11]



(a) Example of high accuracy of our detector over urban area with little to no shadow and clearly visible cars.

(b) Example of low accuracy of our detector over urban area with narow streets and cars partially or fully in shadow.

Figure 3.9: Different pictures over Prague and accuracy of our detector. Green squares are driving cars and blue squares are parked cars as deduced from the information about parking zones.

Chapter 4

Detecting Parked Cars

In this section, we design a method that classifies the detected cars as driving or parked based on the wider context - it is surroundings. We could statistically estimate the number of parking places based on the number of detected cars alone ([6] provides the number of parking places per car) But recognizing parked vehicles provides us with more information. Also, it can help us better localize parking capacities.

Figure 4.1: Examples of patches of cars with context (real size 25×25 m).

(a) Examples of parked cars.

(b) Examples of driving cars with context. Last image shows wrongly detected driving car.

4.1 Dataset

Because we could not find a dataset of parked cars, we created such dataset ourselves using the annotation of parking zones in Prague and Brno. We detected cars in the annotated areas with our previously trained model for car detection and used the parking zones data to put these detected cars into two categories: parked and driving.

class	examples	error rate
driving	8 577	19%
parked	49 569	2%

Table 4.1: Size and error rate of training dataset for car classification neural network

Data about parking zones contain 9 024 zones around the whole of Prague, mainly in the city center. This data also includes the parking capacity of the given area and its pricing. We wanted our data to be as clean as possible for the next neural network, and thus we checked all the zones and fixed missing spots, added parking lots and other areas where cars were parking. We added over 2 000 additional zones with a total area larger than the original zones.

We then combined the data about parking zones with previously detected cars. For each detected car we took its centroid and checked whether it lays within any zone. This way we created a dataset for car classification with 49 569 cars; 40 336 parked and 8 577 driving. Thanks to the low false-positive rate and manually added zones we were able to catch a lot of misclassified cars but we were not able to spot and mark all of them thus the error rate was approximately 19% (9% wrong detections that ended mostly in driving cars dataset and 10% parled cars classified as driving). Even though almost 50 000 seems like a significant number, it is quite a small amount for the training of the classifier. For example the ImageNet dataset contains more than 14 million images in 20 000 categories[24].

4.1.1 Data Augmentation

Neural networks with a large number of parameters trained on a small dataset are prone to overfitting. Overfitting happens when the neural network remembers the training data and can classify them correctly while having a significant error on testing data. This occurs when the neural network learns to recognize small details and noise on training images, and these concepts do not apply to new data.

One of the methods to prevent the overfitting of the convolutional neural network on a small dataset is data augmentation[1]. Data augmentation is an artificial transformation of images to create new images for a dataset of small fixed size. However, these transformations cannot be arbitrary because they would confuse the convolutional neural network. For example, in the classification of animals, horizontal flipping cannot be used. For aerial images, this option can be used because the images are captured from the bird's-eye view. All our images are captured from constant height with constant zoom; as such, we can not use zoom transformation. Shifting cannot be used as well because the car is always in the center of the image. The rest of the transformations that we used is listed, described, and showcased lower.

[1] https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#data-processing

4.2 Classification Network

Convolutional Neural Networks have in the past years shown break-through results in image classification tasks. In the section, we explain how convolutional neural networks can be applied to classifying cars into two categories. We also introduce 3 state of the art architectures for neural networks, Resnet50 4.2.1.2, Densenet121 4.2.1.3, and AlexNet 4.2.1.1.

4.2.1 Network Architectures

We decided to try as many state of the art architectures as possible to find the one that is best for our task. Thanks to the computational power of Metacentrum 4.3.1 we were able to run them with easy and continue our work while they were running. We experimented with the following architectures.

4.2.1.1 AlexNet

AlexNet is named by its author Alex Krizhevsky. This architecture won the ImageNet Large Scale Visual Recognition Challenge in 2012 and achieved a top-5 error of 15.3%, which is more than 10.8 percent less than that of the second place. It was one of the first deep neural network models, and the training was only made viable thanks to the utilization of GPUs. AlexNet contains eight layers. The first five layers are convolutional, some of them followed by max-pooling layers, and the last three layers are fully connected layers.[23]

4.2.1.2 Resnet

Resnet came after AlexNets successes in the year 2015, and its authors tried to tackle a few issues that came with deeper neural networks. Namely, the vanishing gradient problem and the trainability of deep neural networks. The core idea of ResNet is the introduction of so-called "identity shortcut connections" that skips one or more layers. The basic version of ResNet consists of 34 layers. Each layer performs 3x3 convolution with a fixed feature map and every second layer bypasses the input.[15]

4.2.1.3 Densenet

Densely Connected Convolutional Networks (Densenet) solves the problem of slowly vanishing information about input in deep neural networks by directly connecting all layers with matching feature-map sizes. Each layer in the network obtains additional inputs from all previous layers in the network and also passes to all forward layers its own feature map. In contrast to ResNet, the weights are never summed before passed to the layer; they are appended instead. The counter-intuitive effect of this architecture is the lower amount of weights needed in the network. Authors of Densenet further observed that dense connections have a regularizing effect, thus reduce over-fitting of the network smaller data set sizes.[16]

4.2.2 Loss Function and Optimization Algorithm

The backpropagation algorithm was chosen to solve the problem. Since this is a regression problem where the network tries to minimize the difference between the actual class of the car and the predicted class, the loss function for this problem is most often *Binary Cross-Entropy*, equation 4.1.[13]

$$H_{p(q)} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{4.1}$$

Where y is the label (1 for driving points and 0 for parked) and $p(y)$ is the predicted probability that the car is driving for all N cars.

