BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

URBAN ELEMENT DETECTION USING SATELLITE IMAGERY

HLEDÁNÍ MĚSTKÝCH PRVKŮ V SATELITNÍCH SNÍMCÍCH

BACHELOR’S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

DÁVID ORAVEC

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Arch. ADAM ZLÁMAL,

BRNO 2021

Department of Computer Graphics and Multimedia (DCGM) Academic year 2020/2021

Bachelor's Thesis Specification

23945

Student:	**Oravec Dávid**
Programme:	Information Technology
Title:	**Urban Element Detection Using Satellite Imagery**
Category:	Image Processing

Assignment:

1. Study computer vision, the use of neural networks on 2D data.
2. Search for, prepare and process a suitable dataset.
3. Select the elements you want to detect - city gaps, street parking, parks.
4. Experiment with computer vision and machine learning algorithms, find a suitable approach.
5. Evaluate the created solution on the dataset, further expand the dataset and iteratively improve the solution.
6. Demonstrate the capabilities of the solution achieved.
7. Evaluate the achieved results and the possibilities of continuing the project.

Recommended literature:

- Andrew Ng: Machine Learning Yearning, 2018
- Eli Stevens, Luca Antiga, and Thomas Viehmann: Deep Learning with PyTorch, Manning Publications, 2020
- Ian Goodfellow: Deep Learning, The MIT Press, 2016

Requirements for the first semester:

- Items 1 to 3, elaboration of items 4 and 5.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Zlámal Adam, Ing. Arch.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2020
Submission deadline:	May 12, 2021
Approval date:	October 30, 2020

Abstract

This thesis focuses on the right detection of objects in satellite imagery using convolutional neural networks. The goal of the thesis is to detect swimming pools and tennis courts in satellite imagery from different cities using the trained model. The model works with data from 10 different cities. The RetinaNet neural network model and Detectron2 library were used for development. The final trained model can detect objects with the average precision (AP50) at the level of 63.402 %. The thesis can be useful in the field of automating the acquisition of land surface statistics.

Abstrakt

Táto práca sa zameriava na správnu detekciu objektov v satelitných snímkach pomocou konvolučných neuronových sietí. Cieľom práce je pomocou natrénovaného modelu detekovať bazény a tenisové ihriská v satelitných snímkach z rôznych miest. Model pracuje s dátami z 10 rôznych miest. Pri vypracovaní bol využitý model neurónovej siete RetinaNet a knižnica Detectron2. Model, ktorý sa podarilo vytrénovať, dokáže detekovať objekty s priemernou presnosťou (AP50) na úrovni 63,402 %. Práca môže byť prínosom v oblasti automatizovania získavania štatistík o povrchu zeme.

Keywords

computer vision, object detection, image classification, instance segmentation, semantic segmentation, convolutional neural networks, RetinaNet, Detectron2

Kľúčové slová

počítačové videnie, detekcia objektov, klasifikácia obrazov, segmentácia inštancií, sémantická segmentácia, konvolučné neurónové siete, RetinaNet, Detectron2

Reference

ORAVEC, Dávid. *Urban Element Detection Using Satellite Imagery.* Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Arch. Adam Zlámal,

Rozšírený abstrakt

Úvod

S prvkami počítačového videnia sa ľudstvo stretáva každý deň. So vzostupom umelej inteligencie prichádzajú nové inovácie, ktoré v mnohých prípadoch zjednodušujú život. Príkladom môžu byť inteligentné semafóry na križovatkách, rozpoznávanie tvárí, evidenčných čísel vozidiel na parkovisku v obchode alebo napríklad automatická detekcia lopty v športovom prenose naživo. To je len zlomok všetkých vymožeností, ktoré prináša umelá inteligencia.

Táto práca sa dotýka práve odvetvia umelej inteligencie a strojového učenia. Konkrétne ide o počítačové videnie. Existujú rôzne prístupy a techniky v počítačovom videní. Najznámejšími a najpoužívanejšími sú klasifikácia obrazu, detekcia objektov, segmentácia obrazu či segmentácia jednotlivých inštancií.

V práci sa zameriavam na detekciu objektov v obraze, presnejšie o detekciu objektov v satelitných snímkach. V mojom prípade ide o detekovanie bazénov a tenisových kurtov. Vytvoriť správny detektor objektov je náročná úloha, pri ktorej záleží na mnoho faktoroch. Ide napríklad o rozlíšenie obrázku, hustotu objektov v obraze, farebné zloženie obrazu, prekryv jednotlivých objektov, atď.

Cieľom tejto práce je vytvoriť model pomocou konvolučnej neurónovej siete, ktorý dokáže detekovať bazény a tenisové kurty v satelitných snímkach. Zhotovenie kvalitného detektoru by pomohlo výrazne zrýchliť mapovanie a analýzu zemského povrchu.

Riešenie

Na riešenie tohto problému bola využitá konvolučná neurónová sieť. Konkrétne bakalárska práca využíva model RetinaNet. Na vytrénovanie modelu bolo potrebné získať dostatočný počet satelitných snímkov z rôznych oblastí a vytvoriť kvalitnú dátovú sadu. Prvotná dátová sada bola zložená z voľne dostupných satelitných snímkov mesta Austin v Texase. Na doplnenie a získanie dát aj z ďalších oblastí boli pridané satelitné snímky z miest Bellingham, Bloomington, Chicago, Innsbruck, Kitsap, San Francisco, Viedeň a zo západnej a východnej časti rakúskej spolkovej krajiny Tirol. Tieto satelitné snímky sú súčasťou projektu Inria, ktorý je voľne dostupný. Po získaní dátovej sady bola sada rozdelená na trénovaciu, validačnú a predikčnú. Na zväčšenie objemu dát v trénovacej sade bola použitá augmentácia.

Tréning modelu prebiehal v online prostredí Google Colab za pomoci knižnice Detectron2. Po vytrénovaní modelu bola vyhodnotená aj jeho kvalita na validačnej dátovej sade. Na základe tohto vyhodnotenia mohol byť model iteratívne vylepšovaný. Najlepší model bol následne použitý na konečné predikcie, ktoré sa uskutočnili na predikčnej dátovej sade. Táto predikčná sada neobsahovala žiadne značky (obdĺžniky okolo objektov) narozdiel od trénovacej či validačnej dátovej sady.

Zhodnotenie výsledkov

V prvej polovici práce model vykazoval presnejšie výsledky, avšak pracoval iba s jednou dátovou sadou. V druhej polovici práce sa do tejto dátovej sady pridalo ďalších 9 miest a presnosť detekcie klesla o niekoľko desiatok percent. Postupným vylepšovaním modelu sa presnosť detekcie zvyšovala, a po rozsiahlom trénovaní a experimentovaní s modelom, boli nakoniec dosiahnuté výsledky, v ktorých model vykazoval priemernú presnosť (AP50) na úrovni 63,402 %. S týmto modelom boli vykonané aj konečné predikcie na neoznačených dátach.

Presnosť modelu by mohla byť do budúcna vylepšená rôznymi modifikáciami dátovej sady. Pravdepodobne najväčší nárast presnosti by nastal v prípade, kedy by trénovacia dátová sada obsahovala iba obrázky, na ktorých je aspoň jedna inštancia objektu. V mojom prípade by to znamenalo, že v trénovacej sade by sa nachádzali len obrázky, na ktorých by bol aspoň jeden z dvojice bazén - tenisový kurt. Ďalšou možnosťou pre zvýšenie presnosti je detekcia iba jednej triedy objektu. V mojej práci detekujem bazény a tenisové kurty. V prípade zamerania sa iba na jednu z týchto tried by model mohol vykazovať vyššiu presnosť.

Urban Element Detection Using Satellite Imagery

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Arch. Adam Zlámal. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Dávid Oravec
May 9, 2021

Acknowledgements

Firstly, I would like to thank my supervisor Ing. Arch. Adam Zlámal for his guidance, help, and tutor advice whenever I needed it. Secondly, I would like to thank my family and girlfriend for their support and for creating good conditions for writing this thesis.

Contents

Chapter 1

Introduction

Elements of computer vision can be found in everyday life. Intelligent traffic lights or recognition of license plates at the entrance to the parking lot in the shopping center, automatic detection of a hockey puck or a ball in sports live streaming, payment by QR code in banking are only several examples of common things using computer vision. People can make work and life easier thanks to computers and artificial intelligence.

This thesis mainly focuses on one sphere of computer vision, namely, object detection. The goal is to identify swimming pools and tennis courts in satellite imagery. By object detection, I mean the following: given an image that contains a swimming pool or a tennis court, I want to output a bounding box (rectangle), which encloses the object as tightly and precisely as possible. Based on this detection, the approximate incidence of objects can be determined in a designated area without the necessity of a physical visit.

This detection can be done by training a neural network. Neural networks are part of artificial intelligence in computer science. In this thesis, I use a special type of neural network called a convolutional neural network. This type of network is mostly used to analyze images or videos.

To create this kind of a model, I needed to provide suitable data for my network. Therefore, I used a dataset that contains over 20 000 satellite images.

In the chapter 2 are compared computer vision techniques which are associated with this thesis. All of the most used nowadays computer vision techniques like object detection, image classification, instance, and semantic segmentation are described there. Chapter 3 is more theoretical. It discusses neural networks, especially convolutional neural networks and their architecture, individual layers, as well as their functionality in convolutional networks. Next, there can be also found information and difference between two-stage and one-stage detectors with examples like R-CNN, Fast R-CNN, and Faster R-CNN. At the end of the chapter, I talk about the one-stage detector RetinaNet, which I used in my thesis. The practical part of my thesis starts in chapter 4 where I describe the creation of my dataset and its contents. I also mention how I enlarged my dataset using different augmentation types and how I optimized my dataset compared to the referential one. After that, I shortly explain the used technologies such as Python and its libraries which played a significant role in my thesis, or Google Colab where I developed my model. The process of implementation and its details are described in chapter 5. Final predictions, where my trained model scans nonlabeled images and predicts the occurrence of swimming pools and tennis courts are summarized in chapter 6. The conclusion of my thesis and future work is summarized in chapter 7.

Chapter 2

Computer vision techniques

In this chapter, I will go through the most used computer vision techniques for image recognition, their usability, and differences. There are many techniques, but in this thesis, I had to choose one of the four most used ones – image classification, object detection, semantic segmentation, and instance segmentation. I wanted to count the instances of a class in my thesis, so after the study of all these computer vision techniques and a consultation with my supervisor, I decided that object detection will be the best approach.

In general, computer vision focuses on designing computer systems that have the ability to capture, understand, and interpret significant visual information within an image or video data like humans can [25]. The brains in the background of computer vision are comparable to humans' brains. For visual information processing, computers use Convolutional Neural Network (CNN) which is made from neurons. A computer neuron is a mathematical function that models the functioning of a biological neuron.

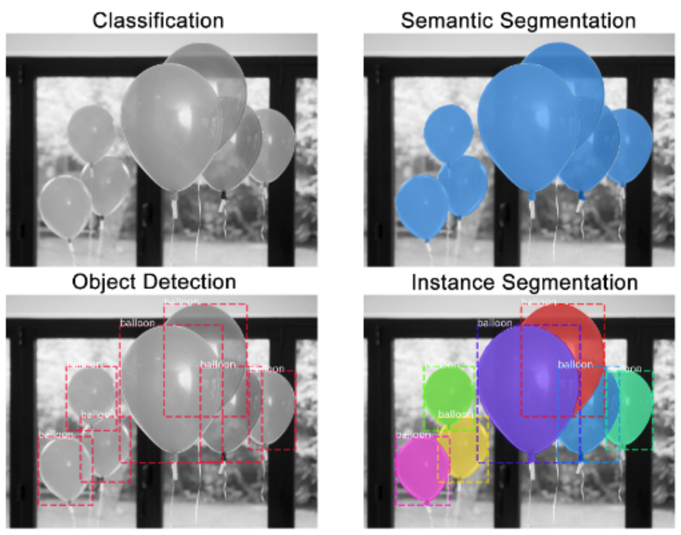


Figure 2.1: Visual comparison of the most used computer vision techniques. Image taken from [8].

2.1 Image classification

Image classification can be defined as a process of categorizing and labeling pixels or groups of them or vectors within an image derived from certain rules [36]. The reason why image classification came into existence was to tighten the gap between computer vision and human vision. This can be reached by training the computer with the data [18]. For example, when shown an image, the human can precisely say what is in that image in a second. The computer has to analyze this image and classify the class where it belongs.

To achieve image classification, the image has to be differentiated into the prescribed category based on the content of the vision. A simple algorithm for image classification can look like this [20]:

- given an input – training dataset – of N images, each labeled with one of K classes
- train a classifier with this training set, to learn what every class looks like
- use trained classifier to predict labels on a new set of images it has never seen before
- compare true labels of images with predicted labels by a classifier

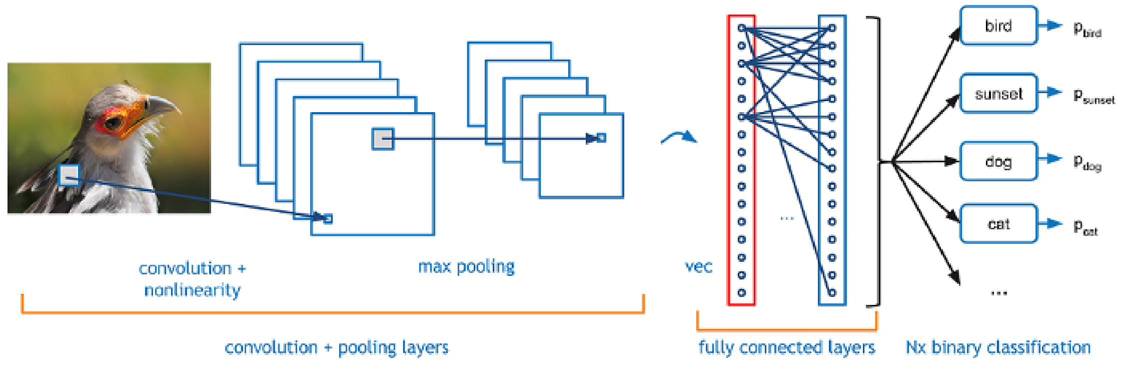


Figure 2.2: The working pipeline of image classification. The image was taken from [20].

As the name machine learning can indicate, the most crucial part is learning. Not only in the context of image classification but also in general, these learning techniques can be divided into three groups [32]:

- **supervised** – the classification class is assigned by a supervisor (user)
- **unsupervised** – the classification class is assigned automatically by a computer
- **semi-supervised** – it is a combination of the previous two techniques when some of the objects are assigned to classes by the supervisor and some of them are assigned by the computer

For example, in Figure 2.1 computer can classify objects into the class named **balloon**.

2.2 Object detection

Unlike image classification, object detection is extraordinarily useful in identifying locations of objects within an image, which can additionally be used for counting detected instances. To be precise, object detection draws bounding boxes around the desired detected objects, which makes it possible to locate a static or moving object through the given image or scene.

