ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bouška** Jméno: **Filip** Osobní číslo: **380719**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Lokalizace a počítání osob na satelitních a leteckých fotografiích

Název diplomové práce anglicky:

Localization and Counting of Humans based on Satellite and Aerial Imagery

Pokyny pro vypracování:

The objectives of the thesis are:
1) Study state-of-the-art methods of object localization in images. Focus on algorithms based on Deep Neural Networks. Also study existing approaches on localization of humans based on satellite or aerial imagery.
2) Select and examine a relevant imagery dataset. Most likely, the dataset will have to be annotated.
3) Design and implement Deep Neural Network-based models for counting and localization of humans in the images.
4) Experiment with different network architectures and optimize metaparameters.
5) Evaluate your models using the selected dataset.

Seznam doporučené literatury:

De Oliveira, Diulhio Candido, and Marco Aurelio Wehrmeister. "Towards Real-Time People Recognition on Aerial Imagery Using Convolutional Neural Networks." Real-Time Distributed Computing (ISORC), 2016 IEEE 19th International Symposium on. IEEE, 2016.
Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.
He, Kaiming, et al. "Mask r-cnn." arXiv preprint arXiv:1703.06870 (2017).
Yosinski, Jason, et al. "How transferable are features in deep neural networks?." Advances in neural information processing systems. 2014.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Drchal, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **23.06.2017** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce:
do konce letního semestru 2018/2019

Ing. Jan Drchal, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Master's thesis

Localization and Counting of Humans based on Satellite and Aerial Imagery

Bc. Filip Bouška

Department of Open Informatics
Supervisor: Ing. Jan Drchal, Ph.D.

May 24, 2018

Acknowledgements

I would like to express my thanks to Ing. Jan Drchal, Ph.D., who supervised my work. Also, I would like to thank my family and friends for support while writing this thesis

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 24, 2018 .....

Czech Technical University in Prague
Faculty of Electrical Engineering

Citation of this thesis

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

Abstrakt

Tato práce se zabývá metodami detekce objektů. Konkrétně lokalizací osob ze satelitních a leteckých snímků. Popisuji zde tvorbu a přípravu datasetu a následně algoritmy a jejich nastavení k získání nejlepšího prediktivního modelu. V práci používám zejména nejmodernější algoritmy založené na hlubokých neuronových sítích (*Faster R-CNN [1], RetinaNet [2]*).

Mnoho prací, které se zabývají zpracováním satelitních či leteckých snímků, řeší úlohy lokalizace větších objektů jako jsou domy, lodě nebo auta. Mnou zvolená úloha je jiná, jelikož se člověk na běžných snímcích jeví pouze jako tečka o rozměru několika pixelů.

Klíčová slova Algoritmy lokalizace objektů, lokalizace lidí, hluboké neuronové sítě, letecké snímky.

Abstract

This thesis is concerned with methods of object detection. Specifically with person localization on satellite and aerial imagery. I describe here the creation and preparation of a dataset and then the algorithms and their settings to obtain the best predictive model. In this work, I use the state of the art algorithms based on deep neural networks (*Faster R-CNN [1], RetinaNet [2]*).

Many papers on processing satellite or aerial imagery deal with the tasks of locating more massive objects such as houses, boats or cars. The work I choose is different because the person in ordinary satellite or aerial pictures only appears as a dot with the size of a few pixels.

Keywords Object localization algorithms, people localization, deep neural networks, aerial images.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Nowadays the amount of data produced by any kind of source is rapidly increasing. The word *data* is becoming more and more pronounced, and some people even say that the world's most valuable resource is no longer oil, but data. This data can be understood as any information of the real world translated into computer code. Sensors of any kind starting with a thermometer, moisture meter and finishing with cameras are used to obtain this information. In this work, let me focus and speak about the visual data. We are living in the era where everybody has a smartphone with a camera capable of shooting videos with 4k resolution. But this is only part of today's progression. Last years have introduced new drones with size fitting in your pocket which can also shoot videos with high resolution and opening whole new perspective of recording. Let's take this even further and talk about satellites. The times when the only satellites circling around the planet Earth were owned by governments of the wealthiest countries is gone. Today there are many private satellites which take photos of any kind and any resolution. From all of these sources of data, we can obtain useful information when we process it correctly. In today's world, we are in a state when a large number of new techniques for handling this data is being introduced. Especially during last few years, we hear a lot about *machine learning* which is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. And all the hype around machine learning wouldn't be possible without a significant amount of data as well as the computational power of today's computers. All these aspects caused that we can solve problems which looked impossible to answer before.

In this thesis, I will focus on working with the visual data. The processing of visual data (but not only) was not very efficient until the arrival of deep neural networks. This happened in 2012 when architecture called

AlexNet [3] based on newly rediscovered convolutional layers won the ImageNet outperforming the second runner-up approach by a significant amount. Breakthrough in this work changed the view on computer vision tasks ever since. Majority of the current state-of-the-art methods in computer vision field come out from AlexNet and push the accuracy even further. When we realize this happened not more than six years ago and that now we have algorithms which are better than human in specific tasks we have to admit it is fascinating. Also, new improvements focused on speed as well as accuracy are proposed with great speed. It all goes with the progress made in the area of graphics cards where one can buy cards powerful enough to train his own deep neural networks for affordable prices. In this thesis, I will speak about technologies used in computer vision, and I will show how to apply the state-of-the-art algorithms to a real-world task.

1.2 Objectives

This thesis aims to design, implement and experimentally evaluate a deep neural network pipeline for detection of people in high resolution aerial or satellite imagery. The work also includes algorithms and techniques for preprocessing the images before the actual training. Another thing I describe in this work is the process of creation of my own dataset which is then used for training and evaluating the gained models. I also examine different state-of-the-art algorithms for object detection, and I experiment with different settings of these algorithm to achieve the best precision of a prediction model. After the best model is chosen, I would like to use it to detect people on a new dataset and visualize them on a map as a heat map.

1.3 Problem statements

In the beginning, let's speak about tasks we are dealing with in computer vision. The first one is object classification where you have a picture with an object on it, and you want to say what object it is. In other words, you want to determine the object's class from some given number of categories. This is the most straightforward problem, and it is a first problem where convolutional neural networks are used. The next one is an extension of the first one. Not only that we want to predict the class but also we want to get the location of that object in the image. Both of these tasks are concerned only with a single object. Now let's talk about problems with multiple objects. Object detection is one of them. We have some image, and we want to find all objects that we are interested in and draw bounding boxes around them. These objects can be of one class, but more often they are of different classes. Also, the sizes of these objects can differ. Another problem people work on is instance segmentation. Where you have some fixed number of categories,

and you want to find all instance of those categories in your image. However, instead of using a box you want to draw contours around each object and identify all pixels belonging to each instance. See figure 1.1 for overview. In this thesis, I am dealing with the problem of object detection. In my case, I would like to detect people on aerial imagery.

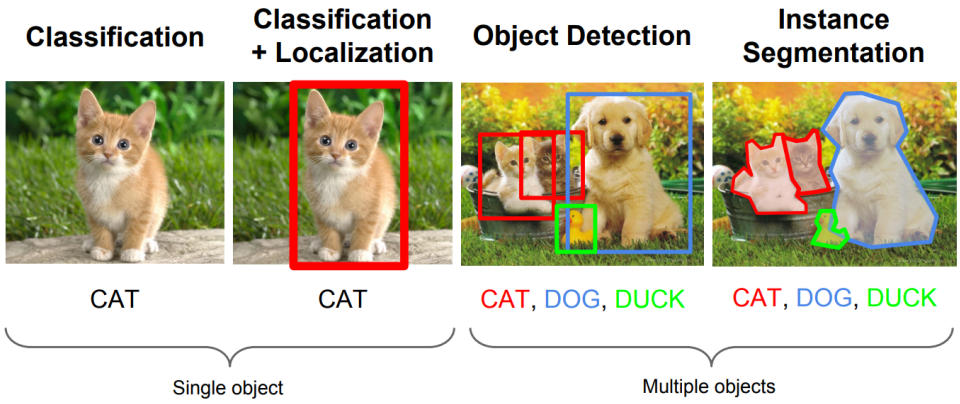


Figure 1.1: Some of the tasks which computer vision field solves. Starting with simply saying what object is on the picture through saying what object is on the picture and its exact location extending to multiple objects and ending up with segmenting the whole image into areas containing searched objects. Image taken from [4].

1.4 Thesis structure

This is a short overview of the thesis structure. The first chapter is this introduction and motivation to the problem and objectives I would like to achieve. The next chapter is about the related work to this field, and you can find some works dealing with satellite images analysis and some works that tackle the problem of people detection. In the chapter after, I describe the technology which stands behind the object detection algorithms and explain some of the key concepts, I have used when working on this thesis. After I introduce these terms, you can read about the actual realization. I write there about the whole process of how I grabbed the problem, and I describe each individual steps I have made. In the next chapter, I show some of the experiments I had made when I was looking for the best prediction model. And the last chapter is the conclusion of what I have achieved.

Chapter 2

Related Work

2.1 State of the Art

During the last few years, we can notice that deep neural networks are becoming the most common technique in the area of the computer vision tasks. We can find a lot of exciting applications of convolutional neural networks for various tasks. The analysis of satellite or aerial images is one of them. Because these images are usually of high resolution, we can examine them to gain interesting information. We can find different works related to this field. For instance, in this work [5], they introduce a pipeline for analyzing historical satellite images for the last 100 years. Another work [6] tracks and targets poverty in developing countries by analyzing satellite photos of these countries. Their method requires only publicly available data and demonstrates how powerful machine learning techniques can be applied in a setting with limited training data. For another instance in this article [7] they detect vehicles in aerial images. They use a region-based convolutional neural network (R-CNN) Faster R-CNN [1] with some proposed modifications. The problem they are solving is that Faster R-CNN struggles to detect small objects and so it is not suitable for detecting cars on high-scale aerial images. Also, they deal with the problem that annotated data to train the network is limited and the manual annotation is generally expensive. To solve this issues, they introduce new region proposal sub-network called accurate-vehicle-proposal-network (AVPN). You can also find other works dealing with detecting cars [8, 9] or segmenting houses [10] and many more. Situation changes when we want in the same way identify people. I did not find any works dealing with this specific problem. There are works on detecting crowds and estimating its density [11] from low flight altitude aerial images. Also, you can find many works that recognize people on pictures taken from UAV (unmanned aerial vehicle) [12, 13, 14] but these works differ in the sense that the analyzed images are not shot from a high height and aren't shot directly from above (orthophotos [15]).