Three optimizing algorithms, Nadam, RMSProp, and SGD, were used to optimize the loss function. Optimization algorithms serve to find a suitable (optimal) solution to a problem, especially when a mathematical description of a problem solution is unknown. Nadam and RMSProp are adaptive optimization algorithms, while SGD offers greater control over selected parameters such as learning rate or momentum, which reduces the risk of deadlock at local minima and increases the speed of convergence.[2]

4.2.3 Transfer Learning

Learning neural networks from scratch requires a vast amount of data and is rather computationally demanding, requiring up to weeks of training. Instead, it is a common practice to pre-train convolution network only once on a huge dataset (e.g., ImageNet with 1.2 million images in 1000 categories) or use weights already pre-trained and then use this pre-trained model as initialization or fixed[3] feature extractor for the given task. Many researchers release their final checkpoint for the benefit of others who can then use these weights for fine-tuning on their own dataset.[21][4]

4.2.3.1 CNN as Fixed Feature Extractor

By taking CNN pre-trained on ImageNet, removing the last fully-connected layer and treating the rest of the network as a fixed feature extractor for the new dataset, we obtain a vector (called *CNN codes*) that contains activations of the hidden layer right before the classifier. Once we extract these CNN codes for all images in our dataset, we can train a linear classifier for the new dataset. This has the benefit of faster training because the weights for the lower part of the network[5] stay fixed. The lower part of the network is also usually trained on large datasets such as Imagenet, and thus the network learns to extract various features.

[2] http://cs231n.github.io/neural-networks-3/

[3] The weights are frozen, and only the upper part of the network is trained.

[4] http://ruder.io/transfer-learning/

[5] Lower part of the networks are the layers that the image is directly fed to.

4.2.3.2 Fine-tuning

Another method of transfer learning is Fine-tuning. In this approach, we not only replace the classifier on top of the convolutional neural network that handles the classification but we also unfreeze the weights of pre-trained model and train them (fine-tune them) with the last custom layer(s) by continuing the backpropagation. This also allows the rest of the network to fit the new data and adjust the recognized features to fit the new data better.

4.2.3.3 When to use Transfer Learning

When deciding whether to use Transfer Learning on a new task, a few things matter the most: The size of the dataset, which is compared relative to the size of the dataset the network was initially trained on and the number of weights, and the similarity to the original dataset, the network was trained on. In our case, the dataset is relatively small and very different from Imagenet. Since the dataset is small, it is a good idea to train just the top-most layer(s) and since the dataset is very different it might work better if we do not train the dataset from top but instead from activations earlier in the network.[21]

4.3 Training

We tested various architectures for the part of the neural network the extracts the features, and for the fully connected layers that create the classification. We worked with Resnet50 4.2.1.2, Densenet121 4.2.1.3, and AlexNet 4.2.1.1. For the implementation, we use Tensorflow[6] and Keras[7] libraries in python, which allow us to focus on the architecture and training itself and handles the underlying mechanisms.

We started by training the detector based on ResNet with one fully connected layer (512 wrights) and DenseNet with two fully connected layers (512 and 256 weights respectively). For both models, we used weights pre-trained on Imagenet for the lower part of the network. We froze the lower layers to lower the number of learnable parameters and to see whether at least some results can be achieved given the small amount of training data.

[6]https://www.tensorflow.org/
[7]https://keras.io/

```
#!/bin/bash
#PBS -N train-densenet
#PBS -q gpu
#PBS -l select=1:ncpus=4:mem=16gb:ngpus=2:gpu_cap=cuda35
#PBS -l walltime=8:00:00
PYTHONPATH=$PYTHONPATH:"$DIR/workdir" python train_network.py
```

Listing 4.1: Bash script for network training with the virtual machine configuration.

We trained on Metacentrum on a virtual machine with 16 GB ram, 4 CPUs and 2 GPUs. Each experiment took from 2 to 8 hours to run. Configuration of the machine is part of PBS commands used to submit task into the computation queue and is shown in 4.3

4.3.1 Metacentrum

All calculations related to network training were performed at the Metacentrum[8]. The Metacentrum is a free project for academics and students. These users have access to computing and storage capacity. The Metacentrum operates several powerful computers across the Czech Republic. We used machines equipped with NVIDIA GeForce GTX TITAN X graphics cards to train the convolutional neural network. In addition to the graphics card, the cuDNN11 library is needed, which effectively calculates common operations in deep neural networks. Tasks are queued and run by PBSPro. The script used for training is shown in 4.3. As part of this work, 228 days of processor time were used, and about 120 GB of storage space was required to store all training images.

4.4 Results

When the first set of experiments finished, we witnessed an accuracy of 0.5 on testing data for both networks, which is the same accuracy as if we decided the classes on random. For ResNet, we see from the training data that the accuracy for training data as well as for validation data was the same, e.g., 0.5 which means that our neural network could not generalize from the data and did not learn anything. For Densenet architecture, the training took another path. The network overlearned almost entirely in the first epoch and achieved accuracy of 0.899 on training data, but the accuracy on validation and testing data was still 0.5, which means that the network over-fitted.