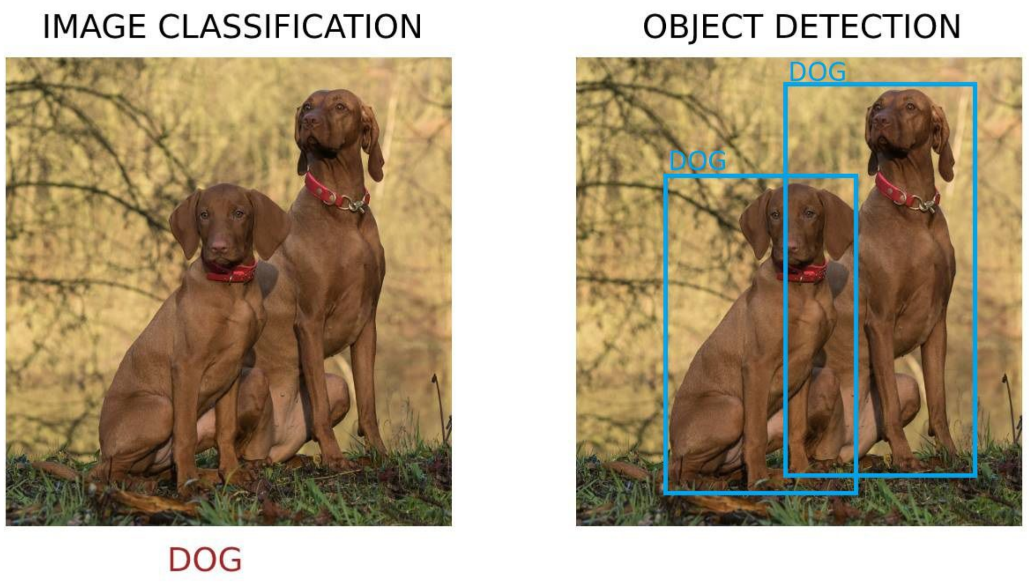


Figure 2.3: Difference between the previous technique image classification and object detection. In object detection, each object is labeled by its own class and bounding box is drawn around.

In image classification, the class is assigned to the object. In this case, it is the class dog, as it can be seen in Figure 2.3. In object detection, bounding boxes are drawn around the objects and each box is marked with its own class (label).

Like in other machine learning tasks, the first step in building a good detection model is to gather suitable images. Furthermore, it is necessary to create labeled data in the desired dataset. Labeled data in this context corresponds to bounding box coordinates around the object in an image and its label (class).

Object detection helps in various branches such as crowd counting, face recognition, anomaly detection, video surveillance, etc. It is also usually combined with other computer vision techniques.

2.3 Semantic segmentation

Semantic segmentation deals with the task of classifying each pixel of the image in a predefined class or set of classes. The difference from object detection is that semantic segmentation does not predict any bounding boxes around the objects. Moreover, it does not classify different instances of the same class [8]. Example output of semantic segmentation can be seen in Figure 2.4.

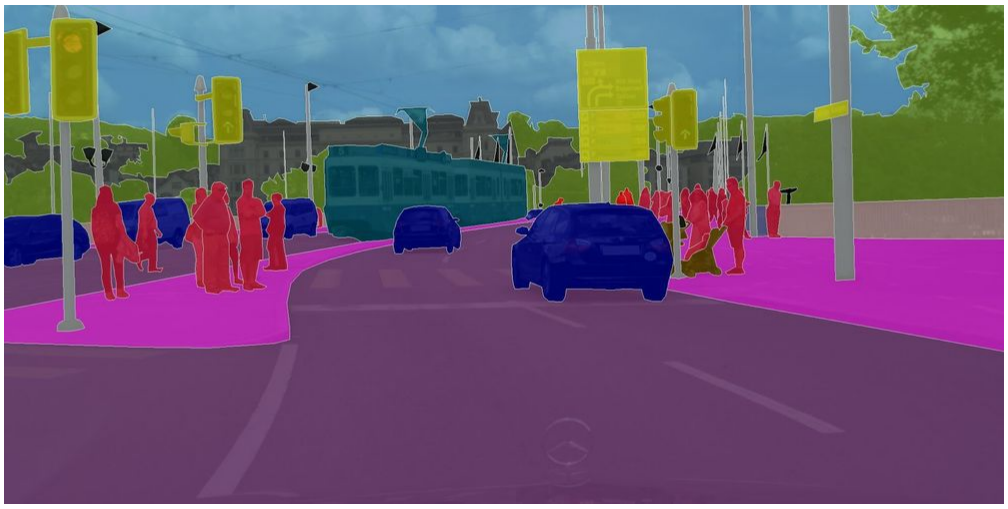


Figure 2.4: Objects from the same class are masked with the same color. For example, vehicles on the road are blue, nature is green, buildings are gray, the road is purple, etc... This image can serve as an example of an autonomous system – a self-driving car. The image was taken from [16].

The difference between semantic segmentation and instance segmentation is that in instance segmentation, the different objects of the same class will be labeled differently. In Figure 2.5 can be seen the major difference between these two techniques. The class person is divided into five differently labeled instances of a person, which each hence a different color in case of instance segmentation.

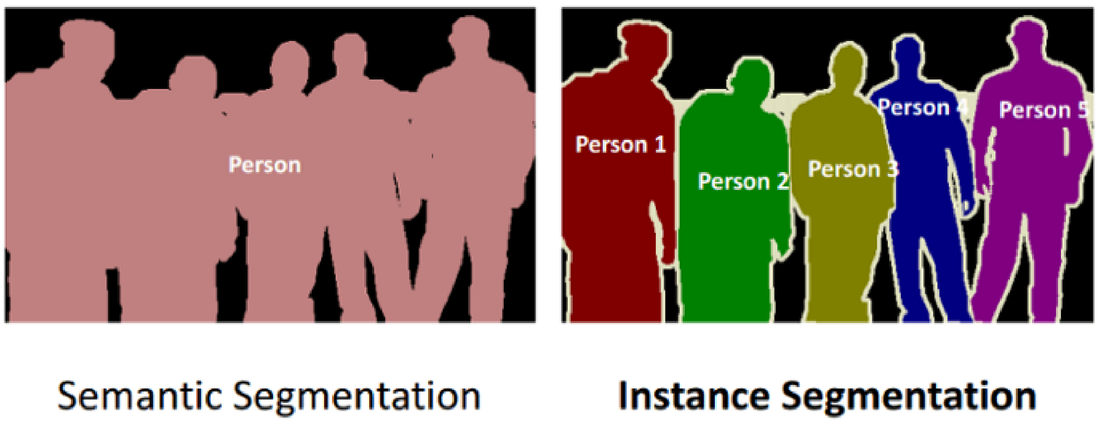


Figure 2.5: Comparison of instance and semantic segmentation. On the one hand, semantic segmentation labels objects of the same class by one label *person*. On the other hand, instance segmentation labels each object by unique label. The image was taken from [1].

2.4 Instance segmentation

The last computer vision technique which I encountered is instance segmentation. It is a task that combines object detection and semantic segmentation. At the same time, when the object is detected in an image, it generates a segmentation mask of the object.

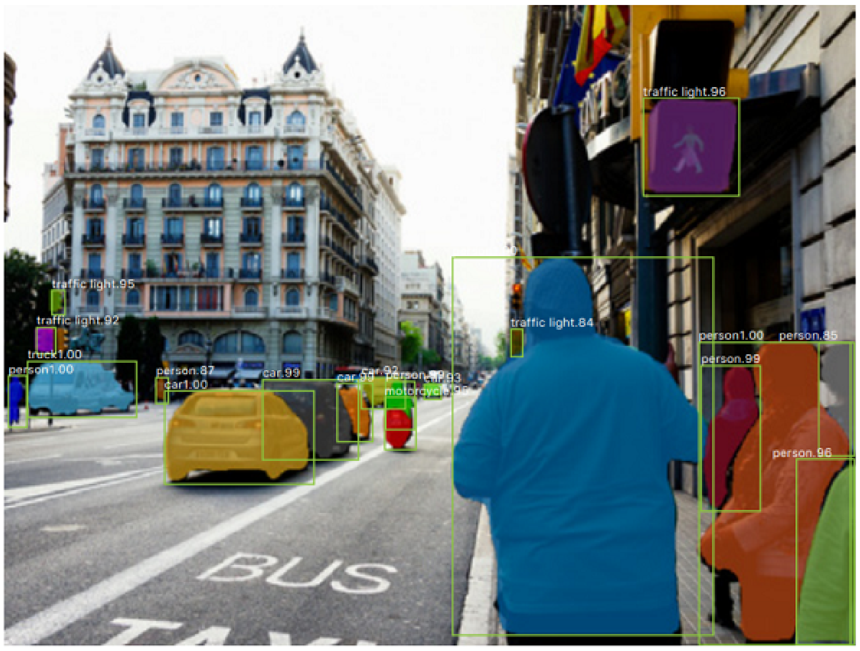


Figure 2.6: The example of instance segmentation using the Mask R-CNN deep learning model. As it can be seen, it is a combination of object detection (bounding boxes around the objects) and semantic segmentation. The image was taken from [13].

Since this task combines two other computer vision techniques, it can be arguably considered the most challenging computer vision task. One of the top methods for instance segmentation is Mask R-CNN and its variants. Instance segmentation has the best appliance in autonomous cars, because it detects individual objects and is able to separate them even if they touch or overlap.

Chapter 3

Neural networks and object detection

This chapter is a theoretical part describing neural networks and their usage in object detection. At the beginning of the winter semester, I just briefly got acquainted with neural networks, because I tried to build the first model as fast as possible. I got this theoretical background for my thesis in the second half of the school year.

The types of neural networks can be different, but in this thesis, I use a convolutional neural network. In this chapter, I first describe convolutional neural networks in general, then I describe their architecture and individual layers. Next, I describe object detection algorithms and the types of two-stage and one-stage detectors. At the end of the chapter, I explain the RetinaNet network, which I use in my thesis.

The information about individual layers were primarily gathered from the paper by O'Shea and Nash [27].

3.1 Convolutional neural networks

Convolutional Neural Network (CNN) is a special type of neural network for processing data, which has a known grid-like topology. The name convolutional network comes from mathematical convolution, which CNNs use. Convolution is a linear mathematical operation that combines two functions to produce a third one. It is widely used and has many other practical applications other than in neural networks. At point x, y, the convolution can be defined as:

$$c(x, y) = f * g(x, y) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] g[x - n_1, y - n_2], \tag{3.1}$$

where $c(x, y)$ stands for the result of convolution at the point (x, y), f stands for convolution filter, $*$ represents convolution operation and $g(x, y)$ stands for input image at coordinates (x, y). The tenet of Equation 3.1 is shown in Figure 3.2.

The main element of the neural network is a neuron. It is a linear transformation of the input which is followed by a fixed nonlinear function and its application (also known as *activation function*) [34]. In CNNs, these neurons are arranged in 3 dimensions:

- width – *image width*,
- height – *image height*,

- depth – *color channel of image.*

The architecture of CNN is described in section 3.2.

3.2 Architecture of CNN

CNN is built from layers of three types, namely, convolutional layer, pooling layer, and fully-connected layer. Once these three layers are piled, CNN is formed.

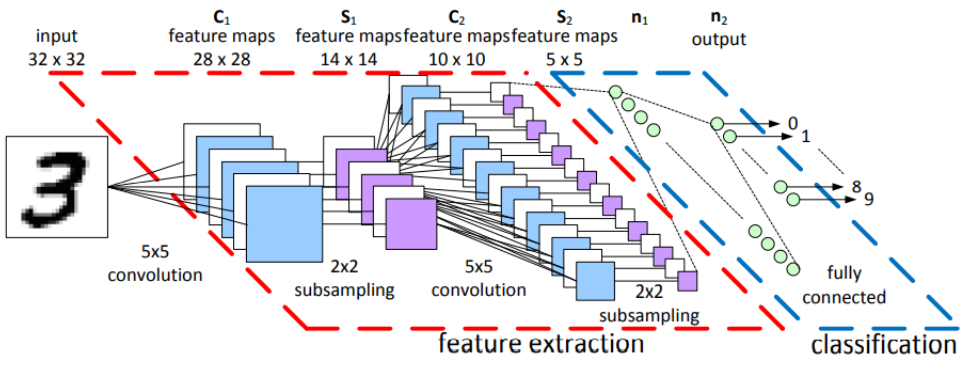


Figure 3.1: The architecture of CNN for digit recognition. The network is piled from two convolution layers with kernels of 5×5 dimensions, two pooling layers that are noted as “subsampling” and a fully-connected layer at the end for classification of an input image. The image was taken from [28].

3.2.1 Convolutional layer

Convolutional layer is the main layer of CNN, which provides the extraction of features from the input layer. In the context of my thesis, the first function of convolution is an input image, whilst the second one is a 2D filter (matrix, also known as *kernel*). The process of convolution retains spatial relations among pixels, thanks to consecutive application onto small square areas in the input image. The technique of sliding window is used to cover the whole input image. The use of a smaller kernel than the input image makes it possible to detect smaller features in large-scale images.

The results of convolution are placed into the final matrix which is called a feature map. Figure 3.2 shows the process of convolution and sliding window.

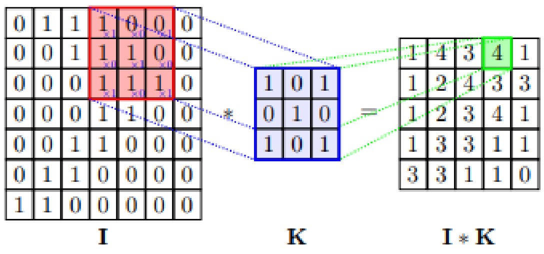


Figure 3.2: Example convolution of filter (kernel) K with a receptive field in the image I. Its output after convolution is one node (green square) in the final feature map $I * K$. Picture was taken from [3].

In a convolutional neural network, each neuron in a layer can share the value of parameters. This technique is called parameter sharing. It operates on the presumption that if a feature is good to use in one region, it is most likely going to be useful also in another region. That is the instance where the common neural network and convolutional neural network differ.

Another important element of convolutional layer is *activation function*. As it is stated in paper by Sharma and others [33], these functions are used to transform input signal into an output signal which is provided to the next layer. Also, it is necessary to add non-linearities into network, if we plan to process non-linear data. There are multiple options of activation functions, for example *sigmoid function*, *binary linear function*, *tanh function*[1], etc..

The most used activation function in CNN is ReLU (Rectified Linear Unit). Its biggest advantage is that not all neurons in the network are active at the same time. It means that the neuron is going to be deactivated when the output of ReLU is 0. Mathematical definition of ReLU is:

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

[1]More information about these functions can be found in paper by Sharma and others [33]

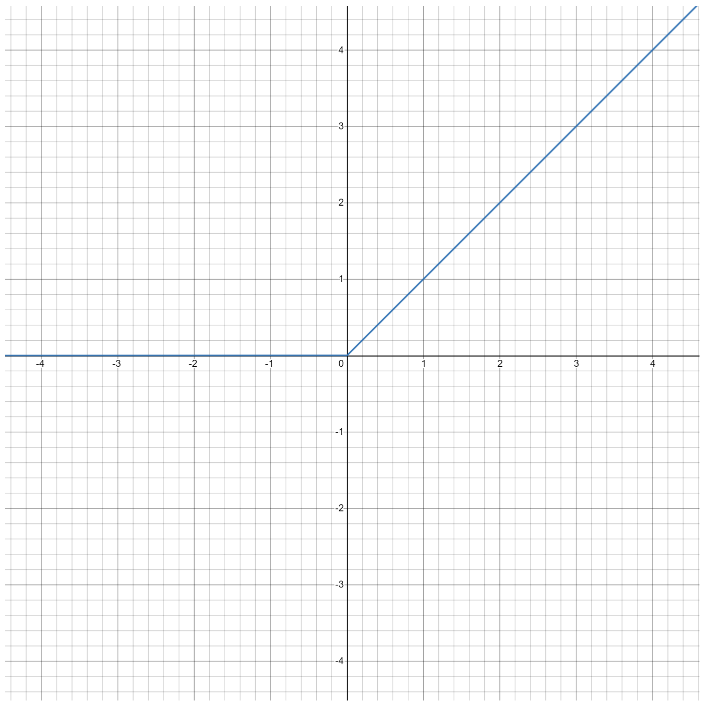


Figure 3.3: Graph of the ReLU function.