The most common algorithm for analyzing satellite or aerial images are Mask R-CNN [16] which is an algorithm for object segmentation based on Faster R-CNN algorithm [1]. These are so-called two-stage detectors, and their advantage is high precision. On the opposite site stand one-stage detecters like YOLO [17] and its successor YOLO9000 [18]. These algorithms provide a much better speed of inference than two-stage detectors and therefore are suitable for real-time detection. But they also have a drawback in the term of accuracy where they fall behind the two-stage detectors. This used to be true until the introduction of successor algorithm called YOLOv3 [19] which is catching up in accuracy but offering much better inference speed. Another representative of one-stage detectors is RetinaNet [2] which is algorithm I have used mostly in this thesis. I will talk more about some of these algorithms and techniques used for object detection in the next chapter.

Chapter 3

Technology

The area of computer vision has been through significant research in recent few years. Many new approaches have been tried, and a lot of them successfully pushed the precision and speed of given task by a big step. Let's focus on the deep learning which is the approach used in the most cases. It all started with the famous article from Alex Krizhevsky, Geoff Hinton, and Ilya Sutskever that won ImageNet in 2012 [3]. And ever since the convolutional neural networks have become the gold standard for image classification. Nowadays the convolutional neural networks have improved to the point where they outperform humans.

In this first part of this chapter, I would like to explain some of the concepts used in deep neural networks in general. Then I will talk about specific algorithms used specifically for object detection and how they evolved during the time. I will not talk about everything because there are many techniques which are somehow related to this topic. Also, I think that you can find a much better explanation in other sources. For example in *The Deep Learning textbook* [20] from Ian Goodfellow you can find an excellent description of deep learning and the mathematics behind it. Another source worth mentioning is *CS231n: Convolutional Neural Networks for Visual Recognition* [4]. This course is focused on computer vision part of the deep learning and therefore is closely related to this thesis.

3.1 Terms in deep learning

In this section, you will find some of the most important terms and patterns I have encountered or have used when doing this thesis.

ResNet

ResNet. Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015 [3]. It features unique skip connections and a heavy use of batch

normalization. The architecture is also missing fully connected layers at the end of the network. ResNets are currently state-of-the-art Convolutional Neural Network models and are the default choice for using ConvNets in practice. In particular, also see more recent developments that tweak the original architecture from Kaiming He et al. Identity Mappings in Deep Residual Networks (published March 2016) [21].

Adam

Adam [22] is a recently proposed optimization algorithm and is currently recommended as the default algorithm to use. It is a replacement optimization algorithm for stochastic gradient descent for training deep learning models which combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. It is also relatively easy to configure where the default configuration parameters do well on most problems.

Batch Normalization

A recently developed technique by Ioffe and Szegedy called Batch Normalization [23] alleviates many headaches with properly initializing neural networks by explicitly forcing the activations throughout a network to take on a Gaussian unit distribution at the beginning of the training. The core observation is that this is possible because normalization is a simple differentiable operation. It has become a very common practice to use Batch Normalization in neural networks and also the networks that use Batch Normalization are significantly more robust to bad initialization. Additionally, batch normalization can be interpreted as doing preprocessing at every layer of the network, but integrated into the network itself in a differentiable manner.

Keras

Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow. It was developed to make implementing deep learning models as fast and easy as possible for research and development. It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license. Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

- **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.

- **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- **Extensibility**: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- **Python**: No separate model files with custom file formats. Everything is native Python.

3.2 Object detection algorithms

One of the basic methods of object detection is *sliding window*. The sliding window is a rectangular region of fixed width and height (smaller than image's resolution) that moves across an image usually from left to right in a horizontal direction and from top to bottom in a vertical direction. For each of these windows, we would typically take the window region and apply an image classifier to determine if the window has an object that interests us. We can apply this also for detecting multiple objects. However, we have to say that making a prediction on each of window position is slow and computationally expensive. Also, this approach is suitable for detecting objects of the same size, but it does not work very well when we try to detect objects that differ in size. This solves technique called *image pyramids* with which we can create image classifiers that can recognize objects at varying scales and locations in the image. I will talk more about this technique in section 3.4.1. These techniques, while simple, play an absolutely critical role in object detection and image classification.

Since these are fundamental techniques and they do not achieve that high accuracy, I decided to skip reinventing the wheel and moved to the current state-of-the-art algorithms for object detection. I discovered many algorithms. Every one of them offered different benefits as well as drawbacks. When we talk about algorithms in computer vision, we want to know how accurate the algorithm is and how fast it operates. Since we do not require the algorithm to work fast because it will not be used in real time application and since the primary goal is to achieve the highest accuracy we discarded algorithms offering high speed with lower accuracy (OverFeat, SSD, YOLO). These algorithms work in fact in the same way and are called *one-stage detectors*. Another group of object detection algorithms is called *two-stage detectors* and offer higher accuracy at the cost of processing time. In next section, I will describe why I chose two-stage based detector and then in the next section why I switched to another one-stage base detector.

Now let's talk about the most recent methods used in object detection. We can group these algorithms into two groups which differ in the style of proposing the regions where the possible objects could be.

3.3 Two-stage detectors

Until the introduction of RetinaNet [2] the two-stage detectors were the dominant paradigm in modern object detection. Two-stage detector work as follows. The process of object detection is split, as the name says, into two stages. In one stage a sparse set of candidate region proposals is generated. These regions are expected to contain all possible objects on the image. The way this set of proposals is created can differ, and you can read about how it evolved and improved over time further in this chapter. Some of the algorithms used for generating these proposal are *Selective Search* [24], *EdgeBoxes* [25], *DeepMask* [26, 27] or *Region Proposal Network* [1]. Also this way the majority of negative locations is filtered out, and only relatively small amount of regions is passed into the second stage. In the second these regions are classified as one of the foreground classes or as background.

R-CNN

The first significant algorithm that changed the approach to the object detection topic was R-CNN [16]. R-CNN does what we could intuitively imagine doing as well. Use this old classical convolutional network multiple times on various parts of an image and see if there is an object. See figure 3.1 for visualization of the R-CNN pipeline.

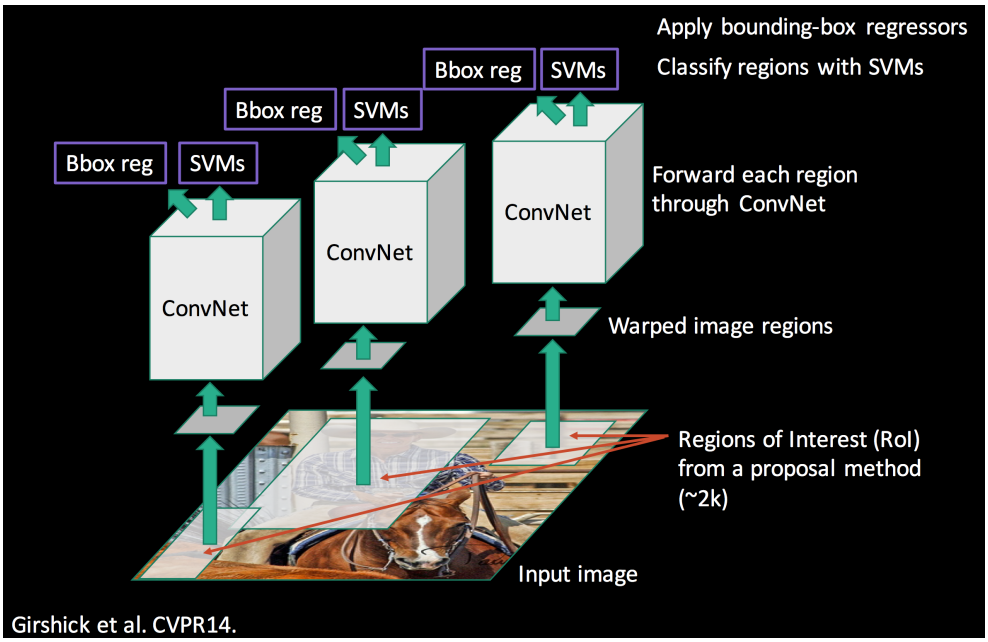


Figure 3.1: The pipeline of the R-CNN algorithm. The most straightforward two-stage detector approach. Image is from [28].

These mentioned parts of images are called bounding boxes or region proposals and are generated with a process called selective search. It looks at the picture through windows of different sizes, and for each size, it tries to group adjacent pixels by texture, color, or intensity to identify objects. When we have this sub-images, R-CNN resize them to standard square size and passes them through the convolutional neural network (AlexNet [3]). However, the network is a little bit modified. R-CNN adds super vector machine (SVM) on the top of the network, and its objective is to classify if there is an object and what object it is. Next step is to add linear regression head parallel to the SVM. Its purpose is to correct the bounding boxes coordinates for given proposals. R-CNN work very well, but there are some drawbacks. It has to make forward pass for every region proposal, and because there are usually about two thousand of them, it is really slow. The second thing is that it is quite difficult to train the whole model because you have to train three separate parts separately - the convolutional neural network which generates image features, the classifier that predicts the class, and the regression model to tighten the bounding boxes.

Fast R-CNN

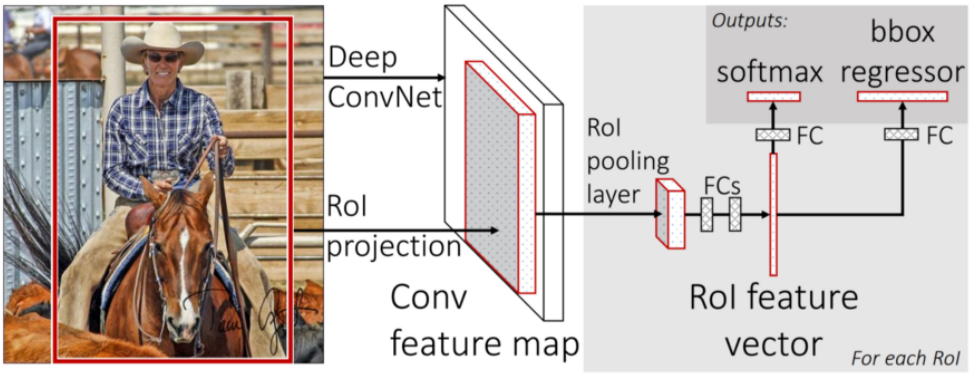


Figure 3.2: The architecture of Fast R-CNN algorithm. It does not process all region proposal through CNN but takes the whole image just once. The network includes two parallel branches, one for classification and one for bounding box regression. Image taken from [29].