4.4.1 Fighting with Overfitting

The first experiments ended with bad results. We tried multiple things to prevent overfitting and achieve better results with our network. We obtained more data

[8] https://metavo.metacentrum.cz/

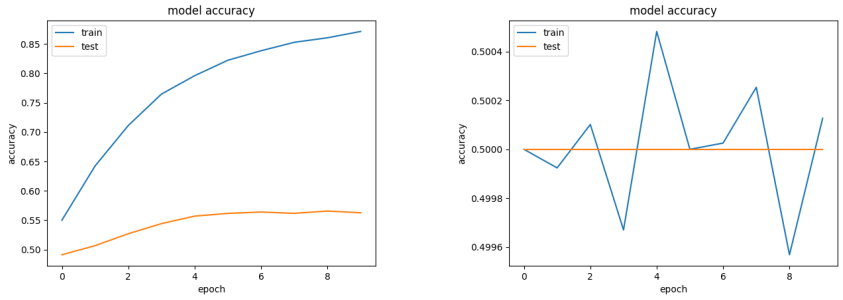
using data image augmentation and adding images from the Brno 4.4.1.1. We tried different batch sizes 4.4.1.2 to make the weight updates more random and enable the network to escape local minima. We also tried different configurations network configuration 4.4.1.3.

4.4.1.1 More Training Data

First, we try to boost the amount of data using Data Augmentation (4.1.1). For each original image of a driving car, we create three augmented images using 0 to 5 augmentations in random order. This way we obtain a dataset with 40 336 parked and 25 731 driving cars, in total 66 067. We augment only the driving cars to make the dataset more balanced. Although it is possible to train the network on unbalanced datasets using weights to compensate it is better to have balanced data.

Training with the augmented dataset showed only a slight improvement of 5% with the Densenet architecture, we did not have a random classifier anymore but classified 55% of the cars correctly. Although that is still a bad result, it showed us that the assumption that the training dataset is too small was correct and that we might get a better result with more data.

Figure 4.2: Training progress of ResNet and DenseNet architectures on augmented data.



(a) DenseNet (b) ResNet

Next, we tried to obtain more examples of driving cars. We added data for Brno that we left out previously because of their small size. This provided another 1 374 driving car samples. We also added more cars from Prague from areas that were not used before because we were missing information about parking in these areas. To obtain these cars, we detect all vehicles over the given region and took only the cars that were close ($<$ 1m) to a street on which cars never park, such as the highway. We had to be very conservative with this approach in order not to add to many misclassified cars. The distance of 1m and less had an accuracy of 90%, as shown in the statistics in figure A.1a. In total, we added another 2 409 cars which summed up to 32 958 driving car samples after boosting.

After training with the once again bigger dataset, the results again improved

only slightly, from 0.550 to 0.559 accuracy. We also tried this dataset with all the architectures we mentioned (ResNet, AlexNet, etc.). All of them achieved worse than Densenet with accuracy 0.56.

4.4.1.2 Batch Size

Using too large a batch size can hurt the accuracy of the network during training because it reduces the randomness of the gradient descent. Using a smaller batch size produces stochastic and choppier weight updates, which can have two positive effects. Firstly, it can help the training to exit local minima in which it might have gotten stuck previously. Secondly, it can cause the training to settle in flatter minima, which generally indicates better generalization. We started training all networks with the recommended batch size 16. After the first unsuccessful results, we tried both smaller batch size (2, 4, and eight images per batch) and bigger batch size (16 images per batch) without any significant influence on the training results except longer running time for the smaller batch size and vice versa.

4.4.1.3 Number and Size of Hidden Units

In some cases, using too many or too few hidden units can make the network challenging to train. With too few units the network might not have the capacity to express the task required, and with too many, it may become slow to train. We tried a different number of hidden units as well as a different number of FC layers. For all architectures, we tried: 1 hidden unit connected to the last layer of underlying feature extractor, One layer with 256, 512 or 1024 units connected to a single unit that outputted the class, and two layers each with 256, 512 or 1024 units again connected to a single unit as the output. The larger number of hidden units usually resulted in faster overfitting and divergence of accuracy on the training and validation set. Smaller sizes tended to produce better results. The best architecture used only 1 unit connected to the last layer of the DenseNet model, the python model of this configuration is shown in 4.4.1.3.

```
model = Sequential()
model.add(DenseNet121(include_top=False, pooling='avg', weights=
    'imagenet'))
model.add(Dense(1, activation='sigmoid'))
model.layers[0].trainable = False
model.compile(optimizer='Adam', loss='binary_crossentropy',
    metrics=['accuracy'])
```

Listing 4.2: Python script with the setup of DenseNet model for training.

4.4.2 Discussion

The difference between the networks and the reason why one over-fitted while the other did not learn anything, is most likely caused by the different architecture and

thus various features and their weights that are learned and that the network outputs in the last layer. For Densenet, these features were different enough for our fully connected layers to pick up the slight differences between images and overfit while for ResNet the output most likely represented features that were similar between images and from which the network could not generalize.