ReLU is more efficient than other activation functions, thus is the most widely spread in CNNs.

3.2.2 Pooling layer

The main purpose of this layer is to consecutively reduce the spatial dimensionality. It combines a group of neurons from the previous layer into one neuron to reach this goal.

The most used approach for value combination is a ***max*** function. In most CNNs, this function uses kernels with dimensionality 2×2 and stride with a size of 2.

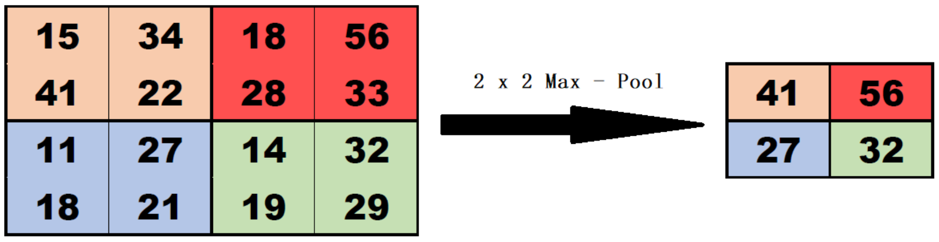


Figure 3.4: Example of max pooling with 2×2 dimensions. Maximum is taken over four numbers with a stride of 2.

Another example of pooling is *average pooling*. In this case, the average of the receptive field is calculated and saved. Other than this, the L_2 norm of a rectangular neighborhood or weighted average based on the distance from the central pixel are also popular.

Nevertheless, pooling helps to make the representation approximately invariant to small translations of the input in all cases above. The meaning of *invariance to translation* is that if the input is translated by a small amount, the values of most of the pooled outputs do not change [12].

The biggest advantages of pooling are the increase in computing speed and lowering memory usage.

3.2.3 Fully-connected layer

This layer is usually at the end of the neural network to produce classifications. This is where all outputs from previous layers are connected to all neurons in this layer. The output of a fully-connected layer is a feature vector. It is calculated as a scalar product of input values.

3.3 Object detection algorithms

There are plenty of object detection algorithms, which can be used for image processing. In this section, I will go through some of them, which I encountered during my work on this thesis. Furthermore, these are the most recent methods in object detection.

They can be divided into two groups based on the style of designing regions where the objects could be.

3.3.1 Two-stage detectors

As the name says, these detectors have two stages. In the first stage, sparse region proposals are generated. These regions should contain all possible objects in the image. The creation of region proposals differs among each method. In the second stage, regions are regressed and classified. They can be classified either as background or one of the foreground classes (pool, car, dog, etc.) [23] [2].

On the one hand, this two-stage approach is very slow, but on the other hand, it produces the best accuracy.

R-CNN

Regions with CNN Features (R-CNN) is an object detection model that uses high-capacity CNNs for bottom-up region proposals. This process is necessary for localizing and segmenting images. To identify region candidates (also called as *regions of interest*) and number of bounding boxes around objects, it uses *selective search* [35] and then it extracts features from each region independently for classification [10].

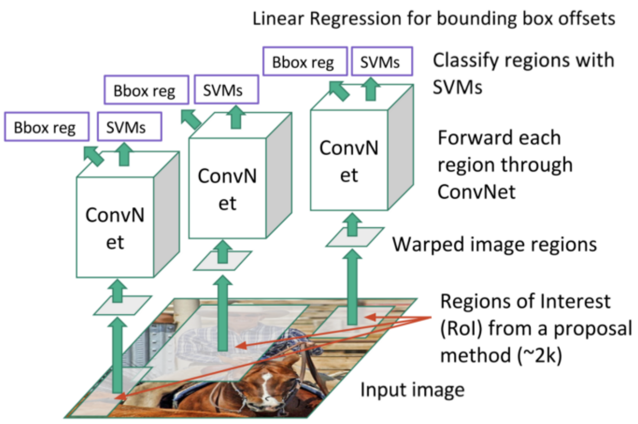

Linear Regression for bounding box offsets
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Figure 3.5: The pipeline of R-CNN. The image was taken from [20].

This whole process starts with scanning the input image for bounding boxes (region proposals). By using a selective search algorithm, these regions are generated and on top of each of them, CNN does its work. The output of each CNN is fed into a super vector machine (SVM). The main function of SVM is to classify whether the object is presented and if yes, what object it is. At the end of this process, there is a linear regression. Its function is to tighten and correct generated bounding boxes.

Although this is an approach that works well, it also has some disadvantages. Training of R-CNN is very slow. It is caused by forward passing, which is needed for every region proposal, and R-CNN commonly generates approximately 2000 of them. Another downside is that R-CNN requires a lot of disk space and its inference is also slow.

Fast R-CNN

The creator of R-CNN, Ross Girschik, came up with a new upgraded version of it. The idea still remains the same, but Fast R-CNN solves mainly the speed of a process.

Fast R-CNN takes the entire image and runs CNN on top of it, instead of running on top of each region. To fasten up the process, a new technique was introduced in Fast R-CNN. This technique is called *Regions of Interest* (RoI) pooling. It shares a forward pass of CNN for an image across its subregions, which helps to decrease the time of the whole process. To prevent the creation of a new model, SVM was replaced with a softmax layer. This layer extends the network for predictions. Additionally, softmax encapsulates regression, which means Fast R-CNN uses just one model instead of the previous three models (CNN, SVM, regressor) [2][9].

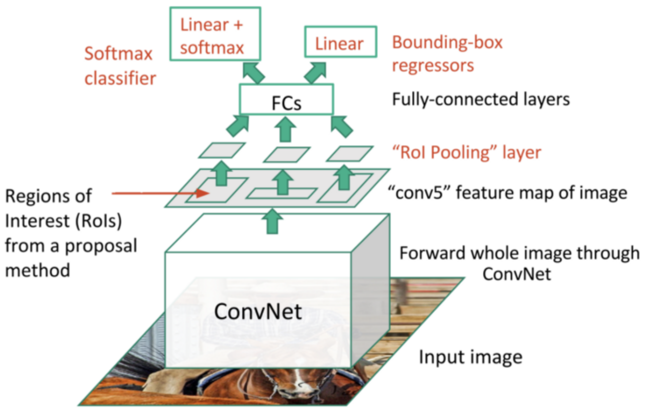


Figure 3.6: Inference pipeline of Fast R-CNN. CNN features are calculated just once and after that are shared among all RoI. The image was taken from [20].

Faster R-CNN

All of these inventions and improvements helped to fasten up the original R-CNN, but something was still missing. It was not fast enough to provide real-time recognition. Researchers found out that the biggest stumbling block was how region proposals were generated. In other words, they had to replace the selective search algorithm [30]. That is, how Faster R-CNN was created.

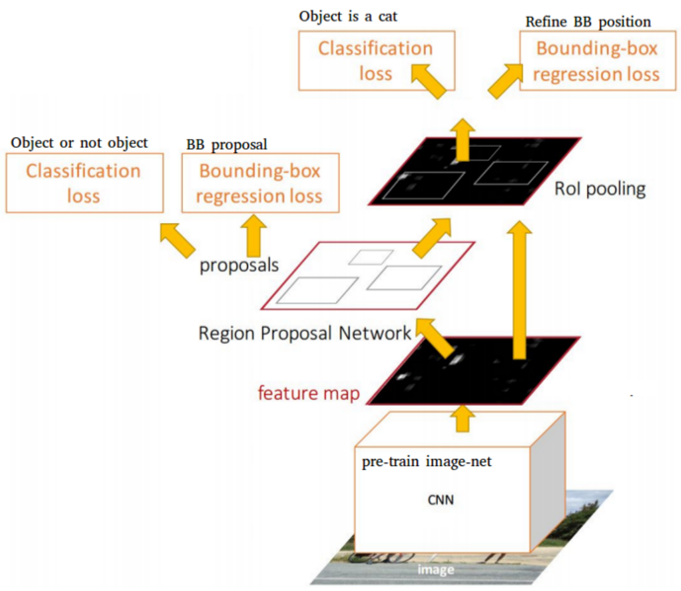


Figure 3.7: The pipeline of Faster R-CNN, which is very similar to Fast R-CNN, but region proposals are produced by the neural network, not by an algorithm. The image was taken from [20].

Faster R-CNN adds a new fully convolutional neural network to predict proposals from features. Its name is simple – Region Proposal Network (RPN). It uses a sliding window that is shifting across each feature map. At each position, it computes and outputs the number of potential bounding boxes. Furthermore, each bounding box has a prediction score of how this box is expected to be correct. These boxes have different shapes and sizes (rectangles, squares, etc.). After this process of generating bounding boxes, they are fed into Fast R-CNN to make a classification and adjust those bounding boxes.

3.3.2 One-stage detectors

As it was mentioned in 3.3.1, two-stage detectors generate only sparse region proposals. That is the main difference from one-stage detectors. One-stage detectors generate a dense set of regions, which cover the whole image area [23]. For my thesis, I chose to use the one-stage detector RetinaNet [22]. It has better speed and accuracy results than all of the two-stage detectors mentioned in section 3.3.1.

3.4 RetinaNet

This single unified network is composed of one backbone network and two task-specific subnetworks. The backbone network is practically a convolutional network which function is to perform computing a convolutional feature map over the whole input image. Also, subnets have their own functions. The first one has to perform an object classification from the backbone's output and the second one performs bounding box regression [22]. Components of RetinaNet are further described in the next sections.

Feature pyramid network

This is the backbone network that was mentioned in 3.4. The feature pyramid network (FPN) is a structure for multiscale object detection. It contains multiple levels that serve as a classifier and regressor. FPN produces feature maps of different sizes and scales on these levels. The detailed description of FPN can be found in paper by Lin, Dollár, Girschik and others [21].

The RetinaNet uses ResNet as a backbone network. FPN is placed on top of the backbone network. RetinaNet uses a modified version of FPN. The high-resolution pyramid layer P2 was removed because of its computational demands. Pyramid layer P6 uses stridden convolution for computation instead of downsampling and pyramid layer P7 was added to improve large object detection. These improvements of FPN provided better results while maintaining the accuracy [22].

Anchors

Anchors in RetinaNet work similarly as anchors, for example, in Faster R-CNN. To remind, in Faster R-CNN on top of the convolutional layer, there is a sliding window with predefined scale and aspect ratios (anchor). Features for each of these anchors are sent for regression and classification. Anchors in RetinaNet are different because they are slid not only on a single-scaled feature map, but they group heads with the same design on each pyramid level. That means this setup makes possible the detection of objects with different scales even with one-sized anchors. According to the paper by Lin, Girshick, Dollár and others [22], the usage of multi-sized anchors together with the feature pyramid produced better results. To be precise, anchors with three aspect ratios `{1:2, 1:1, 2:1}` and three sizes $\{2^0, 2^{1/3}, 2^{2/3}\}$ were added at each level. This setting improved the average precision.

Each pyramid level has 9 anchors and is capable to detect objects with a scale range of 32 - 813 pixels concerning the network's input image.

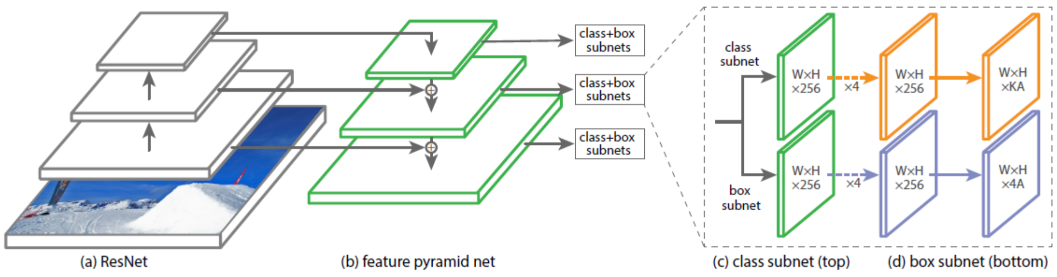


Figure 3.8: The architecture of RetinaNet. The Figure shows a convolutional network (a) next to feature pyramid network (b) and two subnets for classification (c) and regression (d). The image was taken from [22].

Classification subnet

This subnet is a small fully-connected network that is attached to each level of the feature pyramid network. Its function is to predict the probability of object occurrence at each spatial position for each of the object classes and anchors. This subnet shares parameters across all pyramid levels. The workflow can be seen in Figure 3.8 image (c). As an input, it takes a feature map from one of the pyramid levels and performs a convolutional operation followed by application of an activation function. Then it outputs a prediction for each of the object classes and anchors [22].

Regression subnet

Regression subnet is defined as a parallel subnet to the classification one. Both of these subnets are almost identical. The difference is that the regression subnet produces four linear outputs for each of the anchors. These numbers show a predicted offset between the ground-truth bounding box and the anchor.

Chapter 4

Solution design

In this chapter, I will introduce the whole process of making this bachelor's thesis. I chose to use a top-down approach to this thesis. It means that I tried to have a type of model as soon as possible to start my experiments and based on their results I could improve my model. I studied the theoretical background of computer vision and machine learning in the summer semester while writing the textual part of my thesis. After studying some basics of computer vision, machine learning, and neural networks, the first step was to search for a suitable dataset.

4.1 Dataset

Data is a significant element in nowadays era, especially in computer vision. It is necessary to provide quality data to make a powerful detection model. The more, the better.

I was deciding between making a brand new custom dataset, which would cover some other city, or finding a free open-source dataset, which would meet my requirements and ideas. I searched for datasets at desired websites like Kaggle[1] or Dataset Search from Google[2].

4.1.1 Austin Zoning Satellite Images

The first dataset that I used and trained my first detection model on was the Austin dataset[3]. It contains 3666 satellite images of Austin, the capital city of state Texas, United States of America. This dataset was a part of a project that classified small parts of the city into their corresponding zones (e.g., residential, industrial, etc.).