A solution of these problems was proposed by the creator of R-CNN Ross Girshick. Symbolically this new technology is called Fast R-CNN [29]. The main idea stays the same, but it also has some neat insights. One of the main is that we do not run every proposed region through the convolutional layers, but we take whole high-resolution image instead. Fast R-CNN introduces a technique called the region of interest pooling, and at its core, RoI pooling shares the forward pass of a convolutional neural network (CNN) for an image

across its subregions. CNN features for each region are then obtained by selecting a corresponding area from the CNN's feature map, and then they are pooled usually using max pooling. The second idea focuses on process to jointly train the whole network. Fast R-CNN replaces the SVM classifier with a softmax layer on the top of the CNN to output a classification. It also adds a linear regression layer parallel to the softmax layer to output bounding box coordinates. So Fast R-CNN uses just one model to compute all desired outputs opposite to three separate models in simple R-CNN: image features (CNN), classification (SVM) and tighten bounding boxes (regressor).

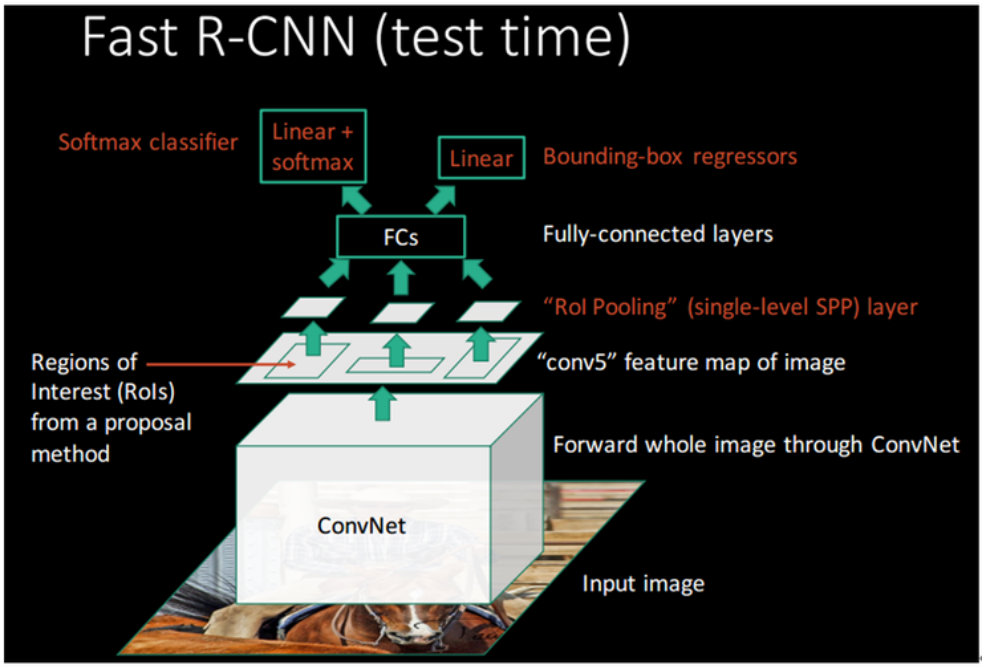


Figure 3.3: The pipeline of Fast R-CNN during inference. The features from convoltional neural network are computed just once and shared among all RoIs to make classifications and regression. Image taken from [29].

Faster R-CNN

All these improvements made the system work much faster than the original R-CNN, but still, it does not work as fast to serve real-time recognition. They found out that the bottlenecking part slowing the whole system down is the region proposal method (selective search [24]). Later researchers found a way to make the region proposal almost cost-free with architecture called Faster R-CNN [1]. Let's speak for a while about how Faster R-CNN generates these region proposals from CNN features. It adds another fully convolutional network on the top of the features of the CNN. This network is called region

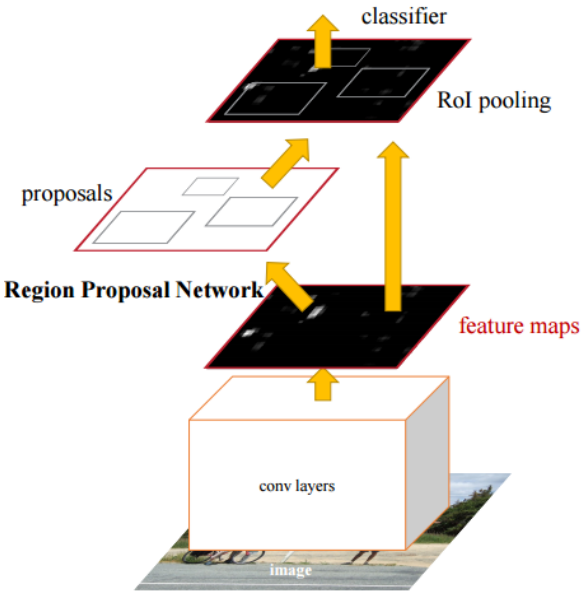


Figure 3.4: Similarly as in Fast R-CNN but now the region proposal are produced by standalone neural sub-network not by any region proposal method. Image is from [1].

proposal network and works as follows. It shifts a sliding window across the CNN feature map and at each position it outputs a certain number of potential bounding boxes and scores for how precise each of those boxes is expected to be. These boxes differ in their sizes as well as their aspect ratios. For instance, when we want to localize human, we know that the box should be a vertical rectangle. After we have these proposal boxes, we then pass each such bounding box that is likely to be an object into Fast R-CNN to generate a classification and tightened bounding boxes.

Mask R-CNN

So far we have seen how to localize objects using methods mentioned above. All of these methods outputs bounding box in which is expected to be some object. They also return percentages of class labels which this object has. Mask R-CNN [16] is different. It extends these techniques to a further level. It does not output just bounding box of an object but exact pixels of each object (object segmentation). Mask R-CNN adds a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object. The branch (in white in the above image), as before, is just a Fully Convolutional Network on top of a CNN based feature map. So it outputs matrix with ones in all locations where the pixel belongs to the object and zeros for the location where it does not.

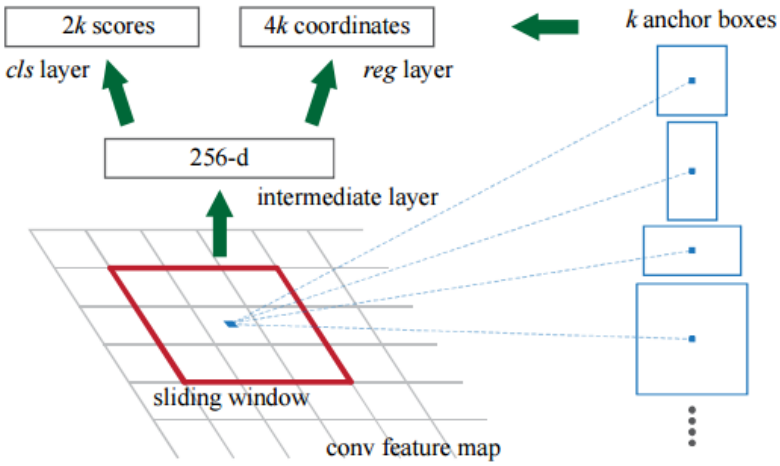


Figure 3.5: The production of region proposal for each of k anchor boxes in the Region Proposal Network. Image is from [1].

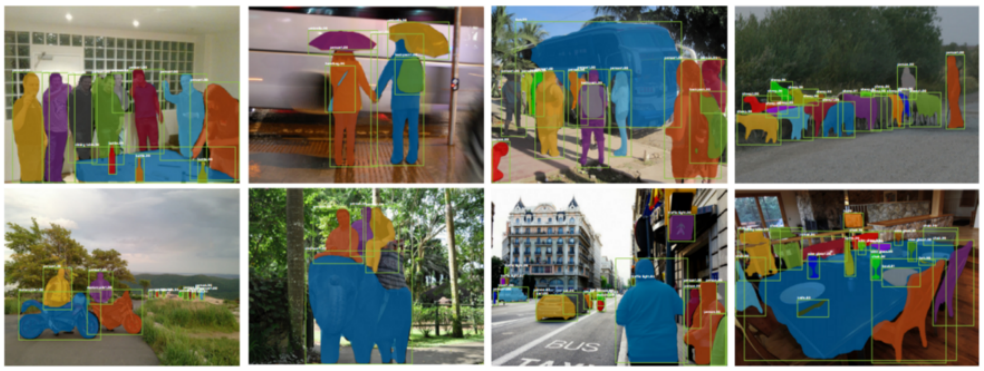


Figure 3.6: Some of the results generated by Mask R-CNN. Not only we get the object's bounding box but also the pixel-wise mask of this object. Image taken from [16].

If we ran it like this without any other modifications, it would generate slightly inaccurate results. Imagine we have an original image of size 256 x 256 pixels and its feature map which is scaled down in size to 50 x 50 pixels. For instance, we want to find an object which is in the original image in the top right corner taking 15x15 pixels. We calculate that pixel in the original image corresponds to $50 \div 256$ pixel in feature map, so the object would take $15 * (25 \div 256) \approx 2.93$ pixels. This value is rounded down to 2 pixels in default RoIPool causing these misalignments. So instead we use RoIAlign which is the technique that uses bilinear interpolation to generate more precise results. After the masks are created, Mask R-CNN runs them through the

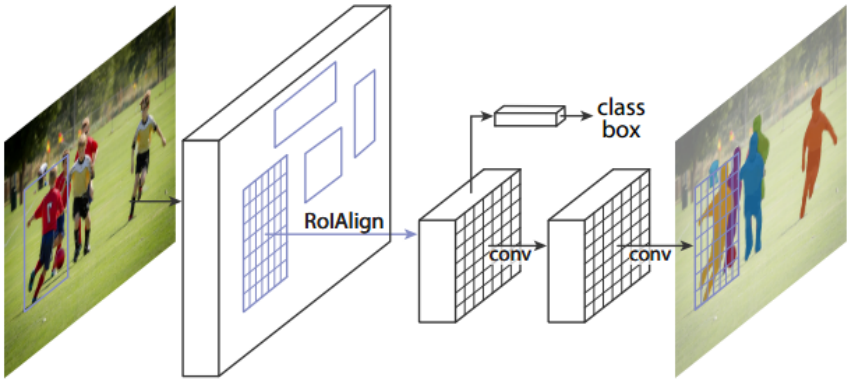


Figure 3.7: In this figure you can see the mask branch which is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Image is from [16].

classification and bounding box regression from Faster R-CNN and generate accurate segmentation.