4.4.3 Statistical Classification

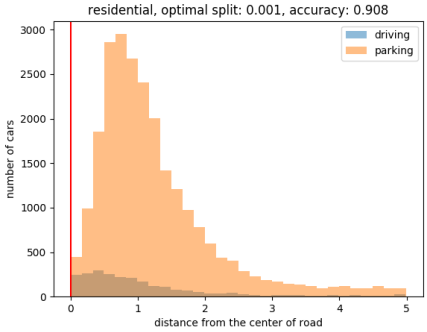
Although this part of our method failed, we wanted to continue our experiment. The assumption is that with more resources and data, this part would be successfully solved. To compensate for the classification neural network, we used statistic methods to classify the cars. For each detected car, we found the nearest street segment as obtained from OSM. We then separated the cars by the type of the closest street. Four main categories with the highest car count were residential, primary, secondary, tertiary (descriptions of the types can be found on OSM wiki[9]). Histograms with a description of individual street types are shown in figure A.1. For each street type we then the select distance from the street, where cars under this distance are classified as driving and cars further are classified as parked. This distance is selected based on the available data to provide the best possible result. Accuracy of this method was 0.881. Accuracy for individual street types is in table.4.2.

Street Type	Cars Count	Accuracy	Driving Cars	Parked Cars
residential	26532	0.908	2445	24087
tertiary	4456	0.831	1164	3292
secondary	3134	0.752	1722	1412
primary	2317	0.845	1697	620
trunk	572	0.862	376	196
living_street	323	0.882	39	284
secondary_link	109	0.972	67	42
trunk_link	90	0.889	70	20
primary_link	88	0.898	72	16
unclassified	34	0.853	6	28
tertiary_link	13	0.923	10	3

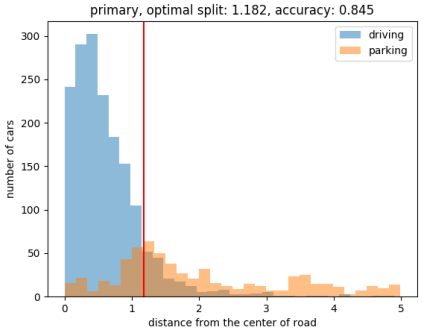
Table 4.2: Car classification by nearest street type. Our occuracy for given street type and number of cars for driving and parked class.

[9]https://wiki.openstreetmap.org/wiki/Key:highway

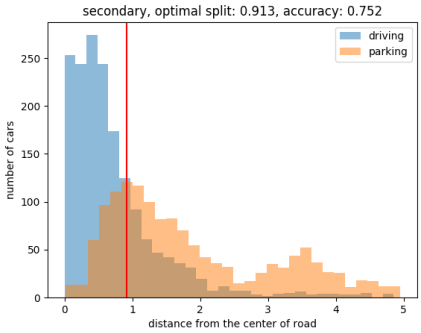
Figure 4.3: Histograms of distances to the nearest road by road type and car class for 4 street types with highest car count.



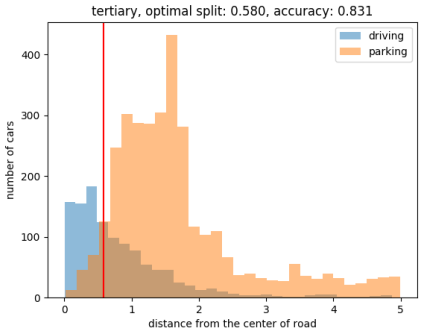
(a) Residential; roads accessing residential areas or around residential areas.



(b) Primary; a major highway linking large towns, normally with 2 lanes.



(c) Secondary; a highway which is not part of a major route, but nevertheless forming a link in the national route network.



(d) Tertiary; roads linking villages and towns.

4.5 Summary

The main complication for parking vs. driving cars classification was the scarcity of data, which is a solvable problem for more prominent institutions that can afford the creation of a bigger and accurate dataset for this step. Our results have shown that more data improved the resulting accuracy of the network, and this trend would probably continue if we were able to obtain more data. Another improvement would have a cleaner dataset. Obtaining such dataset is hard because even if the data about parking zones were complete and accurate there is still a lot of people that park on places where they should not, outside of these annotated zones. Even though the second method, statistical classification 4.4.3, that we used to substitute the classification neural network achieved good results, it is not well suited for this

task because it classifies almost all cars on residential roads as parked. The neural network, on the other hand, could learn to differentiate driving from parking cars no matter what the street type is.

Chapter 5

Parking Capacity Estimation

As the name of the paper suggests, our primary goal is to provide estimates of the parking capacity of individual areas. Because of the complexity of the task, there is not much research publicly available to build on. Davis et al. [6] estimated the parking lot footprints in the Upper Great Lakes Region of the USA. They calculated the parking lot of coverage from digitized orthophotos. The time and labor intensive nature of this task forced them to use sampling for larger scale areas. They estimated the number of parking places per car to be in the range from 2.49 in Indiana to 2.95 in Michigan.

5.1 Street Classification

We were mainly interested in the on-street parking capacity as there already exist solutions for parking lot parking capacity estimation but not any yet for on-street parking. We took previously classified parked cars. As we are interested in the streets, we filtered out cars that are further than 5 m from the closest road. We then assigned each street the cars that are closest to this street. If the street did not have a car assigned, we considered it non-parking. If there were cars assigned we check whether they are on the same side or both sides of the street. This way, we classified the street as either with parking on one side or both.