[1]https://www.kaggle.com/
[2]https://datasetsearch.research.google.com/
[3]https://www.kaggle.com/franchenstein/austin-zoning-satellite-images

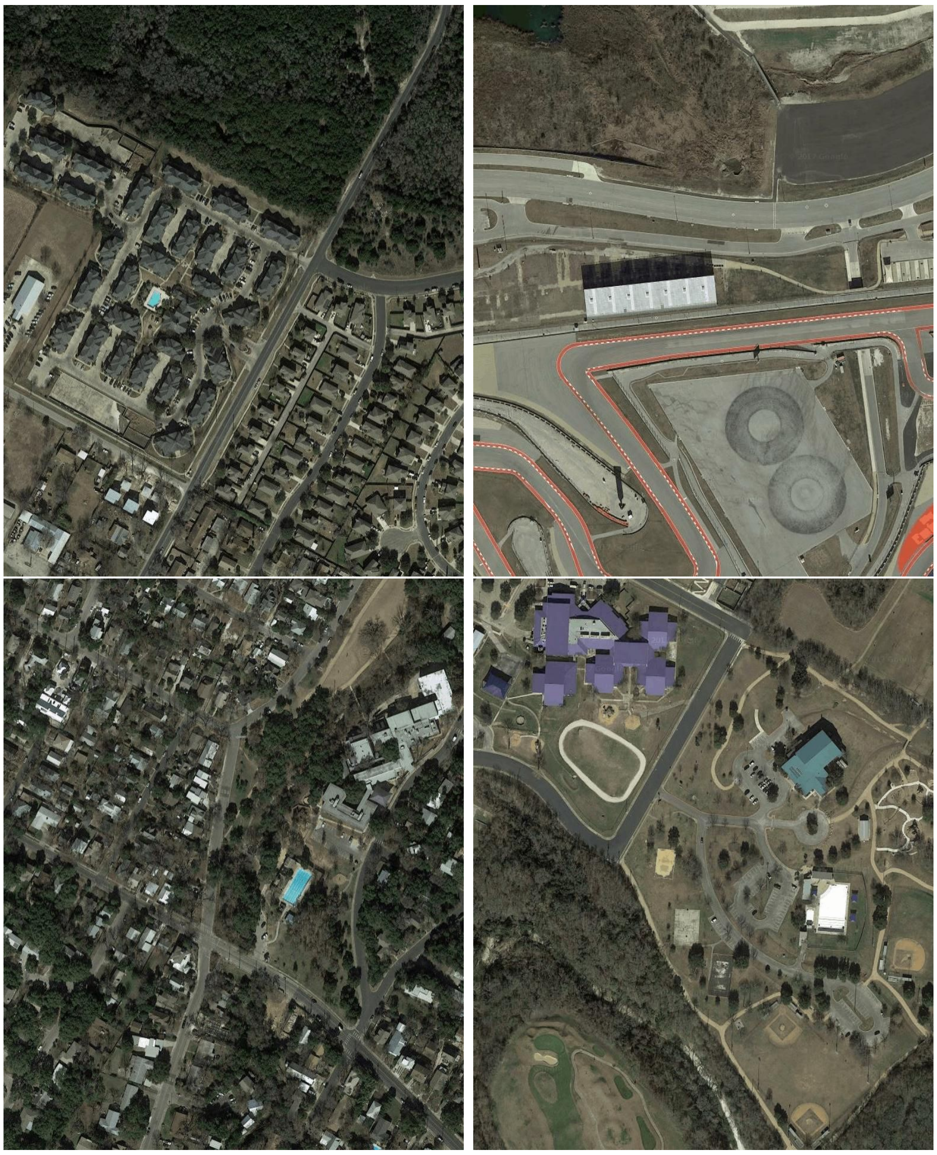


Figure 4.1: A few examples from the Austin dataset.

At first, I wanted to detect sports fields like football pitches or tennis courts. As I was going through my dataset, I figured out that there are not enough sports fields in the images, which meant not enough data (labels) for training a good detection model. Thus, I decided to detect not only sports fields (in particular tennis courts), but also swimming pools because that will provide me with more labeled data.

The images in my dataset were labeled by the supervisor (in this case, it was me). That means I created bounding boxes (rectangles) around swimming pools and tennis courts by myself via the online platform Makesense[4] and exported these labels into a single `.csv` file. After exporting the data, I split the dataset into *train* (training) and *val* (validation) set. The ratio between the two sets was approximately 90:10.

[4]https://www.makesense.ai/

label	x1	y1	width	height	filename	img_width	img_height
pool	159	42	20	25	austin_104.jpg	773	960
pool	303	296	16	17	austin_104.jpg	773	960
pool	211	336	32	26	austin_104.jpg	773	960
pool	165	140	29	32	austin_1040.jpg	773	960
court	383	541	75	84	austin_1042.jpg	773	960
court	335	59	45	59	austin_1043.jpg	773	960
pool	363	93	27	55	austin_1043.jpg	773	960
pool	143	638	29	43	austin_1046.jpg	773	960
court	439	82	103	91	austin_1054.jpg	773	960

Table 4.1: Example of CSV file after exporting annotations.

As it can be seen in Table 4.1, the first column named **label** is to store whether the labeled object is a pool or a court. Columns **x1**, **y1**, **width**, **height** determine the coordination of a labeled object in image. The last three columns represent the name and resolution of the image.

Augmentation

To enlarge the number of images in the training dataset, I used augmentation. This technique is used to gather more data for training a neural network. It is a process that generates new data from the existing ones. In my thesis, I used a library named `Albumentations` [4]. Their documentation makes this library not only powerful but also user-friendly. That was the main reason behind selecting it.

There is a variety of options for how to augment images. I chose four augmentation types and applied them to those images, which contained either a swimming pool, a tennis court, or both of them.

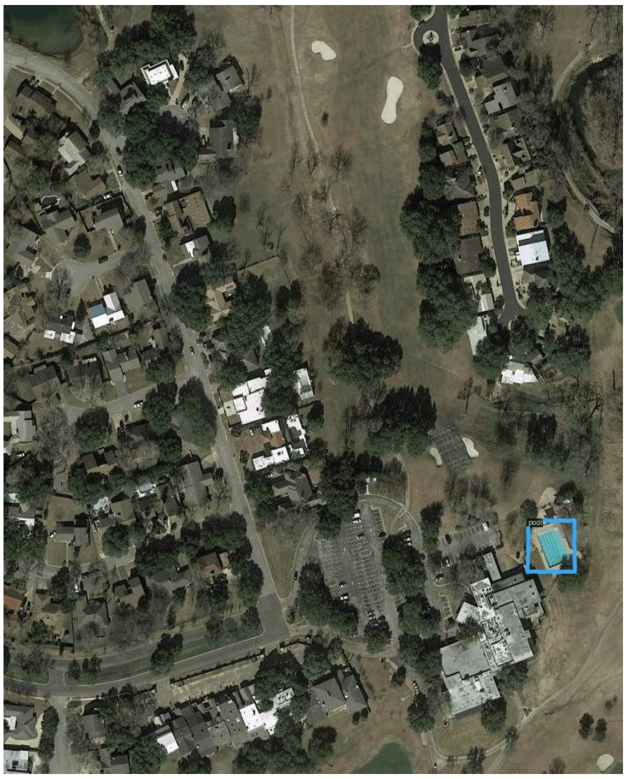
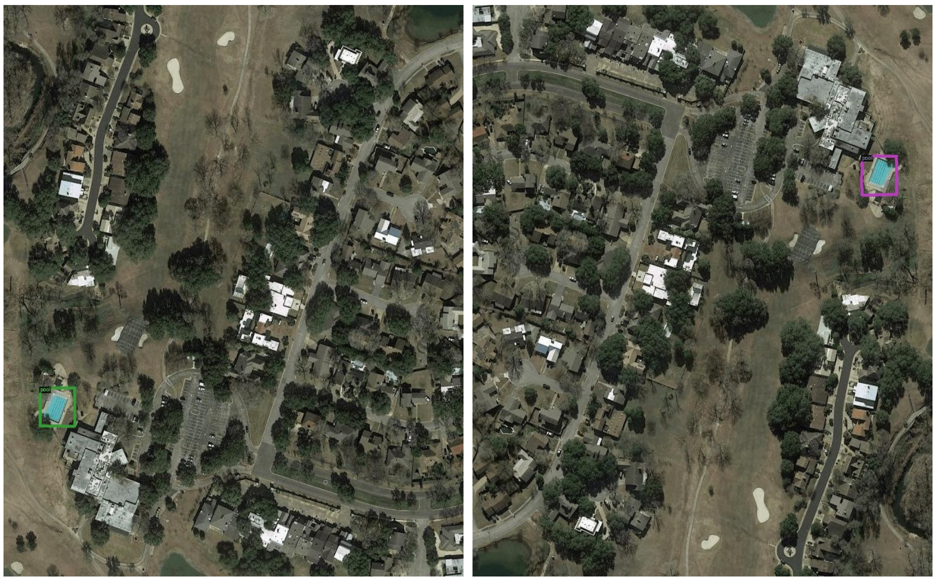


Figure 4.2: Example of the original image from the dataset before augmentation with a ground-truth bounding box of a swimming pool.

At first, I tried only two augmentations. A horizontal and vertical flip of the original photo. These transformations change the position of pools or courts in the image. Therefore, bounding box coordination changes had to be added into the `.csv` file with labels.



(a) Horizontal flip of the original image (b) Vertical flip of the original image

After some experiments with the trained model, I wanted to get better results, so I decided to further expand my dataset with other two types of augmentation.

(a) Horizontal and vertical flip

(b) Original image in gray scale

4.1.2 Inria dataset

In the winter semester, I used only the Austin dataset, which was described in section 4.1.1. To perform the detection of desired objects in other cities, the dataset had to be further expanded. I was deciding again between creating a new one or finding an appropriate one.

My first choice was to find an appropriate dataset instead of creating a custom one. The two criteria which I set for myself were:

- dataset must consist of 3 or more different cities,
- number of images in the dataset must be at least double than in the Austin dataset.

The one that met my criteria was the Inria Image labeling Dataset[5]. It addresses a core topic in remote sensing, which is automatic pixel-wise labeling of aerial images [24]. Images are split into two portions. One is desired for training and includes five cities, the other one is desired for testing and includes also five cities.

Train	Tiles	Total area	**Test**	Tiles	Total area
Austin, TX	36	81km^2	Bellingham, WA	36	81km^2
Chicago, IL	36	81km^2	San Francisco, CA	36	81km^2
Kitsap County, WA	36	81km^2	Bloomington, IN	36	81km^2
Vienna, Austria	36	81km^2	Innsbruck, Austria	36	81km^2
West Tyrol, Austria	36	81km^2	East Tyrol, Austria	36	81km^2
Total	180	405km^2	Total	180	405km^2

Table 4.2: This table shows a list of the cities in each of the sets, tiles (number of images), and their land coverage. The table was recreated and inspired by [24].

[5] https://project.inria.fr/aerialimagelabeling/

As it can be seen in table 4.2, every city is divided into 36 images. Each image has resolution of `5000x5000` pixels and all 36 images together cover `81km`^{2} of land in the desired city.

The purpose of the Inria project was to create a classifier trained on one dataset and use it on another dataset with completely different places on Earth leading to reaching the best accuracy possible. They work with two semantic classes: *building* and *not building*. Inria project and its dataset is further described in the paper by Maggiori and others [24].

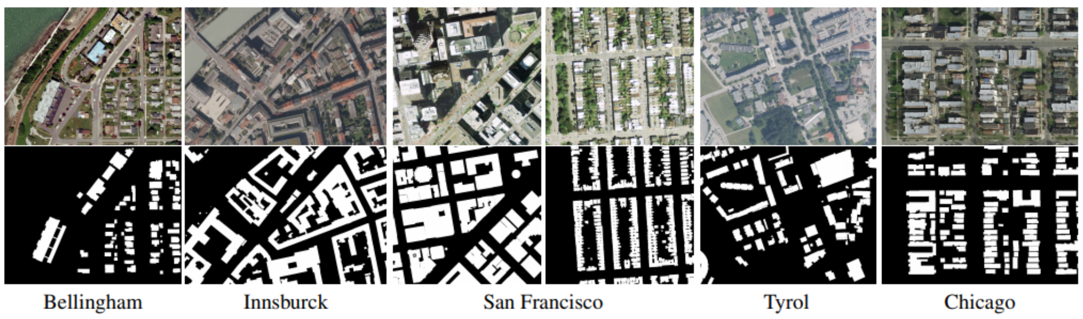


Figure 4.5: Images from Inria dataset and corresponding labeled images. It shows white - *building class* and black - *not building class*. Image was taken from [24].

Figure 4.5 shows that the Inria project used segmentation to decide whether there is a building or not in the image. Therefore, the Inria dataset also includes labels for this segmentation. However, I did not use these labels but I had to create my own because their labels did not fit into my thesis.

Dataset optimization

Due to the resolution of each image (`5000x5000` pixels), I had to make some changes to the original Inria dataset. I created a script, which helped me to fasten up the process of cropping. Each image was cropped into smaller images with the same resolution as my first Austin dataset – `773x960` pixels. The output of the script was 42 new images for each image (tile column in table 4.2).

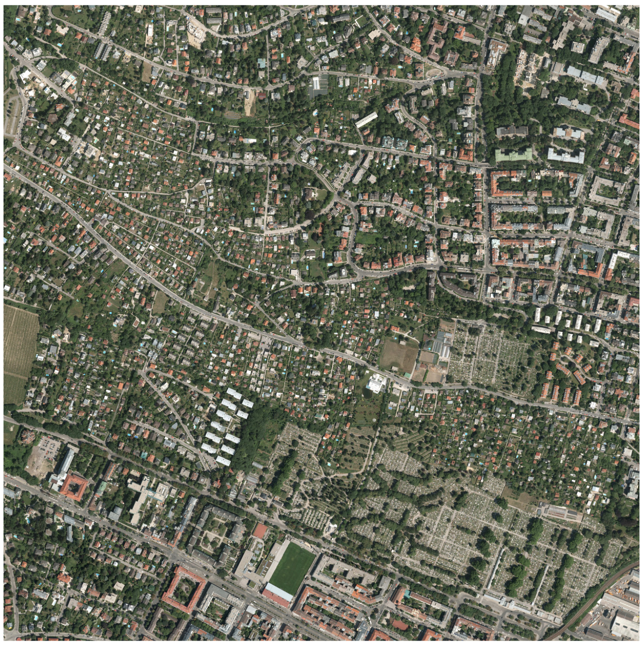


Figure 4.6: Original image from Inria dataset. Image is from Vienna and its original size is 5000x5000 pixels.

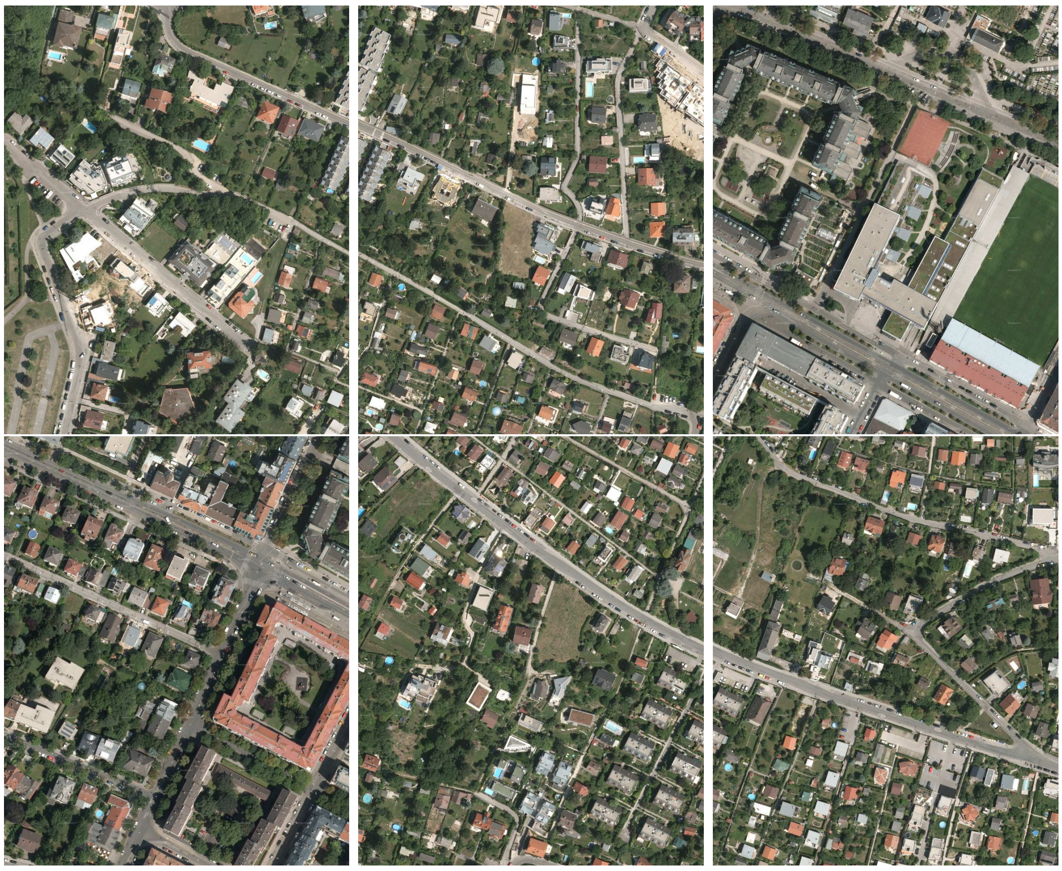


Figure 4.7: Few examples of cropped images from Figure 4.6.