3.4 One-stage detectors

The main difference between one-stage and two-stage detectors is the set of regions they yield for prediction. As written above the two-stage detector produce only a sparse set of regions, in contrast, the one-stage detectors produce a dense set of regions which cover the whole area of the image. I chose RetinaNet as a representative for one-stage detectors because it surpasses all these two-stage algorithms both in accuracy as well as in speed because it utilizes some of the new concepts about which you can read in following sections.

3.4.1 Feature Pyramid Networks

Recognizing objects at vastly different scales is a fundamental challenge in computer vision. Let me briefly explain the evolution of this pyramid approach to object detection tasks. As you can see in figure 3.8there are several ways of building feature pyramid. I will not talk in much detail about these older types of feature pyramids, and I will focus on the feature pyramid network.

- From the image which is scaled into pyramid are features computed directly which is slow. (a)
- Some of the recent systems make the prediction only on the top layer which is fast but lacks in terms of accuracy. (b)
- Other approach is to take features from the convolutional network and use them as features for the feature pyramid. (c)

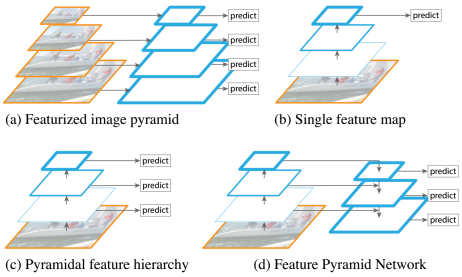


Figure 3.8: The different approaches used for generating the feature pyramid [30].

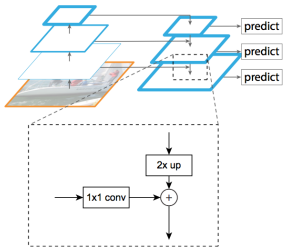


Figure 3.9: The process of the bottom-up and the top-down pathways with lateral connections [30].

- Feature pyramid network. (d)

The old technique introduced in 1984 called *Pyramid methods in image processing* [31] was heavily used as a critical component in the era of hand-engineered features [32] and [33]. Symbolized in paradigm (a) on figure 3.8.

This has been largely replaced by the first type of deep convolutional networks, which are able of representing higher-level semantics as well as they are more robust to scale variance. See image (b) on figure 3.8. However, even convolutional networks use pyramids to get the most accurate results. All recent top entries in the ImageNet [3] and COCO [34] detection challenges use multi-scale testing on featurized image pyramids.

The Single Shot Detector (SSD) was one of the first attempts at using convolutional pyramidal feature hierarchy as if it was a featurized image pyramid. Image (c) on figure 3.8. This approach is fast but lack in terms of accuracy and does not work for detecting small objects.

Feature pyramid network is a structure for multiscale object detection introduced in [30]. It produces feature maps of different scale on multiple levels built into network serving as a regressor and classifier. Also, this structure can be used with any backbone convolutional architecture. They have used ResNet [35] in the article. Let me explain how the whole process of building this pyramid is done.

Bottom-up pathway

In this phase, the image is fed in traditional forward fashion through the backbone convolutional network, which computes a feature hierarchy consisting of feature maps at different scales. In the backbone network, there are many layers producing output with the same size and they are grouped together and called stages. For each one of these stages, they assigned one level of feature pyramid. They took the output of the last layer of each stage to be the input for the corresponding level of the feature pyramid. It is because

the deepest layer contains the strongest features from the whole stage. Let's talk about the specific backbone architecture which they used in the article - about ResNets. They used the feature activations output by each stage's last residual block. They named these outputs of residual blocks as C_2, C_3, C_4, C_5 standing for conv2, conv3, conv4, and conv5 outputs. And corresponding strides have values of 4, 8, 16, 32 pixels concerning the input image. They did not use the output from conv1 due to its large memory footprint.

Top-down pathway and lateral connections

During the top-down phase, the features are processed from higher level pyramid levels to lower levels. It constructs higher resolution features by upsampling spatially coarser, but semantically stronger, feature maps from the higher levels. This way each level starting from the top is upsampled to double size (using nearest neighbor). These features maps are then enhanced with features from the bottom-up pathway via the lateral connections. During this enhancement, each of the feature maps from top-down pathway is merged with feature map from bottom-down of the corresponding spatial size. On each of the map from bottom-up pathway is applied 1x1 convolution to reduce channel dimension. They also applied 3 x 3 convolution on each merged map to reduce the effect of aliasing. The final set of this process is feature pyramid consisting from feature maps P_2, P_3, P_4, P_5 corresponding to C_2, C_3, C_4, C_5 from the bottom-up pathway. The dimension of each feature map is set to the same value because they share the classification and regression layers. The optimal depth of the feature map was found to be 256 channels. The main idea was to keep the whole structure as simple as possible, and it was tested that more complex components yielded only slightly better results.

3.4.2 Focal loss

The Focal Loss is designed to address the one-stage object detection problems with the imbalance where there is an extreme number of background classes and just a little amount of foreground classes. This is because one-stage detector produces a dense set of locations regularly sampled across an image. In practice, this often amounts to enumerating hundreds of thousands of locations that densely cover spatial positions, scales and aspect ratios. And the majority of these is just background, and only a few locations contain objects. This imbalance causes two problems. The training is inefficient as most locations are easy negatives that contribute no useful signal and the massive amount of these negative examples overwhelm the training and causes the models to degenerate. Some solutions to this problem were introduced [] but the focal loss is tackling this problem in a more simple way without sampling the hard examples or reweighing schemes.

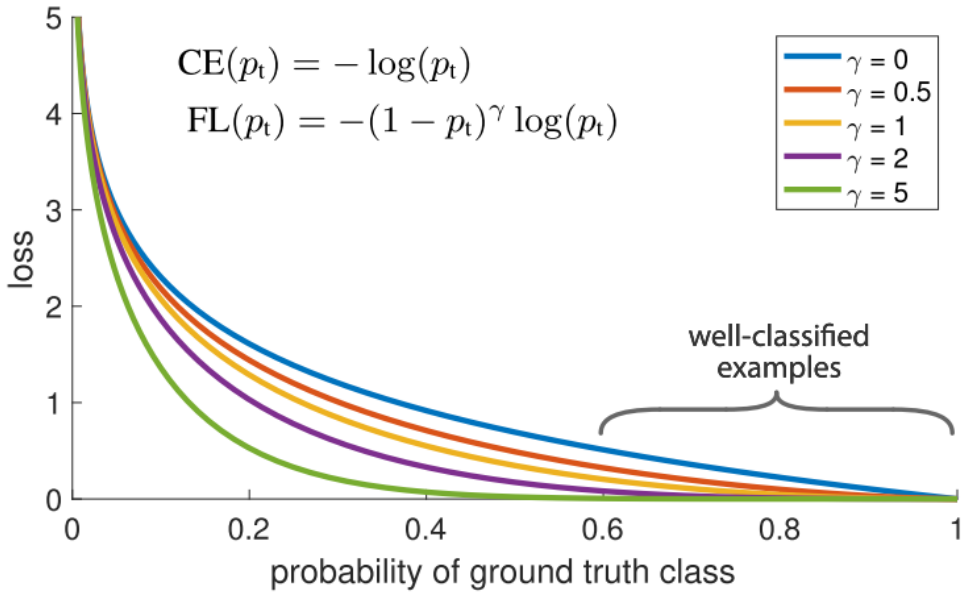


Figure 3.10: On this figure you can see visualized focal loss function for different values of parameter γ. The bigger γ is, the smaller is the loss for well-classified examples. When the γ is equal to zero focal loss is reduced to standard cross entropy (the top blue curve). As you can in this case even though every easy example receives loss with low magnitude when it is summed over a large number of examples it overwhelms the rare classes. Image taken from [2].

$$\mathrm{FL}(p_\mathrm{t}) = -(1-p_\mathrm{t})^{\gamma}\log(p_\mathrm{t}) \tag{3.1}$$

The focal loss is based on standard cross entropy loss, and you can see its definition of the equation 3.1. It add new modulating factor $(1-p_t)^\gamma$ with meta parameter γ. This parameter γ can be tunned and takes values greater than zero. You can see the visualization of the focal loss for five different values of γ on figure 3.10. As you can see the focal loss down-weights easy examples and advantages the hard examples. This causes that the model learns (gain information) more from these hard examples and partially avoids the easy examples which are in the majority. So for instance when an example is misclassified and p_t is small, the modulating factor is near 1, and the loss is unaffected. When p_t goes to 1, the factor goes to 0, and the loss for this well-classified example is down-weighted. For instance, with $\gamma = 2$ an example classified with $p_t = 0.9$ would have 100x lower loss compared to cross entropy and with $p_t = 0.968$ it would have 1000x lower loss. This causes that the model is learning the most from the misclassified examples. In the original

article, they also discuss another extension to focal loss and other approaches for addressing the class imbalance, and you can read about that there.

3.5 RetinaNet

Now let's talk about object detection architecture that I have based my thesis on. This architecture called RetinaNet [2] is currently the best algorithm for object detection achieving highest accuracy without trading off the processing speed. It is the first one-stage detector which is achieving the accuracy of two-stage detectors but remaining simplicity and speed of inference. Few months before I was writing this, a brand new algorithm called YOLOv3 was introduced. This algorithm achieves comparable accuracy but with much better inference times. But, for now, let's talk about RetinaNet and leave YOLOv3 as a nice point for future work. RetinaNet is a network which takes advantages of all above-mentioned techniques. It forms a unified network consisting of several parts. It is backbone network responsible for computing feature map from the input image and two task-specific subnetworks. One for object classification and the second one for bounding box regression. In this section, I will describe how they work and what is their purpose.

Feature Pyramid Network Backbone

Similar to the feature pyramid network described in section 3.4.1 but with some minor modifications. To recapitulate, this feature pyramid network augments the standard convolution network with top-down pathway and lateral connections forming multi-scale feature pyramid. With this pyramid, it is possible to detect objects with various sizes and aspect ratios. The backbone architecture used is ResNet, and the feature pyramid is placed on the top of it. They made these light modification to improve speed: P2 pyramid layer was discarded entirely due to computational reasons, P6 pyramid level is created differently, and they added P7 for detecting large objects. You can read more details in the article.