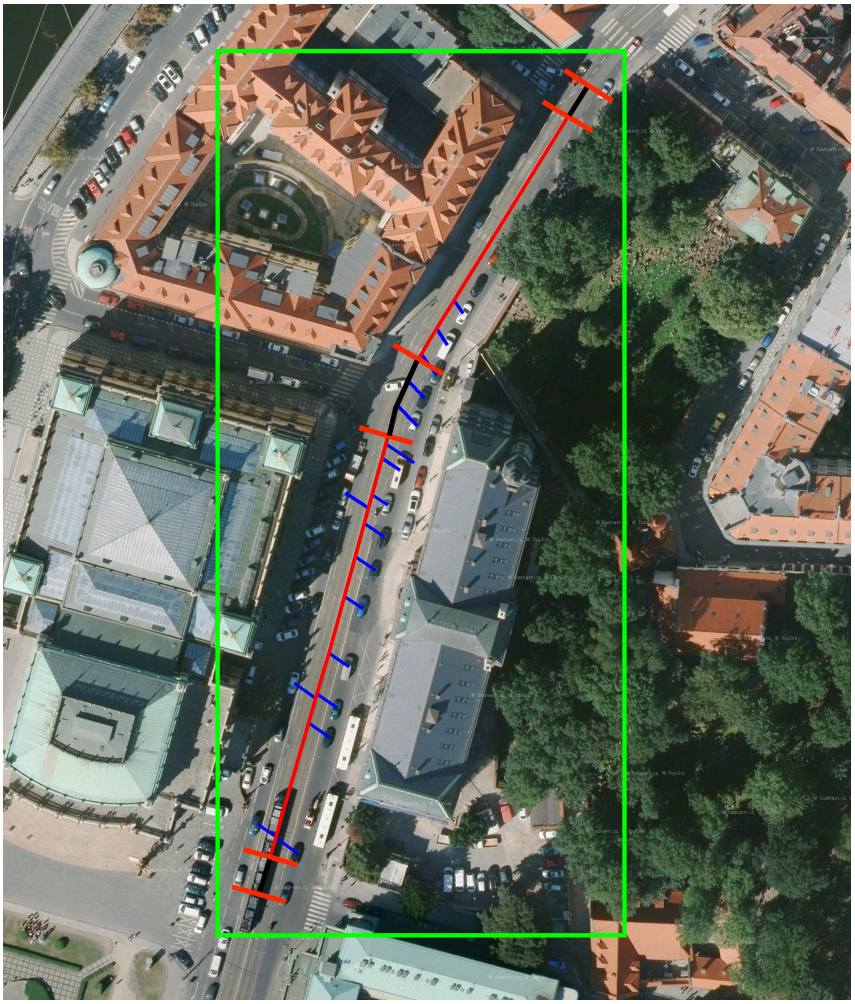
5.2 Street Parking Capacity Estimation

To obtain an accurate prediction of the parking capacity of each street, we went through all segments that form the street as obtained from OSM. Segments meet in the middle of the crossing, which means that part of the segment is always without parking. In the EU, it is also not allowed to park closer than five meters[1] from the crossing. We took the length of each segment, subtracted 14 meters from its length for each end; previously mentioned 5 meters and 2 meters for the part that is in the

[1] StVO §12

crossing. This process is illustrated in figure 5.1 In the Czech Republic as well as in the rest of EU the size of one parking spot must be at least 2.20 m × 6.0 m for parallel parking and 2.40 m × 5.30 m for perpendicular.[2] We did not differentiate parallel and perpendicular parking in our estimation method, so we assumed the size of one parking spot to be 5 m, close to the average. To obtain the parking capacity for a street segment, we then divided the adjusted length by 5 m.

Figure 5.1: Process of estimating the length of the street that is available for parking. The red line is the center of the street and places where it is split into individual parts. Black lines are segments that were removed from the length of the road. Blue lines show perpendicular lines from the detected car to the street. Slight inaccuracies are caused by transfering between coordinates and pixels.



[2] ČSN 73 6056

5.3 Results

To assess the accuracy, we went through 10 randomly selected streets and manually counted the parking capacity from satellite images and compared them with the predicted capacity. For streets with parking on both sides, the average real[3] parking capacity was 45. Our method had an average error of 13, out of the ten streets only one was wrongly classified as having parking on both sides while it had parking only on a single side. Four examples are shown in figure 5.2. For streets with parking on one side, we again took ten random streets. The average parking capacity was 23.1, and our average error was 7.2. Three of the ten streets were wrongly in the category of streets with one side parking and had parking space on both sides. The overall average error was 0.327; average real street parking capacity was 34.217 cars, which we predicted with average error 10.542 cars. All results are in table 5.1.

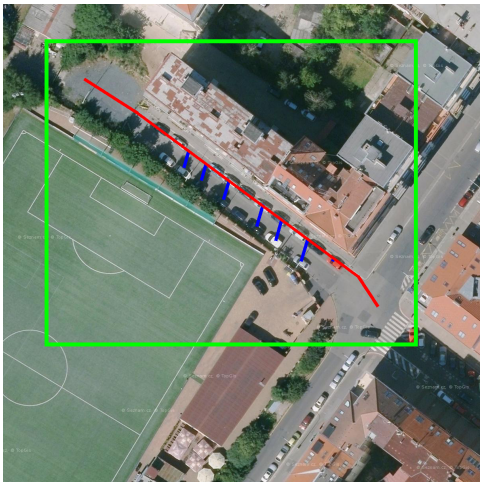
Street Class	Avg. PPC	Deviation of PPC	Avg. Error
Parking on both sides	45.1	±13.0	0.386
Parking on one side	32.1	±7.2	0.251
Both	39.6	±10.5	0.327

Table 5.1: Street parking type classification results. PPC stands for Parking Places Count.