The Inria dataset is divided into 2 sets, specifically one for training and one for testing. I merged *train* and *test* sets together into one set because I wanted data from as many cities as possible. Additionally, I created another script, which randomly selected 100 images from each city that occurred in the merged set. I did not choose images from Austin, because I already had enough data from this city, so I completely discarded Austin. It means I had 900 new images at disposal for my training set (which I used in the winter semester). I labeled swimming pools and tennis courts and applied augmentation in the same manner as in section 4.1.1.

The next step was similar to the previous one, but instead of adding the images into the training set, I had to add these new images to my validation set. I used the same script, but I randomly selected only 50 images from each city and labeled them afterwards. The augmentation was not applied, since this was the validation set.

All other images from the merged set, which I mentioned above, were used for final predictions. This set is named *predictions*.

4.1.3 Dataset summary

To summarize and better understand how I split my dataset, how many images are in each set, and what is the total number of images, I created Table 4.3.

Set	Number of images
train	5 879
val	898
predictions	13 760
Total	20 537

Table 4.3: Summary of my dataset.

To demonstrate the representation of individual city images in my dataset, I created a script that plots these results into a pie chart. The names in the legend are the names of my variables in Python script. To get the exact name of the city, look at Table 4.2.

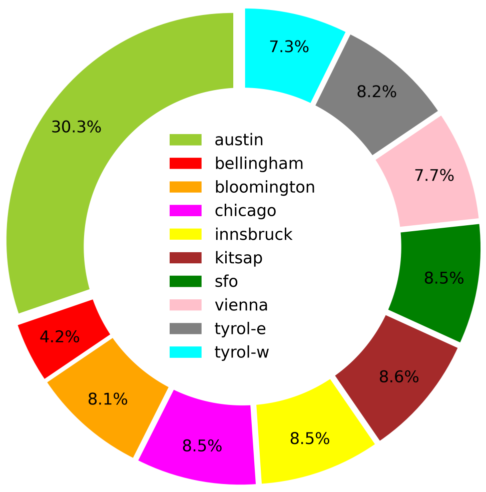


Figure 4.8: Representation of individual city images in training set.

There are two main reasons why the training set is not uniformly distributed. Firstly, my first dataset was solely the Austin dataset and I had already labeled many images from it. I used only some of them, not all of them, in my new training set. However, as it can be seen in Figure 4.8, it still creates the major part of this set. Secondly, I did the augmentation only on images, where either a swimming pool, tennis court, or both were presented as I mention later in 5.1. That means if a city had more base images with no objects in the training set, they were not augmented. The best example is Bellingham, which did not comprise of many pools and courts.

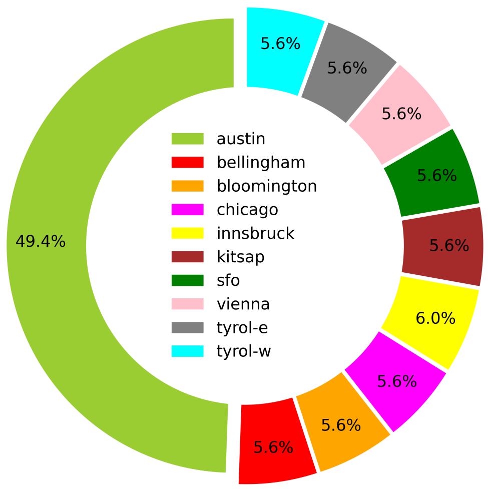


Figure 4.9: Representation of individual city images in validation set.

It is possible to see in Figure 4.9 that images are uniformly distributed except for Austin again. The reason remains the same as in the *train* set.

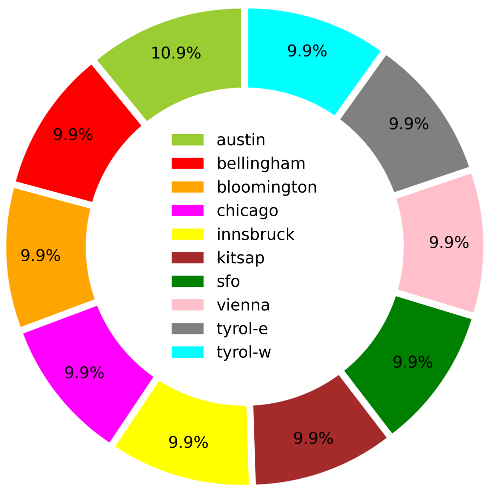


Figure 4.10: Representation of individual city images in prediction set.

Figure 4.10 shows that in *predictions* set, which was used for the final predictions, the images are distributed uniformly with slightly more images from Austin again.

4.2 Used technologies

In this section, I will introduce some of the significant technologies or packages which I used during the process of creating this bachelor thesis. I do not consider it necessary to mention every package which is a part of the native Python language.

4.2.1 Python

The main code was written in Python programming language of version 3.8[6]. Also, all of the scripts, which were created to simplify the work, such as cropping hundreds of images into smaller ones or sorting images into different sets, were implemented in Python. I chose this language primarily because it has great support for machine learning, computer vision, and neural networks. It offers an enormous amount of useful libraries to work with for neural network (Tensorflow[7], PyTorch[8], etc.) as well as has great understandable documentation. Last but not least, I like this language the most and have the most experience with it.

4.2.2 PyTorch

PyTorch is an open-source machine learning library, which core is based on Torch library. It is used for applications related to computer vision or language processing. It was developed primarily by Facebook's AI Research Lab (FAIR)[9] [17]. It provides many high-level features to Python. One of the main ones is definitely tensor computation (e.g. NumPy) with a noticeable acceleration of GPU [29]. At the moment of writing this thesis, PyTorch has

[6]https://www.python.org/
[7]https://www.tensorflow.org/
[8]https://pytorch.org/
[9]https://ai.facebook.com/

released a new stable version `1.8.0`. However, I use the PyTorch version `1.7.1` in my thesis.

4.2.3 OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source library for computer vision and machine learning purposes. This library embodies more than 2500 optimized algorithms for computer vision and machine learning. These algorithms can help to identify objects, detect people in videos or images, recognize faces, classify various actions in a video, and many more [26]. I used this library mostly for operations with images in my dataset (e.g. opening an image, saving into the file, visualizing, etc.).

4.2.4 Detectron2

Detectron2 is a next-generation software developed by Facebook AI Research. It implements state-of-art algorithms for object detection and is an upgraded version of previous `Detectron` [11]. This new version comes with significant changes [37]:

- first version was implemented in Caffe2 which was deprecated and merged with PyTorch. So Detectron2 is powered by PyTorch,
- provides more features such as panoptic segmentation, rotated bounding boxes, Cascade R-CNN [5], and others,
- can be used as a library and support other projects on top of it,
- training is much faster than in the previous version.

I chose this software because of its speed, results in object detection, documentation, and their great introduction to use with a tutorial on Google Colab[10].

4.2.5 Google Colab

Also known as Google Colaboratory is a free environment similar to Jupyter Notebook for Python[11], which completely runs on Google Cloud. Furthermore, this environment has many Python libraries already installed. This product was developed by Google Research to spread the availability of developing machine learning code. Since nowadays machine learning is demanding on computational power, Colab comes with quite a phenomenal solution with built-in GPUs and even TPUs which can be used for free, for example, for training a neural network. The free version of using these sources is limited and these sources are being dynamically assigned while using Colab. There is always a chance of upgrading this free version to a better one and get more computational power for training, but that is not for free anymore.

4.2.6 Albumentations

This library was already mentioned in section 4.1.1 in relation to augmentation. It is a Python library that provides fast and flexible image augmentations. Support of different

[10]https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5
[11]https://jupyter.org/

computer vision tasks like image classification, semantic and instance segmentation, object detection, and pose estimation is one of the reasons why I chose to work with this library. It is also open-source and free to use and has seamless integration with PyTorch, which I use in my thesis [4].

Chapter 5

Implementation

In this chapter, I will introduce the process of implementing the code, training not only my final model but also the older ones.

As it was mentioned in 4.2.4, I used the Detectron open-source tutorial code on Google Colab as a starting point. As it all runs in the cloud, I had to use some storage to save my dataset, `.csv` file with labels, and I stored the newly trained model after training to have the best version always ready and to see the progress between trained versions.

I chose to use Google Drive as a storage location, which has to be mounted at the beginning of each notebook to work with the necessary data. Since I wrote the code in Google Colab, I could not separate individual parts into modules, as I would normally do in Python. Thus, I divided these notebooks into 4 separate parts for a better view of the code.

- Data Analysis and Predictions - the code in this notebook served for calculations about the dataset and its graphical visualization, and also for final predictions
- Data augmentation - this part was used in order to expand my dataset with help of Albumentations library
- Training - this notebook serves for the training of my model
- Inference - this notebook was devoted to evaluating evaluating the model accuracy, precision, and other statistics

Google Colab runs in a "runtime", which can be imagined as a process. After this process ends (for example because of inactivity for a certain period of time or the process is just simply killed by exiting), everything that was stored in this "runtime" is deleted and not available anymore. It means that as it was already mentioned, Google Drive with stored data has to be mounted every time, every installation of a package or library, every import, simply everything has to be loaded repeatedly. Therefore, all of my notebooks have the same first couple of cells.

One of the cells is for registering my dataset in a supported format for Detectron2. In order to make Detectron2 understand how to work with my dataset, I had to create a function, which would return a list of annotations in a certain custom format and register it to Detectron2 catalog alongside metadata. More about using the custom dataset in Detectron2 can be found in their documentation [7].

5.1 Data augmentation

This notebook was used to expand the training dataset by using augmentation. With the help of `Albumentations` library, I wrote the code, which makes it possible to create new augmented images. I used their documentations and followed some example codes in it[1].

I encountered a problem, because my bounding boxes, which were in my `labels.csv` file with labels, did not automatically move and transform into an augmented image. So in addition to augmentation of the image, I wrote also a code that appends new coordination of bounding boxes in the new augmented image into `labels.csv`.

I applied augmentation only to the images, which contained one or more labels, either a swimming pool or a tennis court, because these images were significant for the training of my model. Thus, the final number of my training dataset with all augmented images is **5 874**.

5.2 Training

In this notebook, I trained my neural network model. I trained over a dozen models during the work on my thesis. Every time I trained a new model, I ran an inference and evaluated its performance. These results helped me to choose which model is better and what should I try to change next, to make the performance even better than it was at that moment.

It is normal nowadays that neural networks, which are fed with billions of data, are trained for days or even for weeks. Although I did my thesis in Google Colab, the training was getting longer and longer over the time. The first couple of models at the beginning of the winter semester were trained after two hours maximum. However, with the growth of my training set, the required time for training has also grown. I trained my final models for ten, eleven, and even twelve hours.

Final model

For final predictions described in chapter 6 I used `model_final.pth`. This model was trained for about 11 hours and its settings are described in `training.ipynb` notebook. The crucial hyperparameters, which affected the performance of each model the most were:

```
cfg.SOLVER.IMS_PER_BATCH = 12
cfg.SOLVER.MAX_ITER = 8000
```

The number of `IMS_PER_BATCH` (Images per Batch) was set to 12 because these batches use GPU which is not unlimited in Google Colab. When I tried to increase the batch size, I could not train because the GPU resources were not enough. Lowering of batch size led to worse results of precision by the trained model.

The number of `MAX_ITER` (Maximum Iterations) was set to 8000. I experimented with this number a lot. It determines maximum iterations for training. To avoid overfitting my model, I set this number to 8000. When I set this number to 10 000, the model was overfitted and its evaluation on *val* set was worse than the previous model with 8000 iterations. Furthermore, the loss training curve stopped decreasing. After training, I evaluated my model and its results can be seen in 5.3.3.

[1] https://albumentations.ai/docs/getting_started/bounding_boxes_augmentation/

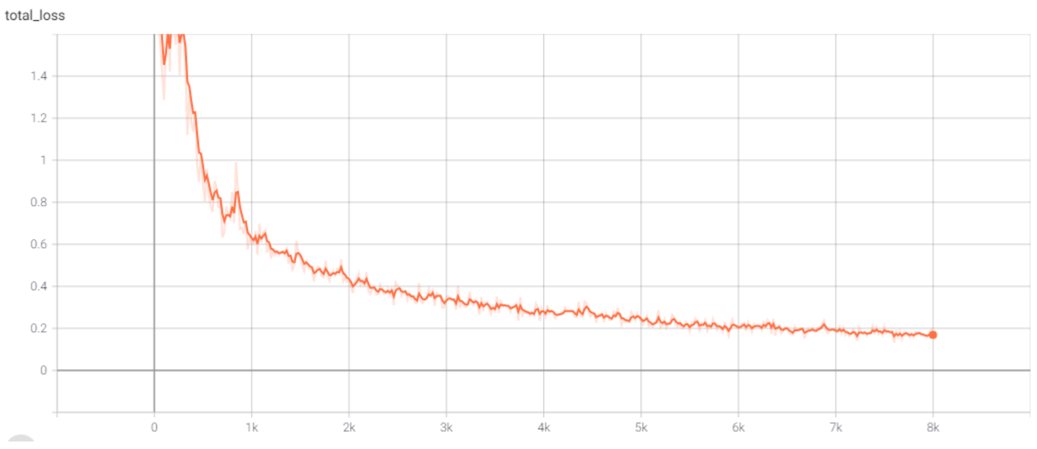


Figure 5.1: The loss curve of a final model. This curve was generated during the training of a model. X-axis is for number of iterations and y-axis represents loss value.

The loss calculation in Detectron2 is done during the training. There are two loss functions [15].

- Localization loss
  - uses L1 loss function[2] which is used to minimize the error of all the absolute differences between the predicted and true values
- Classification loss
  - uses softmax cross entropy loss[3]

5.3 Inference

In the object detection task, it is appropriate to measure how our algorithm is performing. Some kind of technique, which will evaluate the accuracy of our trained model. This evaluation method is not as simple as for example, in image classification. The problem is that, if we compare the predicted bounding box of our model and the ground-truth bounding box and they are not exactly the same, our prediction is incorrect. That is not necessarily true, because our predicted bounding box can be a little off the ground-truth, which means our model correctly detected the desired object, but not exactly the same position as the ground-truth. That is why the *intersection over union* is used as a metric for object detection.

Intersection over union

Intersection over Union (IoU) is the most popular evaluation metric used in object detection tasks, which measures the accuracy of the algorithm. The actual equation is pretty simple. It is a relationship between ground-truth and prediction made by a trained model. In Figure 5.2 the red rectangle represents ground-truth, which was made by a person. The

[2] https://afteracademy.com/blog/what-are-l1-and-l2-loss-functions
[3] https://peterroelants.github.io/posts/cross-entropy-softmax/

green rectangle represents the prediction of the trained model. As it can be seen, the green prediction does not perfectly fit the red ground-truth.