Anchors

In Faster R-CNN the region proposal network works as follow. They move the sliding window of predefined scales and aspect ratios called anchors on the top of the convolutional layer. For each of this anchor (with different position, size and aspect ratio) the features are sent to regression and classification heads to predict if there is an object and to tighten the bounding box around it. Anchors in RetinaNet work similarly with the difference that they are not slid only across single-scaled feature map, but they attach heads of the same design to each of the feature pyramid levels. With this configuration, we can detect an object of different scales even when we use one-sized anchors (due to

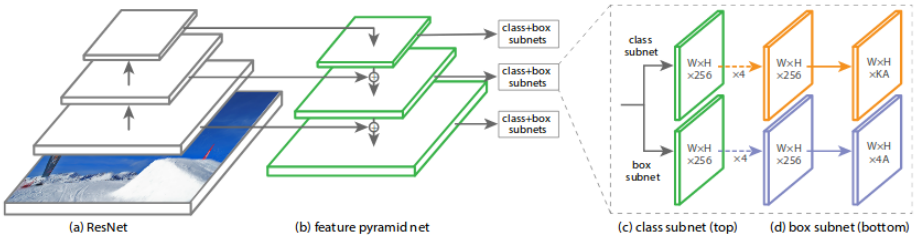


Figure 3.11: The RetinaNet structure. You can see how the feature pyramid is formed next to the convolutional network and how it yields two subnetworks for classification and regresion. Image taken from [2].

the function of feature pyramid). But it turned out that usage of multi-scale anchors with the feature pyramid yields better results. Specifically, they used anchors with tree aspect ratios {1:2, 1:1, 2:1} and three sizes $\{2^0, 2^{1/3}, 2^{2/3}\}$ according to set of three aspect ratios for each of the pyramid levels. With this configuration, there are nine anchors for each of the pyramid level capable of detecting objects with scale range from 32 to 813 pixels with respect to the input image.

Classification network

Classification sub-network is fully connected network attached to each level of the feature pyramid network. Its purpose is to predict the probability of object presence at each spatial position for each of the anchors and each object class. Parameters of this network are shared across all pyramid levels. Take a look at picture (c) on figure 3.11. After taking input feature map from a given pyramid level, it performs some convolutional transformation and finishing by applying sigmoid activation to output binary prediction for each of the possible classes and each of the anchors.

Regression network

In parallel with the classification sub-network, there is regression sub-network. It works almost identically with one difference. It produces four linear outputs for each of the anchors. Each of this four number is predicted offset between the anchor and ground-truth bounding box.

Chapter 4

Realization

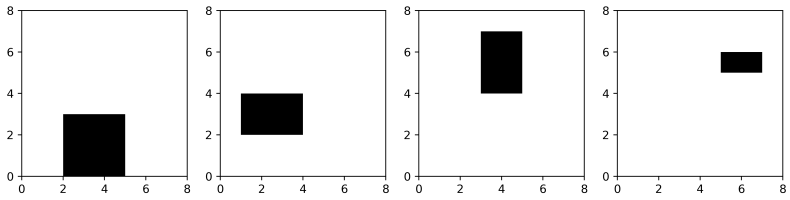
In the following chapter, I will describe how I proceeded with the realization. This first part covers the basic experiments. I started with detection of a single simple object then I tried to detect multiple simple objects to find out how the whole pipeline of object detection works. In the following section, I gripped the problem in bigger and more complex scale. I also had to create my own dataset because no other suitable data were available. After that, I used state-of-the-art algorithm *RetinaNet* [2, 36] and I utilized it for training the prediction model on my dataset. After experimenting with various algorithm settings and data modifications, I selected the best model. With this model, I detected people on new images and visualized them on a map.

When I started working on this thesis, I did not know quite anything about neural networks. The first step I have made was to watch recorded lectures from Andrej Karpathy and his team on Stanford University. The course CS231n: Convolutional Neural Networks for Visual Recognition gave me excellent introduction as well as profound information on this topic. This course covers simple building blocks of neural networks from linear classification through convolution layers and, for me, the essential part object detection and localization. In the following chapter, I will show you the process of how I approached this problem. I will describe individual steps from starting with fundamental tasks followed by more complex and ending by accomplishing this thesis's goals.

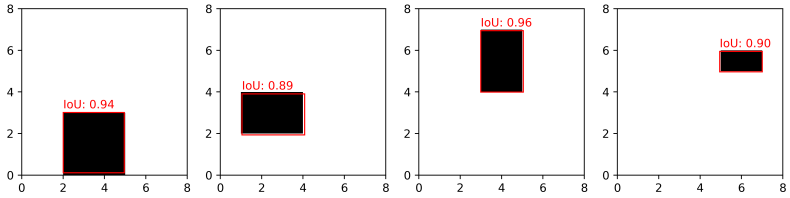
4.1 Simple detection

To get understanding how object detection works I started by detecting an only simple object. First thing I detected was a black rectangle on a white background. In this simple task, we would like to get the position of the rectangle on the image. This position can be described by four numbers (i.e., coordinates of the left top corner of the bounding box and object's width and height). It is a task of a simple linear regression where we predict four

continuous numbers. To accomplish this task, only a few layers without any convolutional layer were sufficient. We can also predict object's class (square, rectangle) by appending classification layer on the top of the network next to the regression head. Also, you don't need to bother with any annotations because as you generate these rectangles, you simply save their positions as well.



Simple objects I tried to detect in the beginning.



Predicted bounding boxes with computed intersection over union.

Figure 4.1: I used rectangles because it was straightforward to generate them. Also to detect these, one needs only simple model without any complicated layers.

```
model = Sequential([
        Dense(200, input_dim=64),
        Activation('relu'),
        Dropout(0.2),
        Dense(4)
    ])
model.compile('adadelta', 'mse')
```

Listing 1: Example of simple Keras model

When I found out that this task is straightforward, I moved to detection of multiple objects placed on the same image. It may sound that it will not

be much harder but to detect more than one object we have to use more sophisticated methods. Usually, we don't know how many objects will be in the image so we can't hardcode the network to predict the coordinates.

4.1.1 Multiple objects detection

As explained in the previous chapter 3.2 there is quite a lot of algorithms for object detection. And I decided to use one of them for my work because trying to build the whole object detection pipeline all by myself would be laborious and would take a long time. Also, this thesis should not be about creating new or innovative algorithm but to apply some already existing algorithm to a practical task. That is why I chose to find an implementation of some of the state-of-the-art algorithms that were public on GIT. I discovered implementation of Faster R-CNN written in python and using Keras framework [37]. I chose Faster R-CNN because it was an algorithm with the highest accuracy at that time. Also when deciding whether to use one-stage or two-stage detector, I went with the two-stage because as I already said it offered higher accuracy. Let me explain some other reasons why I decide to use a two-stage algorithm. One of the reason is that I wanted to detect objects (people) on stationary images and not on video, so I do not mind if the prediction takes arbitrary time. This assumption can change if we would like to apply the prediction on some kind of video feed (for example flying drone or camera situated on some very high place). But I think this is not so common. Another reason is that if we work with some satellite or aerial images dataset, we can make the prediction just once and have all the data annotated by the algorithm. Also, these images are updated most often just once per year and maybe even less often. It is different than for example when we are detecting an object on classic photos because the number of normal photos is enormous (people taking photos and videos with their smartphones etc.). Though the accuracy was my main concern, I also required the algorithm to be tolerably fast because from the practical aspect I decided to do all the training on my home personal computer (you can see the configuration specs in section table 5.1).

As described in 3.3 Faster R-CNN fulfills all needed requirements. You may ask why not to use the newer Mask R-CNN. The main reason is that it would require huge effort to annotate all the data. We would need to have not only bounding box around each annotation but also pixel-wise segmentation of the object inside the annotation. It would mean that I would have to examine if each pixel is a segment of the object and as only the decision if the given dot on the image is a person or not was complicated by itself. I find that object segmentation is not bringing any added value when detecting people. The primary goal of this thesis is to see the density of people over given area so if we have people recognized only as points or small squares it is sufficient. That is why I think that the regular bounding boxes are meaningful enough. Also, we can assume that each person, in fact, takes the same area and utilize this

when setting up the algorithm. On the other hand, segmentation makes sense when detecting bigger objects like cars, buildings or when you want models that should identify for example urban area, water or some other continuous areas where we want to find out the exact area covered by these type of objects. But, again, it is not the case of this work.

When I started working with Faster R-CNN implementation, I did not have any specific dataset to work on. I was trying to find out if there exists some appropriate dataset I could use in my work but unfortunately, I did not find anything. I did not even find any work on processing satellite or aerial imagery to detect people. I had to annotate my own dataset, and you can read about that in the following section. For now, let's talk about experiments I had made before I started to create the dataset. I wanted to make sure that Faster R-CNN algorithm is capable of detecting very many small objects because I did not want to go through all the cumbersome annotation process to find out that algorithm is not able to work correctly with this kind of data. I wanted to somehow test the algorithm without much effort. So similarly as when detecting only one rectangle, I created an image with many much smaller squares, as shown in 4.2.

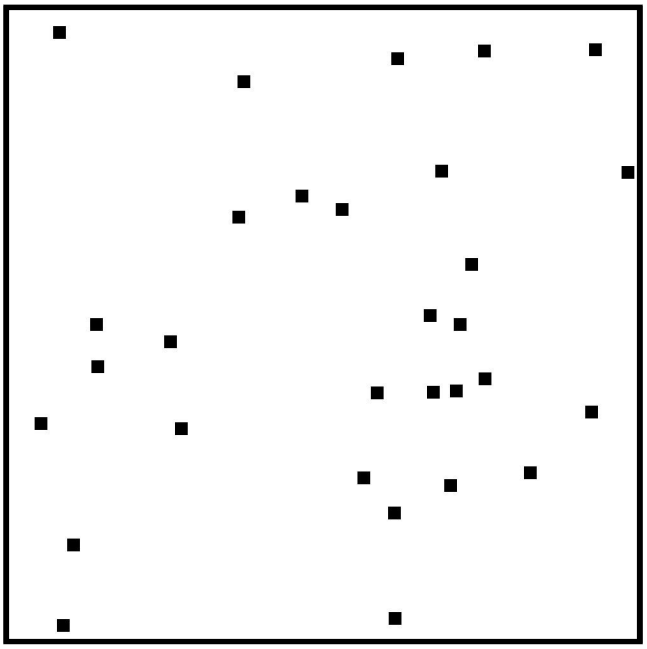


Figure 4.2: One example of the images I generated for testing purposes when I started trying to detect multiple objects with Faster R-CNN algorithm.

When you generate images like this, it has the advantage that you logically also get the annotations of all the squares. Thus you can quickly determine if the algorithm works as you expect and that the prediction model is learning to do the task you train it for. After I generated the data, I started to train

the network. For the training, I used a hundred images each containing thirty randomly placed squares. After just one epoch of training, the model was able to produce some reasonable predictions. You can see these in figure 4.3. These predictions were not perfect, but we have to consider that the model was trained only for one epoch without any other modifications. I am sure if we investigate the settings of the algorithm more deeply we will be able to achieve better accuracy. But this was sufficient experiment for me to continue using this algorithm for a more complex task like human detection. Before I start to describe the steps I took after, I have to introduce you to the data I have used. Also, I have to describe how have I annotate this data. You can read about this in next section.