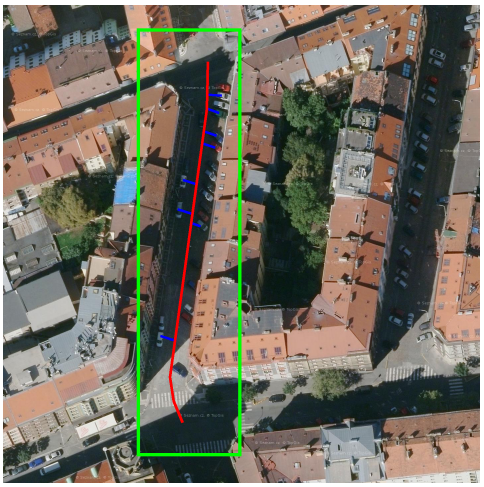
[3] as counted by us

Figure 5.2: Streets with on street parking showing datacted cars with their classification and predicted parking capacity.

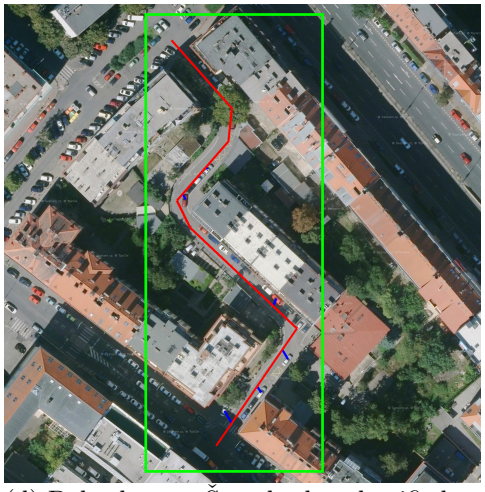
(a) Chelčického, classified as both-side parking street with capacity 56 cars, real capacity ∼60 cars.



(b) Marie Cibulkové, classified as both-side parking street with capacity 32 cars, real capacity ∼22 cars.



(c) Víta Nejedlého, classified as both-side parking street with capacity 36 cars, real capacity ∼25 cars.



(d) Bohuslava ze Švamberka, classified as both-side parking street with capacity 70 cars, real capacity ∼40 cars.

Chapter 6

Conclusion

We proposed, implemented, and tested a new method for on-street parking capacity estimation using aerial images. Parking capacity estimation has a wide variety of use cases. It can be used by the government for city growth planning or in the retail sector for customer flow analysis and peak business hours monitoring. Further usage could be automatic navigation to nearest free parking sport which could save up to €136.4 billion a year in the U.S., United Kingdom and Germany alone. Our method consists of several steps which are described in the following sections and can be applied at any location assuming that aerial images with sufficient resolution exist.

6.1 Car Detection

For car detection, we used the current state of the art convolutional neural network architecture YOLOv3 implemented in Darknet. We trained the network for four days on the COWC dataset containing 32 716 unique cars. We achieved accuracy on par with recent research. Our method detected 91% of the cars in the validation dataset of images over Utah. We further estimated the accuracy in the city of Prague to be 0.821, 9% less than on Utah. This is caused mostly by the shadow cast by buildings into the narrow streets partially or fully obscuring the cars.

6.2 Car Classification

We used three state-of-the-art convolutional neural networks, DenseNet, Resnet, and AlexNet pre-trained on the ImageNet dataset to classify previously detected cars as either parking or driving. We were unable to achieve the desired accuracy because of the small dataset with a high error rate. Our networks overfitted on the data quickly and were unable to distinguish parked from driving cars. We used several methods to prevent overfitting. We augmented the images to have a bigger dataset for training. We tried multiple network architectures and configurations all without success. Our method correctly classified 55.9% of the cars. We also introduced a statistical method that classified cars based on the distance and type of nearest street. This method

has accuracy 88.1% on data from Prague but is unable to distinguish cars on the residential roads, which are the most common type of road with on-street parking.

6.3 Parking Capacity Estimation

Further, we introduced a method for estimation of parking capacity for on-street parking. Our method can assign to the street one of three classes: street with parking on one side, on both sides and without parking, with 80% accuracy. Based on the class and length of the street parking capacity of the given street is estimated with an accuracy of 67.3%.

6.4 Future Work

Our work leaves a lot of space for further research and improvement, mainly in the topic of driving vs. parking car classification and parking capacity estimation. Our method might find its usage in GPS navigation and map systems, surveillance software, or in government surveys.

Bibliography

[1] Ali Ahmad. "Quantified Parking - Comprehensive Parking Inventories for Five Major U.S. Cities". In: June 2018. URL: https://www.mba.org/2018-press-releases/july/riha-releases-new-report-quantified-parking-comprehensive-parking-inventories-for-five-major-us-cities.