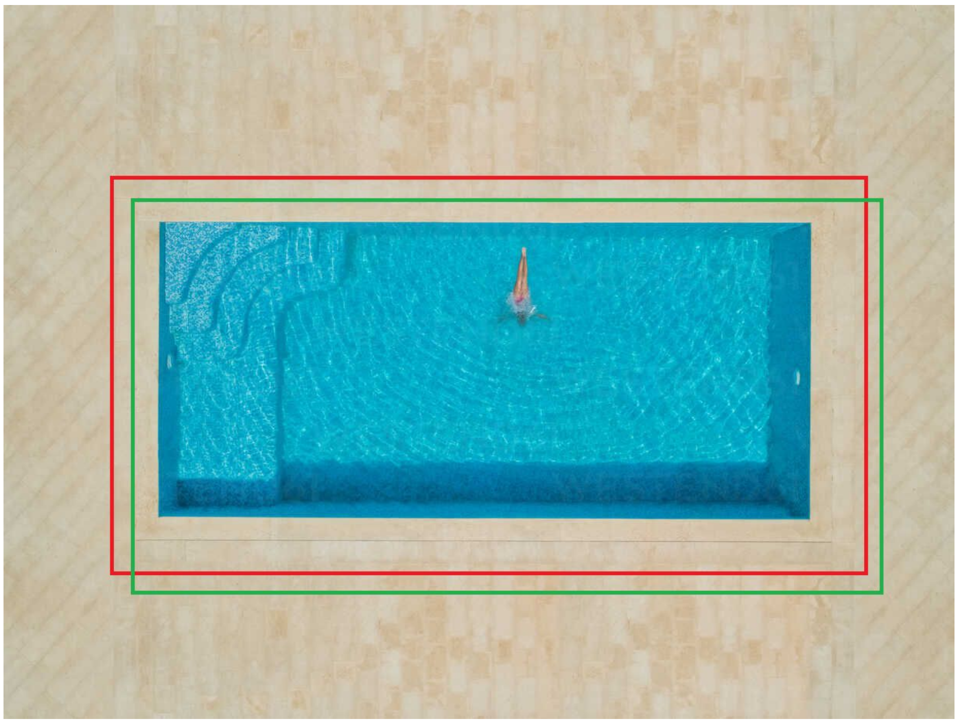


Figure 5.2: Ground-truth marked with red rectangle and prediction with the green one.

From the mathematical perspective, IoU can be described by this simple equation [31]:

$$IoU = \frac{A \cap B}{A \cup B}, \tag{5.1}$$

in the numerator, an overlap between ground-truth and prediction is computed and in the denominator, the union of both is computed. The result is a ratio which is called *intersection over union.*

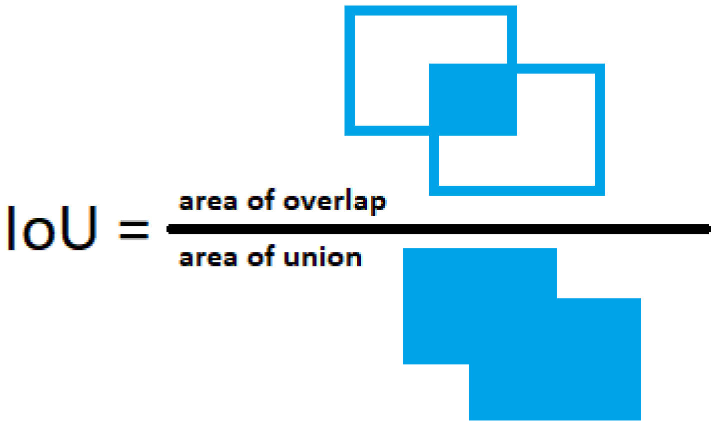


Figure 5.3: Visualization of the IoU Equation 5.1

In general, if the IoU has a score of 0.5 and higher, it is considered to be a good prediction.

5.3.1 Precision and recall

To calculate precision and recall, it is necessary to identify three (four) types of predictions. I set my IoU testing threshold to **0.7**.

- True Positive (TP) – if the IoU is greater or equal 0.7
- False Positive (FP) – IoU less than 0.7, wrong prediction of class, no overlap
- False Negative (FN) – when a model should make a prediction, but misses the object and does not make any prediction
- True Negative (TN) – since every part of an image, where is not any prediction is TN, it is quite useless in this context

Precision: *From all swimming pools (tennis courts), that we detected, what is the fraction of actual swimming pools (tennis courts)* ?

$$precision = \frac{\text{True Positives}}{\text{True positives} + \text{False positives}}$$

The precision in this context measures the ratio between true object detections and a total number of detections. That means, if the precision is close to 1, there is a high probability that whatever the trained model detects as a true positive is a correct detection.

Recall: *From all actual swimming pools (tennis courts), what fraction did we detected correctly* ?

$$precision = \frac{\text{True Positives}}{\text{True positives} + \text{False negatives}}$$

The recall is very similar to precision, but they differ in the denominator. The recall is the ratio of true object detections to the total number of objects that are in the dataset. That means, if the recall is close to 1, there is a high probability that all objects in the dataset will be positively detected by the trained model.

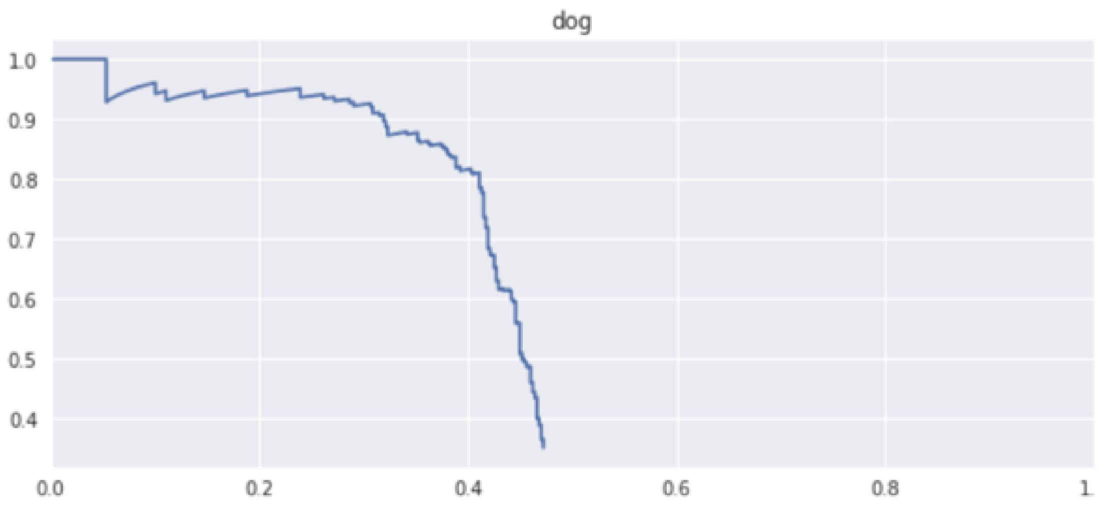


Figure 5.4: The precision-recall curve for a class dog. Recall is on the x-axis and ranges from 0 (no ground-truth objects found) to 1 (every object found). The precision is represented on the y-axis. The reason why the area under the curve is the average of the precision for this class, is that the precision is given as a function of recall. The image was taken from [14].

In the Figure 5.4 can be seen a **precision-recall curve**. This curve is created by computing the precision and recall scores at different threshold levels. In the case of object detection, the threshold is also called as a confidence level of the predicted bounding box. To achieve this curve, it needs some calculations. At first, the precision and recall for the first prediction (with the highest threshold) are computed. Next step is to compute the precision and recall for the first and second predictions (lower threshold), then the top three predictions (even lower threshold). One is supposed to continue until every prediction is covered (the lowest threshold). Each of these values represents an operating point on the precision-recall curve.

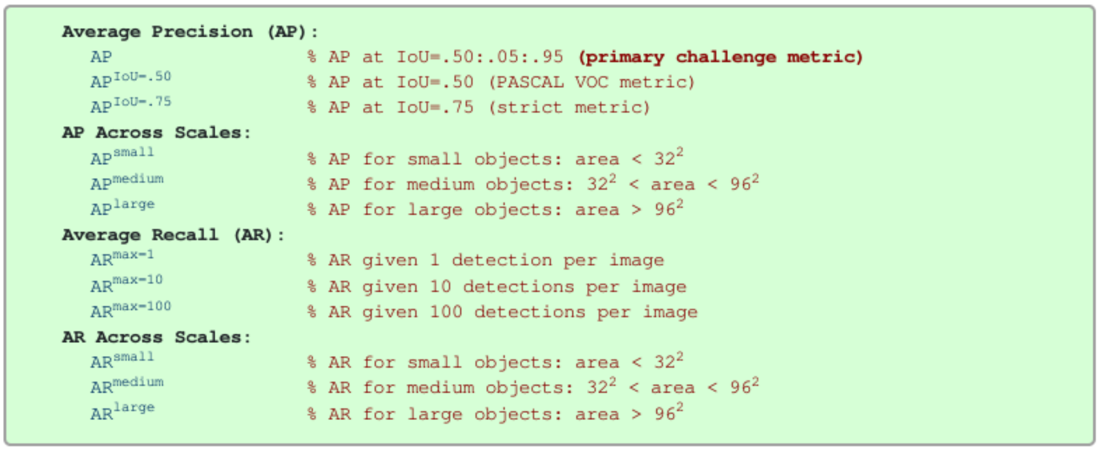
That means if the chosen threshold is high, the model will find fewer predictions and therefore fewer FP (model makes fewer mistakes), but contralily the model will detect more FN (more objects missed). Otherwise, the lower the threshold is, the more predictions will be found. However, usually, these predictions have lower quality [14].

5.3.2 Mean average precision

Mean average precision (mAP) can have different definitions. It is most commonly used in the informational retrieval and object detection sphere. The object detection mAP was first introduced in the Pascal VOC challenge. Aside from Pascal VOC, it is popular also in ImageNet and COCO object detection challenges. This single number metric uses, apart from precision and recall, also IoU and confidence threshold. To calculate the mAP, it is necessary to calculate AP per class.

For evaluation, I used COCO metrics and `COCOEvaluator`. That means AP is the area under the precision-recall curve. Each detection have a confidence level. These detections are then ordered by confidence and matched to ground-truth. Each recall value has a corresponding precision.

In COCO evaluation, the IoU threshold ranges from 0.5 to 0.95 with a step size of 0.05, that is represented as `AP@[.5:.05:.95]`. Fixed IoUs thresholds like 0.5 or 0.75 are marked as AP50 respectively AP75. To get AP and Average Recall (AR), it is necessary to average multiple IoU values, unless otherwise specified. COCO uses 10 IoU thresholds (.50:.05:.95), which is a rather new trend from traditional computation where AP is from a single IoU threshold of 0.5. This new approach rewards detectors with better localization [19].



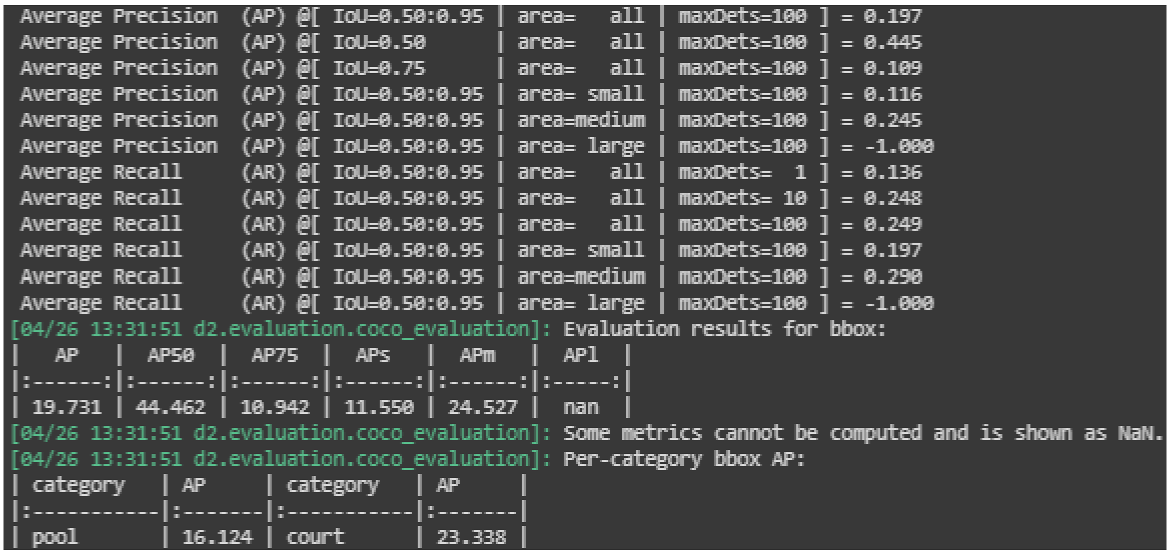
Average Precision (AP):	
AP	% AP at IoU=.50:.05:.95 **(primary challenge metric)**
$AP^{IoU=.50}$	% AP at IoU=.50 (PASCAL VOC metric)
$AP^{IoU=.75}$	% AP at IoU=.75 (strict metric)
AP Across Scales:	
AP^{small}	% AP for small objects: area < 32^2
AP^{medium}	% AP for medium objects: 32^2 < area < 96^2
AP^{large}	% AP for large objects: area > 96^2
Average Recall (AR):	
$AR^{max=1}$	% AR given 1 detection per image
$AR^{max=10}$	% AR given 10 detections per image
$AR^{max=100}$	% AR given 100 detections per image
AR Across Scales:	
AR^{small}	% AR for small objects: area < 32^2
AR^{medium}	% AR for medium objects: 32^2 < area < 96^2
AR^{large}	% AR for large objects: area > 96^2

Figure 5.5: Table of individual COCO metrics. The image was taken from [6].

5.3.3 Inference comparison between the trained models

As it was mentioned in the Section 5.3, I used `COCOEvaluator` to evaluate the model. The *val* set was used for evaluation. Output of this evaluation is in a format like in Figure 5.5. The metric, which I focused on was `AP50`. In the winter semester, when I used only one city for both training and validation, my AP50 was approximately 70 percent with a testing threshold (confidence level) of 0.7. After I made changes to the dataset in the summer semester, this metric dropped down to ~45 percent with the same threshold. Therefore, my goal was to get back to or close to the previous 70 percent at a threshold 0.7.

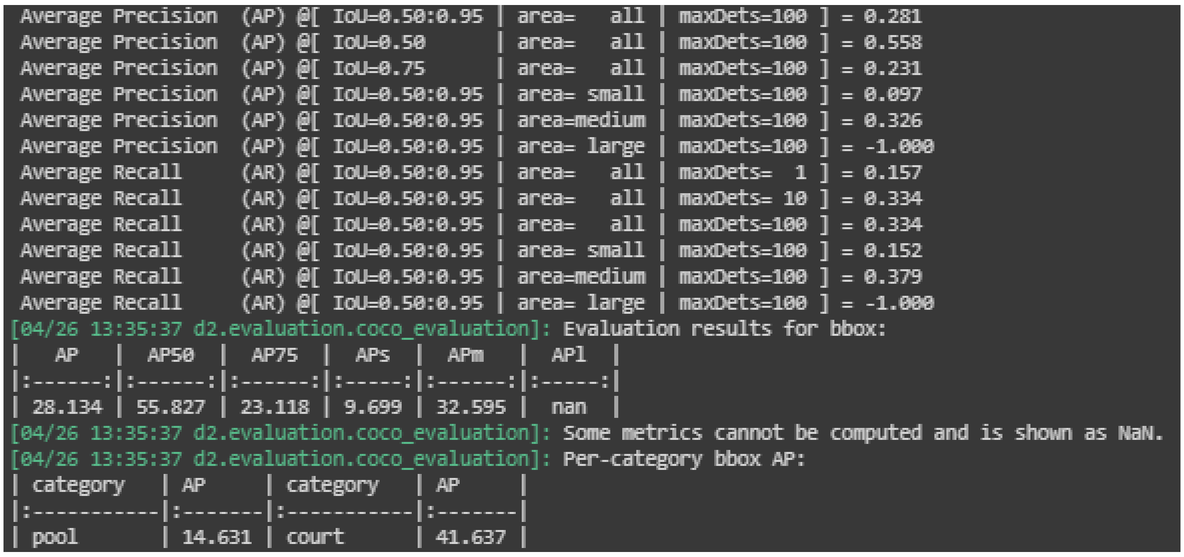
The winter semester model was evaluated on new *val* set with additional images from each city, as I described this process in Section 4.1.2. Results of the evaluation can be seen in Figure 5.6. I knew that the decrease of AP50 is caused by the new additions to the validation set and also because the model had not seen any of the new images in training. Moreover, a lot of labeled data from the Inria dataset had a bit smaller bounding boxes than it was in the Austin dataset. That means this model was not ready to detect such tight bounding boxes correctly. This model had `cfg.SOLVER.IMS_PER_BATCH` set to 8 and `cfg.SOLVER.MAX_ITER` set to 3000.



```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.197
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.445
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.109
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.116
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.245
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.136
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.248
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.249
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.197
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.290
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
[04/26 13:31:51 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl  |
|:------:|:------:|:------:|:------:|:------:|:-----:|
| 19.731 | 44.462 | 10.942 | 11.550 | 24.527 |  nan  |
[04/26 13:31:51 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[04/26 13:31:51 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|
| pool       | 16.124 | court      | 23.338 |
```

Figure 5.6: Output of `COCOEvaluator` for winter semester model evaluated on new (actual) *val* set.