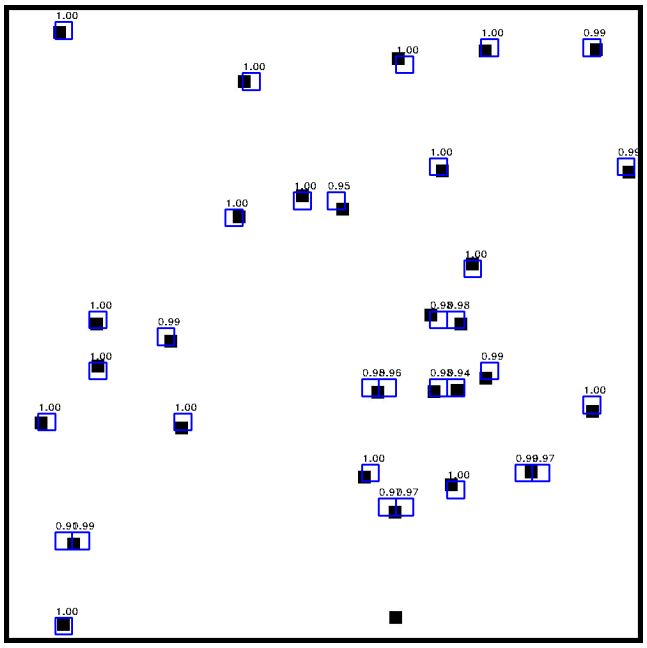


Figure 4.3: Visualized bounding boxes obtained from the prediction model. As you can see the predictions are not perfect and even there is no detection for the square on the bottom of the picture.

4.2 Dataset creation

In this chapter, I will describe the data I have worked with. Further, I will depict the process of annotation of this data and the complications I have encountered.

When I was looking for proper data, I discovered many sources where to get data from. This work should deal with, as its title says, *Localization and counting of humans based on satellite and aerial imagery.* After examining

Figure 4.4: This is how the image with annotated people looks like. (timelapse from annotation process: https://youtu.be/i-JvNql9lJg)

sources providing satellite imagery, I discovered that there is no chance to use this data for localizing people. I have found no resource which could provide photos with such high-resolution to even be able to see people on them. I had decided to work only with aerial imagery. Usually, aerial images are with a precision of one pixel on the image to be equal to fifty centimeters in real. As you can guess these photos are also not suitable for this task because we can assume that average person takes the area of about 50 x 50 centimeters and so he would be equal to one pixel. However, fortunately sometimes, you can also find more precise shots with a precision of one-pixel corresponding to ten centimeters. I ended up by only working with this kind of aerial imagery. The first attempt I have made was that I made about ten screen-shots from aerial images on Mapy.cz [38] and annotated people on them. As expected, this small amount of data was not sufficient to produce any good results.

Then I decided to approach this problem in bigger scale because as it is known the more data, we have the better is the result. This especially holds when dealing with deep neural networks. As my hometown is Prague, I decided to work with orthophotos [15] provided by Geoportal Prague [39]. This official Prague website offers open data of any kind including aerial photos.

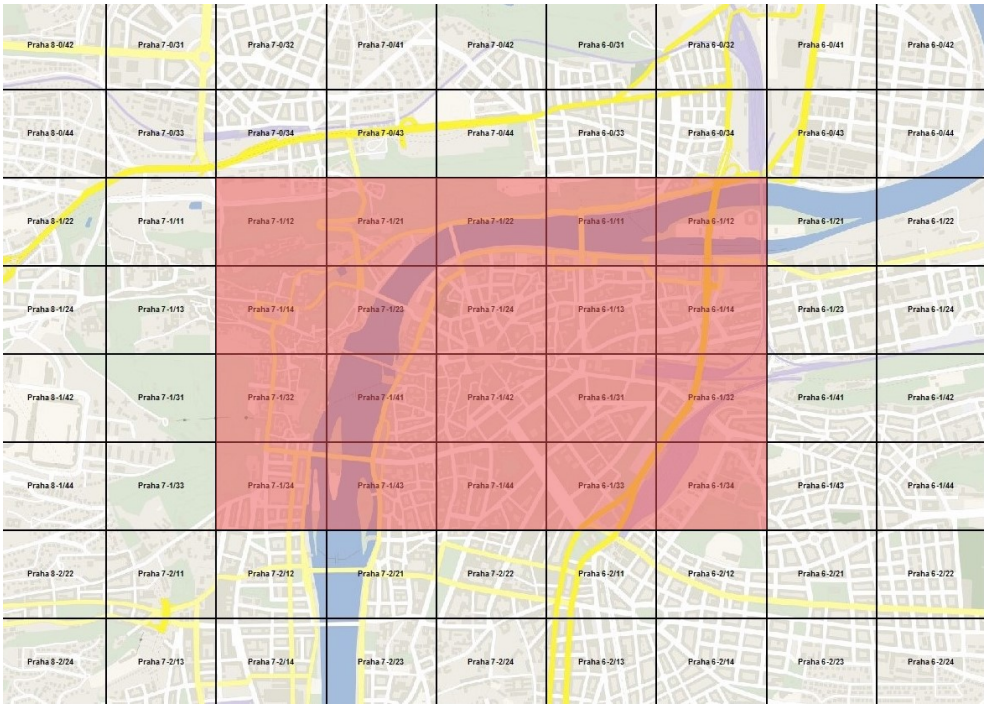


Figure 4.5: Map of Prague with highlighted area of processed images.

I downloaded the whole dataset of size about 8 GB. This dataset consists of about 1500 images of Prague and its surrounding areas. Size of every image is 6250 x 5000 pixels. For my purposes, I have used only 20 images covering the center of Prague because I expected to find the most people in this area. On figure 4.5 you can see the exact area I have used displayed in red. I divided each of these twenty images to other twenty-five sub-images, so I did not have to process such high-resolution images. Look on the figure 4.6 to see how this smaller part of the original image looks like. Also, it is good to mention that for each of the original image there was another file containing geolocation information. This information is useful when we have predictions from the final working model, and we want to match them with the real world longitude and latitude. I will talk about some of these applications further in this chapter.

4.2.1 Annotation utility

After searching for some utility that will allow me to annotate my dataset in a reasonable time, I decided to write my own program. All available tools that I have found were too complicated and it took a lot of time to make an annotation. I needed to annotate five hundred images with a resolution of 1250 x 1000 pixels containing thousands of people. After checking some interesting areas, with a lot of people, like Charles Bridge, I have found out that the

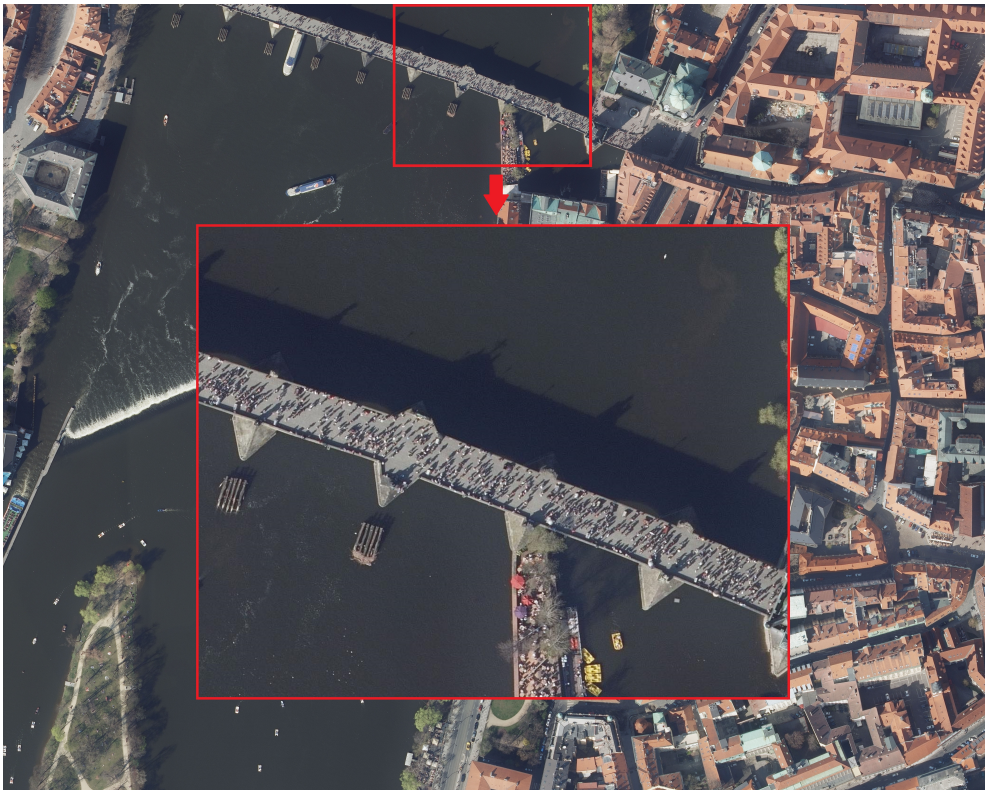


Figure 4.6: Original sized image with area of image I have used when doing annotations marked with red rectangle.

annotation tool has to be quite fast. In figures 4.7 you can see some of the more complicated images I had to annotate. When I subjectively summarize the whole process of annotation I have to say that it was really arduous. Let me share some observations from it. You can easily detect people standing on a sidewalk because the background is coherent and of the same color. So you can nicely see the shadow each person cast. These shadows were helpful when I wanted to find out how many people are there because they were usually bigger than just to person from above. Another thing is that people sitting on the grass or on benches are harder to annotate because their shadows are smaller and do not have the specific shape. Next complication I was facing was when the people were in shadow or even worse when they were under the branches of a tree. At the first attempt, I was annotating all people with only one class. So the nicely visible person on a free area was assigned the same class as the barely visible person in the shadow. I quickly realized that this would not work and decided to propose more classes. I ended up using six classes. It is difficult to describe these classes with words but let me try. One for the cleanest and most obvious person, second for still pretty clear but little less obvious person, third for people where you are not that sure if it is a

person or the person is partially obscured, and the last of this type is class for people where you are not sure at all (for example people in the shadow). The other two classes were a little bit different. One class was for people sitting on a bench or on the ground and during the annotation process I decided to discontinue with this class and just annotate sitting people depending on how good they were visible. The last class was for crowds of people where it was difficult to determine each individual person and I just made bigger bounding box of the crowd. In the end I also decided to discard this class and I annotated every individual person in the crowd. You can find more information in section 4.2.2.