[2] Yassine Alouini. *Lessons Learned from Kaggle's Airbus Challenge*. Jan. 2017. URL: https://medium.com/@YassineAlouini/lessons-learned-from-kaggles-airbus-challenge-252e25c5efac.

[3] Lokesh Boominathan, Srinivas S. S. Kruthiventi, and R. Venkatesh Babu. "CrowdNet: A Deep Convolutional Network for Dense Crowd Counting". In: *CoRR* abs/1608.06197 (2016). arXiv: 1608.06197. URL: http://arxiv.org/abs/1608.06197.

[4] Marc Canter. "Your AI is only as good as its Data". In: (Nov. 2016). URL: https://medium.com/ai-blogging/your-ai-is-only-as-good-as-its-data-cb8cb783f351.

[5] V. Coric and M. Gruteser. "Crowdsensing Maps of On-street Parking Spaces". In: *2013 IEEE International Conference on Distributed Computing in Sensor Systems*. May 2013, pp. 115–122. DOI: 10.1109/DCOSS.2013.15.

[6] Amélie Y. Davis et al. "Estimating parking lot footprints in the Upper Great Lakes Region of the USA". In: *Landscape and Urban Planning* 96.2 (2010), pp. 68–77. ISSN: 0169-2046. DOI: https://doi.org/10.1016/j.landurbplan.2010.02.004. URL: http://www.sciencedirect.com/science/article/pii/S0169204610000356.

[7] Arthur Douillard. *Object Detection with Deep Learning on Aerial Imagery*. July 2018. URL: https://medium.com/data-from-the-trenches/object-detection-with-deep-learning-on-aerial-imagery-2465078db8a9.

[8] Adam Van Etten. *Car Localization and Counting with Overhead Imagery, an Interactive Exploration*. Mar. 2017. URL: https://medium.com/the-downlinq/car-localization-and-counting-with-overhead-imagery-an-interactive-exploration-9d5a029a596b.

[9] Adam Van Etten. *Satellite Imagery Multiscale Rapid Detection with Windowed Networks*. Sept. 2018. URL: https://arxiv.org/pdf/1809.09978.pdf.

[10] Adam Van Etten. "Satellite Imagery Multiscale Rapid Detection with Windowed Networks". In: *CoRR* abs/1809.09978 (2018). arXiv: 1809.09978. URL: http://arxiv.org/abs/1809.09978.

[11] Adam Van Etten. *The Satellite Utility Manifold; Object Detection Accuracy as a Function of Image Resolution*. Apr. 2017. URL: https://medium.com/the-downlinq/the-satellite-utility-manifold-object-detection-accuracy-as-a-function-of-image-resolution-ebb982310e8c.

[12] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[13] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. Nov. 2018. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[14] Takamochi Minamoto Gou Koutaki and Keiichi Uchimura. "EXTRACTION OF PARKING LOT STRUCTURE FROM AERIAL IMAGE IN URBAN AREAS". In: *International Journal of Innovative Computing, Information and Control* 12.2 (2016). URL: http://www.ijicic.org/ijicic-1509-0005.pdf.

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[17] Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. Mar. 2018. URL: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

[18] Michael P Johnson. "Environmental Impacts of Urban Sprawl: A Survey of the Literature and Proposed Research Agenda". In: *Environment and Planning A: Economy and Space* 33.4 (2001), pp. 717–735. DOI: 10.1068/a3327. eprint: https://doi.org/10.1068/a3327. URL: https://doi.org/10.1068/a3327.

[19] Martina Marek Jordan Cazamias. "Parking Space Classification using Convoluional Neural Networks". In: (Aug. 1999). URL: https://cs231n.stanford.edu/reports/2016/pdfs/280_Report.pdf.

[20] Andrej Karpathy. *Convolutional Neural Networks (CNNs / ConvNets)*. URL: http://cs231n.github.io/convolutional-networks/#conv.

[21] Andrej Karpathy. *Transfer Learning*. URL: http://cs231n.github.io/transfer-learning/.

[22] Martijn B.W. Kobus et al. "The on-street parking premium and car drivers' choice between street and garage parking". In: *Regional Science and Urban Economics* 43.2 (2013), pp. 395–403. ISSN: 0166-0462. DOI: https://doi.org/10.1016/j.regsciurbeco.2012.10.001. URL: http://www.sciencedirect.com/science/article/pii/S0166046212000890.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[24] *Large Scale Visual Recognition Challenge (ILSVRC)*. URL: http://image-net.org/challenges/LSVRC/.

[25] Tsung-Yi Lin et al. "Feature Pyramid Networks for Object Detection". In: *CoRR* abs/1612.03144 (2016). arXiv: 1612.03144. URL: http://arxiv.org/abs/1612.03144.

[26] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[27] K. Liu and G. Mattyus. "Fast Multiclass Vehicle Detection on Aerial Images". In: *IEEE Geoscience and Remote Sensing Letters* 12.9 (Sept. 2015), pp. 1938–1942. ISSN: 1545-598X. DOI: 10.1109/LGRS.2015.2439517.

[28] Helmut Mayer. "Automatic Object Extraction from Aerial Imagery—A Survey Focusing on Buildings". In: *Computer Vision and Image Understanding* 74.2 (1999), pp. 138–149. ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.1999.0750. URL: http://www.sciencedirect.com/science/article/pii/S1077314299907506.