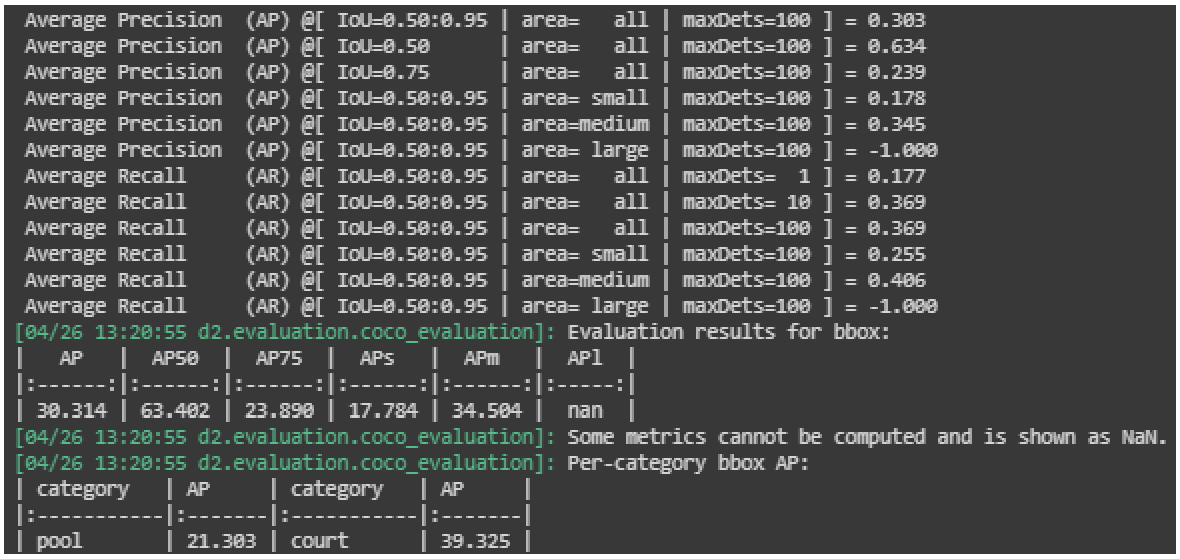
To improve the precision, I added the images to my *train* set as it is again described in Section 4.1.2. After these additions, I trained a model with the same hyperparameters as when I trained my last winter model. The results were awful, as the AP50 was less than my previous model had. It reached the verge of ~20 percent. I changed images per batch to 12 and iterations to 5000 to get better results that can be seen in Figure 5.7. AP50 had reached the verge of almost 56 percent and it started to look promising.



```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.281
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.558
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.231
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.097
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.326
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.157
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.334
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.334
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.152
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.379
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
[04/26 13:35:37 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs  |  APm   |  APl  |
|:------:|:------:|:------:|:-----:|:------:|:-----:|
| 28.134 | 55.827 | 23.118 | 9.699 | 32.595 |  nan  |
[04/26 13:35:37 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[04/26 13:35:37 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|
| pool       | 14.631 | court      | 41.637 |
```

Figure 5.7: Output of `COCOEvaluator` for model with changed hyperparameters before augmentation.

The next step to improve my model and its precision was augmentation. I applied this process as it is described in Section 5.1 and Section 4.1.1. The hyperparameters remained the same as in the previous model at the first try. After the training and evaluation, the results were quite similar to the previous model and I got AP50 to ~61 percent. Therefore, I changed the iteration hyperparameter to double the size – 10000. As I mentioned in Section 5.2, this caused overfitting of the neural network. The results were worse than before and AP50 decreased from the previous 61 percent to ~58 percent. Then, I tried to train the next model for 8000 iterations. The results of its evaluation can be seen in Figure 5.8. Even when I tried to change the hyperparameters afterwards, these were the best results that I have achieved with a testing threshold of 0.7.



```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.303
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.634
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.239
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.178
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.345
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.177
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.369
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.369
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.255
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.406
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
[04/26 13:20:55 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl  |
|:------:|:------:|:------:|:------:|:------:|:-----:|
| 30.314 | 63.402 | 23.890 | 17.784 | 34.504 |  nan  |
[04/26 13:20:55 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[04/26 13:20:55 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|
| pool       | 21.303 | court      | 39.325 |
```

Figure 5.8: Output of `COCOEvaluator` for final model after augmentation and change of hyperparameters.

My final model, which was used for the final predictions in Chapter 6 has reached AP50 of **63.402**, which is close to my goal of 70 percent with the testing threshold set to 0.7.

5.4 Data Analysis and Predictions

This notebook can be divided into two parts. In the first part, I analyze my dataset and make graphs for its better visualization. Figures 4.8, 4.9 and 4.10 are the output of the code in this notebook. Moreover, the necessary computation for Table 4.3 can be found here.

The second part is a code where the final predictions are done. These results were used in Chapter 6. Additionally, there is also a code, where the statistics from labeled data are computed.

Chapter 6

Final predictions and results

In this chapter, I will summarize the results of the final model predictions. For predictions I used my *predictions* set which contains unlabeled images. I wrote a code which calculates the occurrence of every detected instance within an image. The output of this code is a dictionary in a form:

```
name_of_the_city : [number_of_detected_pools, number_of_detected_courts].
```

The functionality of the code in `Data Analysis and Predictions.ipynb` is simple. After loading all the images from *prediction* set into a variable `dataset_dicts`, there is a big `for` loop in which images are opened one by one. If the image has at least one detected instance and its prediction class is labeled by 0, it is a **pool**, otherwise it is a **court**. The results of these predictions can be seen in Table 6.1.

City	Number of pools	Number of courts
Austin	287	77
Bellingham	55	30
Bloomington	48	68
Chicago	12	124
Innsbruck	183	33
Kitsap county	36	11
San Francisco	25	114
Tyrol - east	104	10
Tyrol - west	61	30
Vienna	410	73

Table 6.1: This three column table represents results of the final predictions which were made on *predictions* set. In the first column (from the left) the cities are listed, in which the predictions were made. The second column displays the number of detected swimming pools by my final model. The last column demonstrates the number of detected tennis courts.

To verify the predictions, it is also possible to visualize the detected instances in the image.

```
...
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
...
```

Listing 6.1: These two lines of code serve to visualize desired image with detected instances. In `Dataset Analysis and Predictions.ipynb`, this part of code is not executed and it is only a part of block comment. However, it can be executed if necessary.

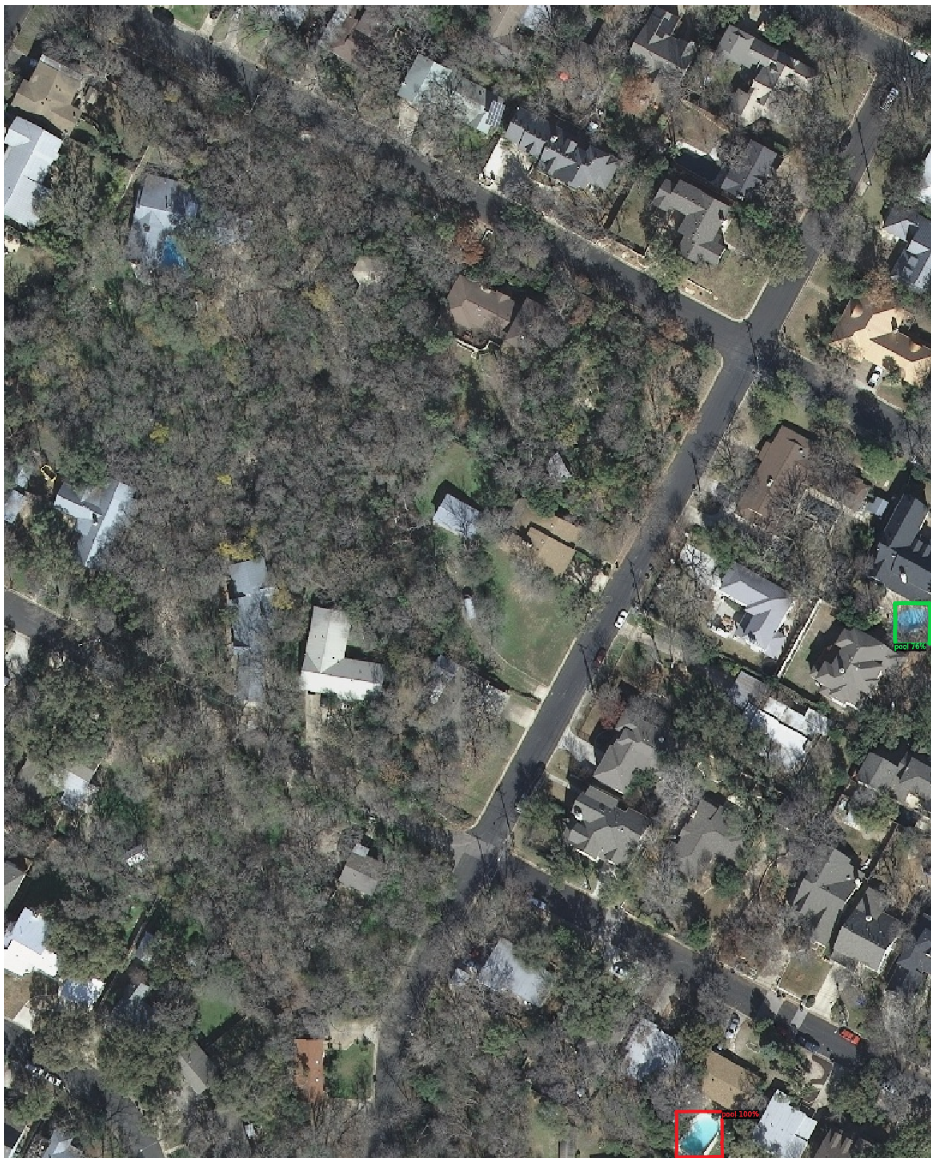


Figure 6.1: An example image with detected instances – in this case swimming pools – in Austin.

The Figure 6.1 shows an image, where two swimming pools were detected. The swimming pool with a green bounding box has a confidence level of 76 % and the red bounding box swimming pool has a confidence level of 100 %. But as the Figure 6.2 points out, there is one missed swimming pool. I made a blue circle with an arrow by hand in this image for better recognition. This pool was covered by a tree and not visible well even by a human, thus my model did not detect it.

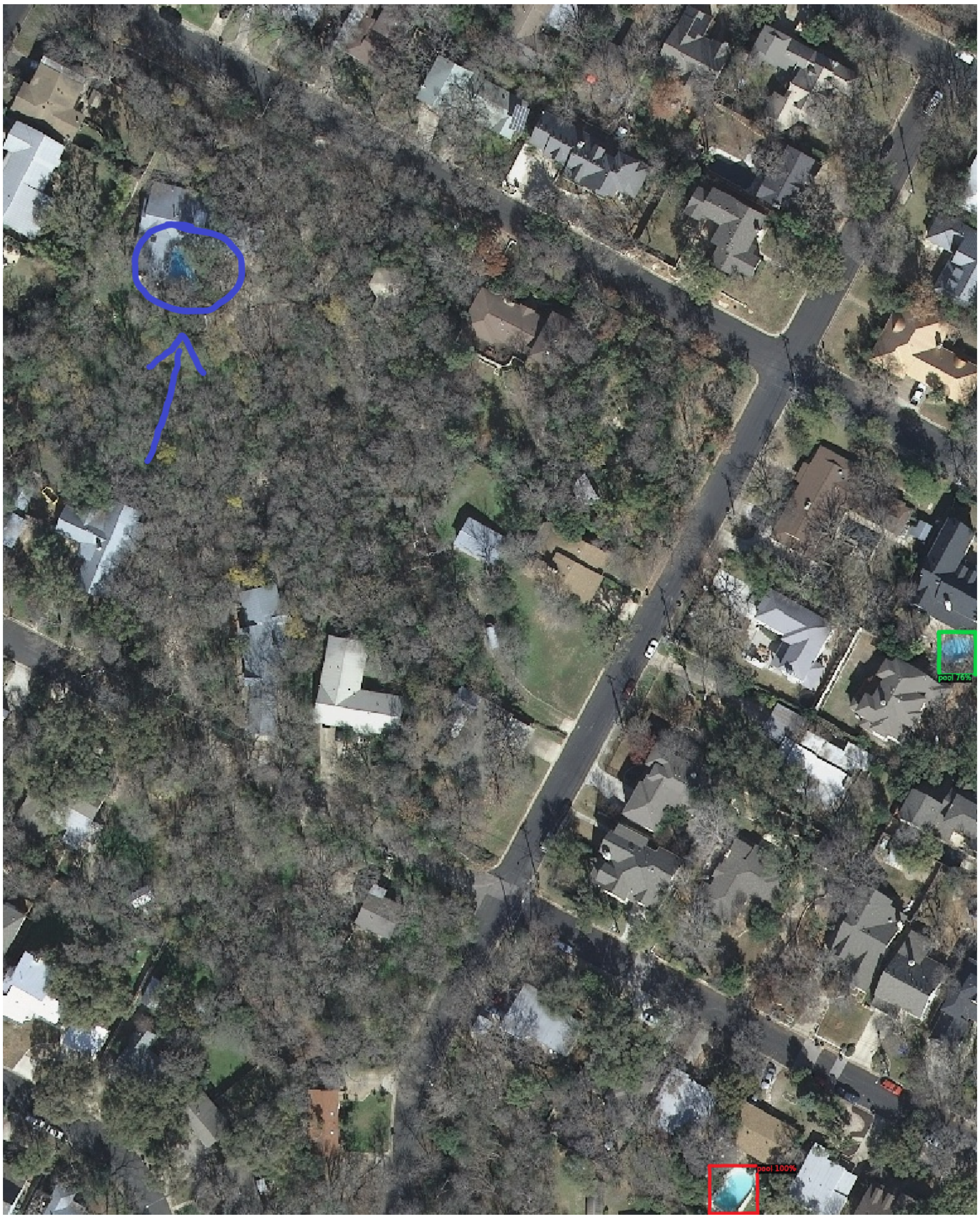


Figure 6.2: The blue circle in the upper left corner of the image indicates the missed detection. This pool is covered by trees and is really difficult to see it even by a human.