When I was annotating the images, the distinction between these classes was not always obvious and easy. Sometimes I had to follow my intuition, and I have to admit that during the whole time my decision-making which class to assign may have changed so the annotations are in fact not perfect and some revision would be valuable. This revision is one point of the possible future work. This partition into classes was very useful because I could filter out certain classes and train the model only with specific data. I will talk more about this later.

Figure 4.7: Example image without annotations.

I do not know the exact time I have spent annotating all the images, but

for example, the mentioned image 4.7 took me around thirty minutes to finish. And you can look on the figure 4.8 to see the annotated picture. You can also find a time-lapse video of me annotating one part of the Charles bridge in the appendix. Fortunately, a lot of other images were not that complex and did not contain that many people. I have spent a lot of time on this, and I had to force myself to keep going and keep annotating because as you can imagine this work was not that much fun. But it was necessary to finalize it because without the dataset I could not start working on the essential part of this thesis.

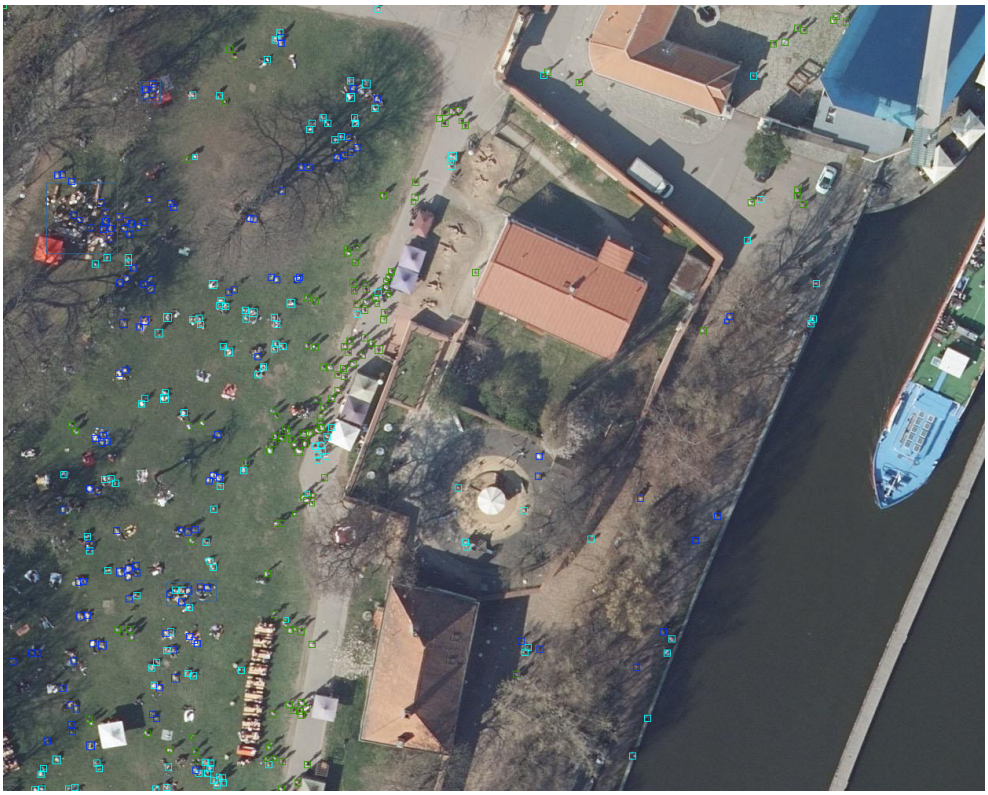


Figure 4.8: Example image with annotations.

Now let's talk about some technical details of my implementation of the annotation utility. In general, the utility is simple python script iterating over all pictures in given directory. It shows the image and allows you to double-click on the center of an object to create a bounding box around it. You can also adjust the size of that box to make the object fit better. I decided to choose this setting against clicking first on the top left corner and then clicking on the bottom right corner of the object to get its bounding box. For my specific usage when I have to annotate massive amount of objects of very similar size is this approach much faster. You can also show or hide all bounding boxes from the current image. This is extremely helpful when

annotating objects close to each other. Further, I bound numbers from one to six to save current bounding box with the corresponding class. When I tried to start annotating the data, I quickly realized that this is not possible to do without some zooming. All the people appeared extremely small, and it was complicated to make the right bounding box around them. In the first version of my utility, I was resizing my images so I could better see the objects. This worked quite nice until I realized that my algorithm is generating some inaccuracies caused by the resizing. Because the average size of a bounding box was around 8 pixels and it is important to store all annotations with pixel precision. Instead of trying to repair my algorithm I was looking for some other ways to solve this problem. The first thing that comes to my head was to use default Windows 10 Magnifier. It allows to you zoom by a specific percentage and you see the zoomed area around your mouse cursor. This proved to be very functional. I could change the zoom level easily with provided keyboard shortcuts, and I did not have to worry about changing the original data in the first place. Just one perception, when I zoomed too much, it was hard to distinguish persons it was the same when I zoomed just a little. After some practice, I have found the proper amount of zoom that worked the best for me.

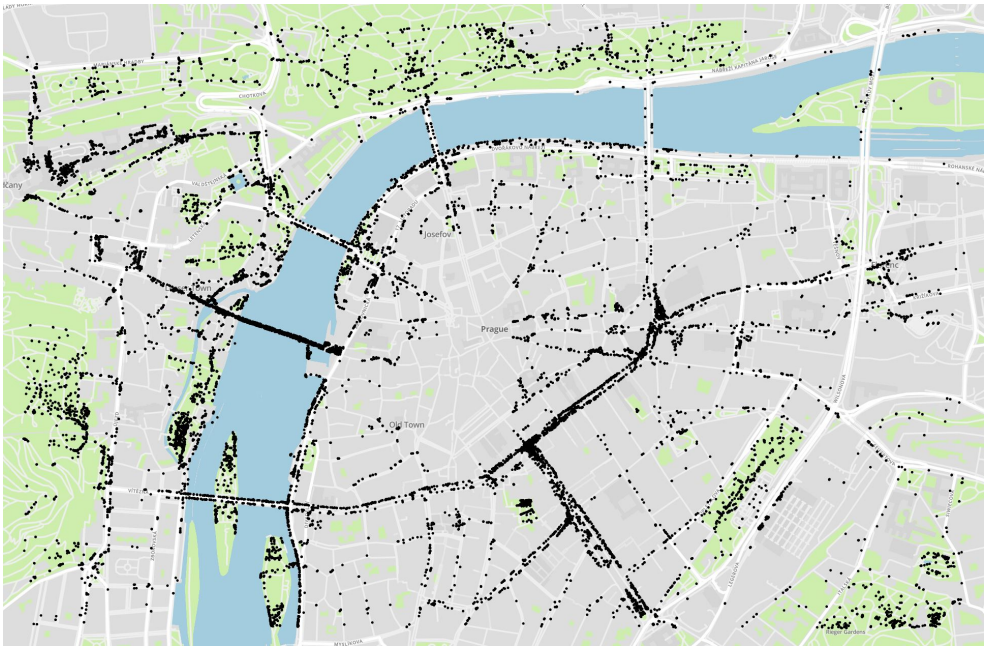


Figure 4.9: All annotations visualized on the map. You can there are no dots in the middle. I had to discard images of this aerea because they were blurred and not suitable for annotation.

4.2.2 Dataset information

In this section you can find some information about the dataset. I have made exactly 16 579 annotations with average bounding box size of 8 pixels. There was 298 annotations with the class *cleanest* (4.10a), 7 732 annotations with the class *clean* (4.10b), 3 752 annotations with the class *not clean* (4.10c), 4 271 annotations with class *uncertain* (4.10d) and 524 annotations with class *bench* or *sitting*. You can see all these annotations visualized in Mapbox [40] on figure 4.9. Notice the zone in the center contain no dots. I had to discard images from this area because they were blurred and thus not suitable. You can also look at figure 4.10 to see how each of the mentioned classes differs. Mind that each of the annotation is in average just 8 x 8 pixels big so the resolution is small.

(a) Examples of the cleanest person annotations.

(b) Examples of still pretty clear person annotations.

(c) Examples of the worse or obscured person annotations.

(d) Examples of badly visible and uncertain annotations.

Figure 4.10: On this figure you can see four different classes I have used for distinguishing different types of annotations.

4.3 Training with the RetinaNet

As soon as I finished creating the dataset I tried to train the model using Faster R-CNN algorithm. In the beginning, I trained the model just for one epoch to see if everything works as it should. For example, I watched if the loss is decreasing and if the algorithm is not generating any errors. During the training, everything looked alright, but when I tried to make some prediction, it did not work at all. I even tried training the model during the whole night for about 10 hours which I expected to be sufficient for this dataset, and I did not obtain any satisfactory results. At this time a new implementation of RetinaNet algorithm appeared and I decided to give it a try.

In this section, I will describe how applied the techniques and procedures explained in previous chapters of this thesis. I will explain how I proceeded with the people detection using the mentioned state-of-the-art algorithm RetinaNet. Also, you will find here some insights and ideas I have made to achieve the best accuracy with my prediction model. When I started using the RetinaNet algorithm, I was not successful as well. I just wanted to try to feed the original annotated data to the algorithm without doing any modifications on it. My intention was to find out if the algorithm will work just like that on the first attempt. I wanted to largely overfit the model on only a few images to see if the algorithm is working correctly. But during the training, the regression loss was settled at value zero and was not moving. I have done some research to find out that the ratio between the size of the image and the size of annotated bounding boxes was disproportionate. For detecting objects of different scale, the RetinaNet uses the feature pyramid network. As explained in section3.4.1 this feature pyramid network slides the window of different size (corresponding to the pyramid level) across the whole image. When we detect objects in more common and classical sense it is expected that the objects will take a bigger area of the image and the low level (high resolution) layers of the feature pyramid network is not used. But in this case, the objects are much smaller, and the network was not able to learn anything from the annotated dataset because the smallest size of an object that can be detected by feature pyramid network is about 32 x 32 pixels which is four times bigger than bounding box of our objects.