[29] Andrew Myers. *An artificial intelligence algorithm developed by Stanford researchers can determine a neighborhood's political leanings by its cars*. Oct. 2017. URL: https://news.stanford.edu/2017/11/28/neighborhoods-cars-indicate-political-leanings/.

[30] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[31] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[32] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[33] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[34] Oriol Pujol Santi Seguí and Jordi Vitriá. “Learning to count with deep object features”. In: *CoRR* abs/1505.08082 (2015). arXiv: 1505.08082. URL: http://arxiv.org/abs/1505.08082.

[35] Donald C. Shoup. “Cruising for parking”. In: *Transport Policy* 13.6 (2006). Parking, pp. 479–486. ISSN: 0967-070X. DOI: https://doi.org/10.1016/j.tranpol.2006.05.005. URL: http://www.sciencedirect.com/science/article/pii/S0967070X06000448.

[36] *SpaceNet Challenge*. URL: https://spacenetchallenge.github.io/.

[37] Sebastian Türmer et al. “Evaluation of selected features for car detection in aerial images”. In: *ISPRS Hannover Workshop 2011*. June 2011, pp. 1–6. URL: https://elib.dlr.de/70366/.

[38] Chuan Wang et al. “Deep People Counting in Extremely Dense Crowds”. In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM ’15. Brisbane, Australia: ACM, 2015, pp. 1299–1302. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806337. URL: http://doi.acm.org/10.1145/2733373.2806337.

[39] Gui-Song Xia et al. “DOTA: A Large-scale Dataset for Object Detection in Aerial Images”. In: *CoRR* abs/1711.10398 (2017). arXiv: 1711.10398. URL: http://arxiv.org/abs/1711.10398.

[40] Michael Ying Yang et al. “Vehicle Detection in Aerial Images”. In: *CoRR* abs/1801.07339 (2018). arXiv: 1801.07339. URL: http://arxiv.org/abs/1801.07339.

[41] Nathan Ratliff Young-Woo Seo and Chris Urmson. “Self-Supervised Aerial Image Analysis for Extracting Parking Lot Structure”. In: (Aug. 2018). URL: https://www.ijcai.org/Proceedings/09/Papers/305.pdf.

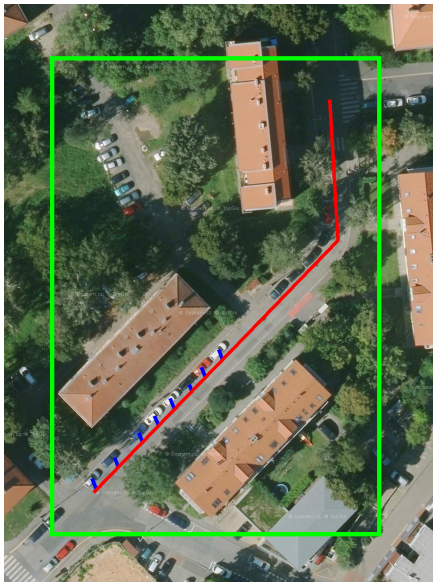
[42] David Yu. *Parking Lot Vehicle Detection Using Deep Learning – GeoAI – Medium*. Aug. 2018. URL: https://medium.com/geoai/parking-lot-vehicle-detection-using-deep-learning-49597917bc4a.

[43] Tao Zhao and Ram Nevatia. “Car detection in low resolution aerial images”. In: *Image and Vision Computing* 21.8 (2003), pp. 693–703. ISSN: 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(03)00064-7. URL: http://www.sciencedirect.com/science/article/pii/S0262885603000647.

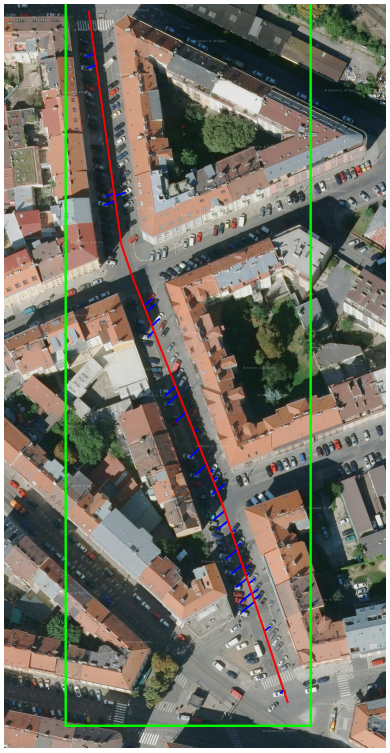
Appendix A

More Experiment Results

Figure A.1: More streets with on street parking showing datacted cars with their classification and predicted parking capacity.



(a) Vršní, classified as one-side parking street with capacity 19 cars, real capacity ∼16 cars.



(b) Rostislavova, classified as both-side parking street with capacity 110 cars, real capacity ∼95 cars.

(c) V Domcích, classified as one-side parking street with capacity 14 cars, real capacity ∼13 cars.

(d) Wichterlova, wrongly classified as both-side parking street with capacity 36 cars, real capacity ∼26 cars.

Appendix B

DVD Contents

This chapter contains a list of files that are included on the disc.

/**src.zip** Archive with source codes.

/**thesis.pdf** Thesis in PDF format.

/**thesis_src.zip** Archive with source codes of this thesis.