The Figure 6.3 serves to demonstrate the capabilities of the model when detecting tennis courts. These courts were all detected with a confidence level of 96 % and higher.

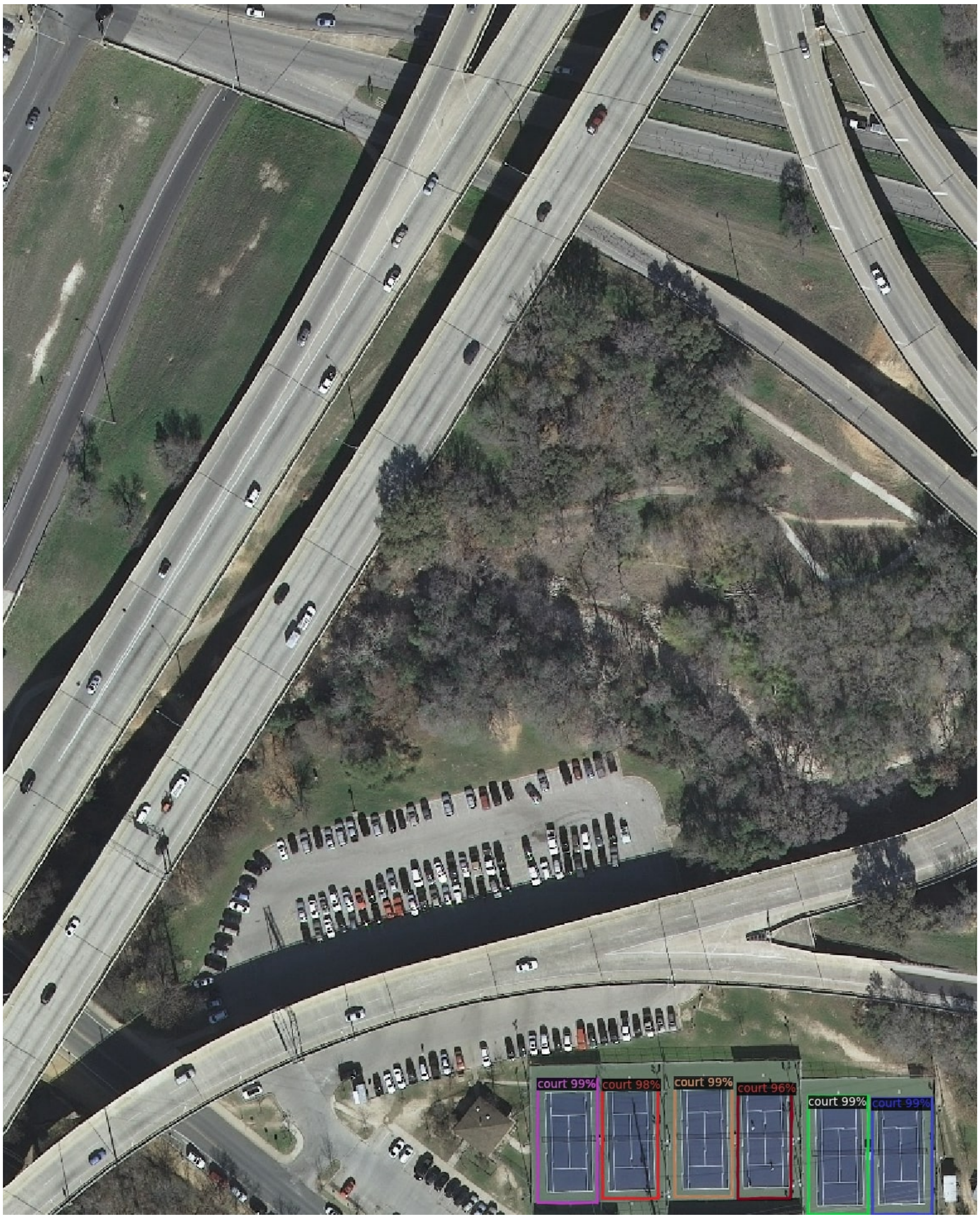


Figure 6.3: Demonstration of a tennis court detection with the trained model.

After I got these results presented in Table 6.1, it was necessary to count also objects in labeled images in *val* set and *train* set. This was a little bit tricky, because *train* set is comprised not only of original images, but mainly of augmented ones. It was essential to filter these images in the code, but the idea remained the same as when counting instances in *predictions* set. The results from both *val* and *train* set are presented in Table 6.2.

City	Number of pools	Number of courts
Austin	281	90
Bellingham	9	8
Bloomington	8	16
Chicago	2	22
Innsbruck	49	0
Kitsap county	5	0
San Francisco	3	27
Tyrol - east	26	4
Tyrol - west	8	8
Vienna	107	4

Table 6.2: This table is generic to Table 6.1 and represents number of pools and tennis courts in each city from labeled data.

It is necessary to combine Table 6.1 and Table 6.2 together to make a final estimation. Since I know how big the land coverage is in the Inria dataset, I can make an estimation count of swimming pools and tennis courts per square kilometer in these cities. Unfortunately, I was not aware the land coverage in Austin dataset, thus I could not make the estimation here.

City	Number of pools	Number of courts	Pools per km^2	Courts per km^2
Austin	568	167	-	-
Bellingham	64	38	0.79	0.47
Bloomington	56	84	0.69	1.04
Chicago	14	146	0.17	1.80
Innsbruck	232	33	2.86	0.41
Kitsap county	41	11	0.51	0.14
San Francisco	28	141	0.35	1.74
Tyrol - east	130	14	1.60	0.17
Tyrol - west	69	38	0.85	0.47
Vienna	517	77	6.38	0.95

Table 6.3: This table represents the final number of pools and tennis courts in individual cities. Values in the second and the third column (from left) are a combination of predictions and ground-truths from labeled data. In the last two columns, the number of pools/courts per square kilometer in each desired city is listed.

The results represented in Table 6.3 show the estimated count of swimming pools and tennis courts per square kilometer based on the predicted labels and ground-truth labels combined. As it can be seen, my model predicted the highest number of **6.38** swimming pools per square kilometer in Vienna, followed by **2.86** in Innsbruck and **1.60** in the east part of Tyrol. Other cities did not reach the verge of at least 1 pool per square kilometer. It would probably be Austin with the highest incidence, but I could not provide the same calculations as in the other 9 cities, since I did not have the exact land coverage of my dataset from Austin.

On the other hand, the highest incidence of tennis courts was in Chicago with a result of **1.80** tennis courts per square kilometer. San Francisco finished in a close tie with **1.74** pools per square kilometer. Bloomington had also reached a verge of 1 tennis court per square kilometer with a result of **1.04**. Again, I can hypothetically say that the incidence of tennis courts could be the highest in Austin, based on the numbers in table 6.3.

Chapter 7

Conclusion

I studied the most recent techniques and algorithms for computer vision in this thesis. My special focus was to deeply study object detection. I learnt a lot about machine learning and neural networks, especially convolutional neural networks, which I always wanted to try working with. In my solution, I got acquainted with Detectron2 library, which is a powerful tool not only for object detection, but also for other computer vision techniques like instance and semantic segmentation or image classification. I trained a model that is capable of detecting swimming pools and tennis courts with the AP50 of 63.402 %. Moreover, I had to study evaluating neural network models, their individual metrics, and how they are computed. I also learnt new things about creating datasets for neural networks and how to make an enlargement of them by using augmentation techniques.

During the work on my thesis, I have done multiple experiments with neural networks, specifically RetinaNet model. I have achieved an iterative improvement of my trained model thanks to the ideas which I turned to code and they worked. I also used my knowledge gained in other school courses, especially when creating multiple scripts or graphs for my textual part and improved my skills in Python language.

The goal of this thesis was to correctly detect swimming pools and tennis courts in different cities. I am satisfied with the achieved results and the detection capability of my solution. I got a little limited by GPU resources in Google Colab, and therefore, I could not realise some experiments that I wanted to. However, there is room for improvement especially in precision. I assume that if my *train* set contained only images, where the instance of an object is presented, the precision would grow. Another thing that comes to my mind, is that the model can be more precise for one class detection. From the beginning of my work, I worked with two classes – a pool and a court. A lot of successful object detectors work only with one class. Therefore, there is a possibility that detection for one class only could be more precise. The last improvement, which could possibly help to build more precise model, is to make *train* set more uniform than it is right now. It means that the ratio of images in *train* set from individual cities will be equal. I verified that my model is more precise when the validation set contains only Austin images than when the validation set contains all cities.

In the future, I would like to improve this detection model and transform it to a tool which detects the object in a satellite image correctly almost every time, so it could serve as an automatic tool for land checking. It means that the necessity of a personal visit would be no longer needed.

Bibliography

[1] *How To Label Data For Semantic Segmentation Deep Learning Models?* [online]. Anolytics, 2019 [cit. 2021-04-09]. Available at: https://www.anolytics.ai/blog/how-to-label-data-for-semantic-segmentation-deep-learning-models/.

[2] Bouška, F. *Localization and Counting of Humans based on Satellite and Aerial Imagery.* 2018. Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering.

[3] Burde, V. *Deep neural network for city mapping using Google Street View data.* 2020. Master's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Cybernetics.

[4] Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M. et al. Albumentations: Fast and Flexible Image Augmentations. *Information.* 2020, vol. 11, no. 2. DOI: 10.3390/info11020125. ISSN 2078-2489. Available at: https://www.mdpi.com/2078-2489/11/2/125.

[5] Cai, Z. and Vasconcelos, N. *Cascade R-CNN: Delving into High Quality Object Detection.* 2017.

[6] *Detection Evaluation* [online]. COCO, 2020 [cit. 2021-04-21]. Available at: https://cocodataset.org/#detection-eval.

[7] *Use Custom Datasets* [online]. Detectron2, 2019-2020 [cit. 2021-04-11]. Available at: https://detectron2.readthedocs.io/en/latest/tutorials/datasets.html.

[8] Dwivedi, P. *Semantic Segmentation — Popular Architectures* [online]. 2019 [cit. 2021-04-08]. Available at: https://towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0.

[9] Girshick, R. *Fast R-CNN.* 2015.

[10] Girshick, R., Donahue, J., Darrell, T. and Malik, J. *Rich feature hierarchies for accurate object detection and semantic segmentation.* 2014.

[11] Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P. and He, K. *Detectron* [online]. 2018. Available at: https://github.com/facebookresearch/detectron.

[12] Goodfellow, I. J., Bengio, Y. and Courville, A. *Deep Learning.* Cambridge, MA, USA: MIT Press, 2016. ISBN 9780262035613. http://www.deeplearningbook.org.

[13] HE, K., GKIOXARI, G., DOLLÁR, P. and GIRSHICK, R. B. Mask R-CNN. *CoRR*. 2017, abs/1703.06870. Available at: http://arxiv.org/abs/1703.06870.

[14] HOLLEMANS, M. *One-stage object detection* [online]. 2018 [cit. 2021-04-21]. Available at: https://machinethink.net/blog/object-detection/.

[15] HONDA, H. *Digging into Detectron 2 — part 5* [online]. 2020 [cit. 2021-05-05]. Available at: https://medium.com/@hirotoschwert/digging-into-detectron-2-part-5-6e220d762f9.

[16] KARUNAKARAN, D. *Semantic segmentation — Udaity's self-driving car engineer nanodegree* [online]. 2018 [cit. 2021-04-09]. Available at: https://medium.com/intro-to-artificial-intelligence/semantic-segmentation-udaitys-self-driving-car-engineer-nanodegree-c01eb6eaf9d.

[17] KETKAR, N. Introduction to PyTorch. In: *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017, p. 195–208. DOI: 10.1007/978-1-4842-2766-4_12. ISBN 978-1-4842-2766-4. Available at: https://doi.org/10.1007/978-1-4842-2766-4_12.

[18] KRISHNA, M., NEELIMA, M., MANE, H. and MATCHA, V. Image classification using Deep learning. *International Journal of Engineering & Technology*. march 2018, vol. 7, p. 614.

[19] KUMAR, H. *Evaluation metrics for object detection and segmentation: mAP* [online]. 2019 [cit. 2021-04-21]. Available at: https://kharshit.github.io/blog/2019/09/20/evaluation-metrics-for-object-detection-and-segmentation.

[20] LE, J. *The 5 Computer Vision Techniques That Will Change How You See The World* [online]. Apr 2018. Available at: https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b.

[21] LIN, T., DOLLÁR, P., GIRSHICK, R., HE, K., HARIHARAN, B. et al. Feature Pyramid Networks for Object Detection. In: IEEE. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, p. 936–944. DOI: 10.1109/CVPR.2017.106. ISBN 978-1-5386-0458-8.

[22] LIN, T.-Y., GOYAL, P., GIRSHICK, R., HE, K. and DOLLÁR, P. *Focal Loss for Dense Object Detection*. 2018.

[23] LU, X., LI, Q., LI, B. and YAN, J. *MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection*. The Chinese University of Hong Kong, 2020. Available at: https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123590528.pdf.

[24] MAGGIORI, E., TARABALKA, Y., CHARPIAT, G. and ALLIEZ, P. Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark. In: IEEE. *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. 2017. DOI: 10.1109/IGARSS.2017.8127684. ISBN 978-1-5090-4951-6.

[25] Mihajlovic, I. *Everything You Ever Wanted To Know About Computer Vision.* [online]. Apr 2019. Available at: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e.

[26] *About* [online]. OpenCV, 2021 [cit. 2021-04-03]. Available at: https://opencv.org/about/.

[27] O'Shea, K. and Nash, R. *An Introduction to Convolutional Neural Networks.* 2015.

[28] Peemen, M., Mesman, B. and Corporaal, C. Speed sign detection and recognition by convolutional neural networks. In: *Proceedings of the 8th International Automotive Congress.* 2011, p. 162–170.

[29] *About* [online]. PyTorch, 2018 [cit. 2021-04-03]. Available at: https://web.archive.org/web/20180615190804/https://pytorch.org/about/.

[30] Ren, S., He, K., Girshick, R. and Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* 2016.

[31] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. et al. *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression.* 2019.

[32] Sathya, R. and Abraham, A. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence.* february 2013, vol. 2.

[33] Sharma, S., Sharma, S. and Athaiya, A. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology* [online]. April 2020, vol. 4, no. 12, p. 310–316. ISSN 2455-2143. Available at: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf.

[34] STEVENS, E., ANTIGA, L. and VIEHMANN, T. *Deep Learning with PyTorch.* Manning Publications Co., 2020. ISBN 9781617295263.

[35] Uijlings, J. R. R., Sande, K. E. A. van de, Gevers, T. and Smeulders, A. W. M. Selective Search for Object Recognition. *International Journal of Computer Vision.* 2013, vol. 104, no. 2, p. 154–171. Available at: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013.

[36] Vistasp M. K. (ed.), F. A. e. *Structural Health Monitoring of Civil Infrastructure Systems.* Woodhead Publishing, 2009. Civil and Structural Engineering. ISBN 978-1-84569-392-3.

[37] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. and Girshick, R. *Detectron2* [online]. 2019. Available at: https://github.com/facebookresearch/detectron2.

Appendix A

Contents of the included storage media

- *./BP/* - folder with all necessary data for executing the code
- *./BP/Colab Notebooks* - folder with source code in jupyter notebooks
- *./BP/Dataset* - folder which contains train, validation and predictions set
- *./BP/labels.csv* - file with ground-truth labels in csv format
- *./BP/model_final.pth* - final trained model
- *./results/* - images from final predictions and their detections
- *./text/* - necessary files for building PDF from .tex files
- *./README.md* - complete installation guide
- *./random-choose.py* - additional script for random image choosing
- *./projekt.pdf* - textual part