4.3.1 Splitting original images

When trying to solve this issue, I came up with two possibilities. The first one is more straightforward and did not require any modifications to the network itself, so I started with it. While the images I used so far are still of a quite big resolution, a tried to divide them once again to achieve that the network will process smaller images. That means that it will upscale each image to the default dimension of the network. This way each of the annotation is upscaled as well and becomes detectable. I experimented with various image

dimensions, and I found out that when I divide the original image into quarters (313 x 250 px), it is the marginal size when the network starts to match some of its anchors with the annotations. To make sure every annotation is captured with some anchor I divided the images into fifths resulting in images of size 250 x 200 pixels. This way I produced 12500 small images as a new dataset (from each original image I had twenty-five new ones). I also transformed the original annotation coordinates for each person. When I was doing the image division I have used two nested for cycles to iterate through each of the five *rows* and five *columns* of the original picture. Same as when I divided the most original images from Geoportal. I took indexes from these for cycles and used them as a new name for each of the new small images (starting from zero). I also prefixed each of these names with the name of the original image. This is an example of how the name looks like *Praha_7_1_14__2_2__2_4.jpg*. Part *Praha_7_1_14* is the name of the most original image. It is followed by two underscores then we have two indexes (the first corresponds to the row and the second one to the column) from the first division followed by two underscores with two indexes (first for the row second for the column) from the last division to the smallest images. This way I was able to store the information about the origin of each of the new smaller images. The last step of this division was that I then iterated over all annotations from the original image and calculated new coordinates as well as to which of the new small image each annotation belongs (using the indexes).

One complication that arises when splitting one big image into more smaller images is that we are somehow losing the objects placed right in the line of the cut. Specifically, when dividing the image into twenty-five parts, we have a grid with eight lines crossing the original image. This may sound like a trifle, but many objects can be exactly or near this crossing line. The solution can be that instead of cutting the image into parts without overlapping we can stride the cropping window by half of the new image size creating new parts that overlap the area where else would be the cut. Because I was satisfied even with the result without this proposed improvement, I did not implement it. I propose this as a nice to have improvement for future work.

On figure 4.11 you can see anchors generated by feature pyramid network from the new smaller image (250 x 200 px) assigned to each of the ground truth annotations. As you can see, there are multiple anchors for each ground truth annotation. These anchors differ in scale and in a ratio (some are square, and some are rectangle). This kind of visualization helped me to see if there exist some anchor boxes that can match the annotations during the training and if there exist any it means that the algorithm is capable of training from these objects on this specific image.

Figure 4.11: You can see the visualization of anchor boxes for each of the annotation generated by feature pyramid network displayed with cyan color. These boxes vary in scale and ratios. The original annotation bounding boxes are shown with green squares. This is shown on an image with the smallest resolution I have used. As you can see each of the annotations has a rich amount of anchors.

4.3.2 RetinaNet modification

The second possibility required some changes in RetinaNet structure. My goal was to make the network also work with the original higher resolution images without a need to downsample them as described in the previous section. To ensure that the network can also learn from higher resolution images I needed to modify the feature pyramid network to also produce lower sized anchors. As described in section 3.5 the smallest object that RetinaNet can detect are captured by layer *P3* of the feature pyramid network. This layer can detect objects of size 32^2 which is not sufficient. My thought was to go step by step with adding new higher resolution layers starting with *P2* and then *P1*. When adding the *P2* layer I did it the same as in 3.4.1.

The *P2* layer is created directly from the residual block (stage) from *Resnet* backbone named *C2* and summed elementwise with P3 using top-down and

```
def create_pyramid_features(C2, C3, fs=256):
""" Creates the FPN layers on top of the backbone features.
Args
C2           : Feature stage C2 from the backbone.
C3           : Feature stage C3 from the backbone.
fs : The feature size to use for the resulting feature levels.

Returns
A list of feature levels [P2, P3].
"""

P3 = Conv2D(fs, kernel_size=1, strides=1, padding='same', name='C3_redu')(C3)
P3_upsampled = layers.UpsampleLike(name='P3_upsampled')([P3, C2])
P3 = Conv2D(fs, kernel_size=3, strides=1, padding='same', name='P3')(P3)

P2 = Conv2D(fs, kernel_size=1, strides=1, padding='same', name='C2_redu')(C2)
P2 = Add(name='P2_merged')([P3_upsampled, P2])
P2 = Conv2D(fs, kernel_size=3, strides=1, padding='same', name='P2')(P2)
return [P2, P3]
```

Listing 2: Function to create modified feature pyramid network. New higher resolution layer P2 is added and layers P4, P5, P6, P7 are discarded.

lateral connections. You can see the code in listing 2. After adding this layer, I was able to detect objects of are 16^2 pixels. This is still not enough for my intended usage. But to make sure that the modification is working properly I also created new dataset similarly as in the previous section 4.3.1. The difference was just in the final resolution of the new images. I did not partition the original image into twenty-five new but in only into four new with a resolution of 625 x 500 pixels. This way the annotation were upscaled to become about 16^2 pixels big. When I started the training with this new dataset, I immediately saw that it takes much longer time. It is because when adding the higher resolution layer *P2* we need to compute an exponentially bigger number of anchors than in layer *P3* to capture this larger amount of smaller objects.

It turned out that going this way is not the right direction to go. Even with only just *P2* layer added, I encountered problems with memory during the training. For example when the network was training from images with a bigger number of objects, like in figure 4.12, my GPU was not able to hold all the needed information in memory, and the program crashed. Before I decided not to continue with this approach, I tried to artificially regulate the number of anchors by removing the anchors of higher layers *P4, P5, P6, P7*.

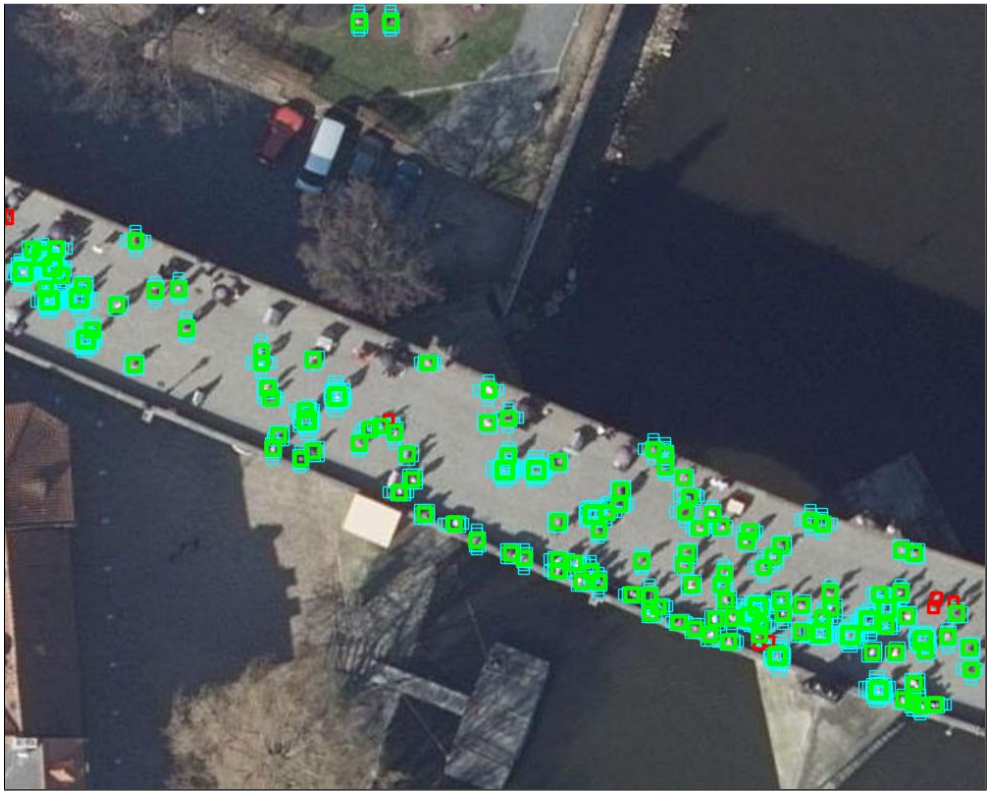


Figure 4.12: Same as in figure 4.11 but now for the *middle* image I have used in 4.3.2. You may notice that the difference against anchors from the *small* image is that each annotation has fewer anchors assigned and also you can see red squares corresponding to the annotations without any assigned anchor. This happens when the size of the annotation is smaller.

I could remove them as they were unnecessary because detection of objects with bigger size was not needed. But again the sum of anchors produced by these layers was much smaller than the sum of anchors from layer *P2*, and it helped just a little bit. Another regulation I have made was that I removed the scaling and ration changing when generating each anchor. This way I was using only the anchors of original size with 1:1 ratio (square). Both these modifications were enough to do the training without memory problems.

After struggling with only the addition of *P2*, I decided not to expand the feature pyramid network even further by adding the layer *P1*. I can certainly say that it would not be possible to train network this big on my home computer. I could have tried to run the training on some cluster with more powerful graphics cards. But another reason I decided not to go this way was that the intermediate results were significantly worse than results I had when training the unmodified network with just the small images. You can see more detailed results as well as graphs in chapter 5.

4.4 Application of the prediction model

Another part of the realization was to choose the best prediction model and use it for detecting people on new images. I decided to use part of the remaining unannotated images from Geoportal. I wanted to detect people on these images and visualize them similarly as the original annotations 4.9.

When you use the prediction model, you get the coordinates of each of the object on the given image on which you make the inference. These coordinates are relative to this image, but you need to transform them to real-world coordinates. For each of the original image, there was also georeference file which contains the coordinates for a corresponding image. This file is used for referencing the location of aerial view maps. The extension of these files is JGW, and they specify the scale of the image per pixel as well as the (X, Y) coordinate of the upper left pixel and using this information, the actual geographical area of the related image can be reconstructed. Let me describe the process of the coordinates transformation.

First, I split the new images into small parts to match the train-set images. Then I used the prediction model to infer all people bounding boxes predictions and save them into one file. I saved all predictions with the probability confidence higher or equal to 0.55, which I set experimentally. The structure of each of these predictions was following. It contained the name of the file on which it was predicted, a probability of the prediction and coordinates of the bounding box on the image. This is where the proposed naming convention comes in handy. I could use indexes from the filename to determine the location of this small image due to the original image for which I have the JGW file. This way I recalculate coordinates of each prediction to real-world coordinates. This recalculation required a transformation of the coordinates from S-JTSK planar coordinate system to WGS84 planar coordinate system, and you can find the script in appended files. Then I created *GeoJSON* file containing all the coordinates saved as *Features* of type *Point*. I have also included the value of probability in properties field. This way if anybody would like to filter some predictions over some specific threshold he can. Next, I used the Mapbox Studio which is an application for managing geospatial data and designing custom map styles to create map shown in figure 4.13. You can find images of maps with different zoom level in the appendix B.

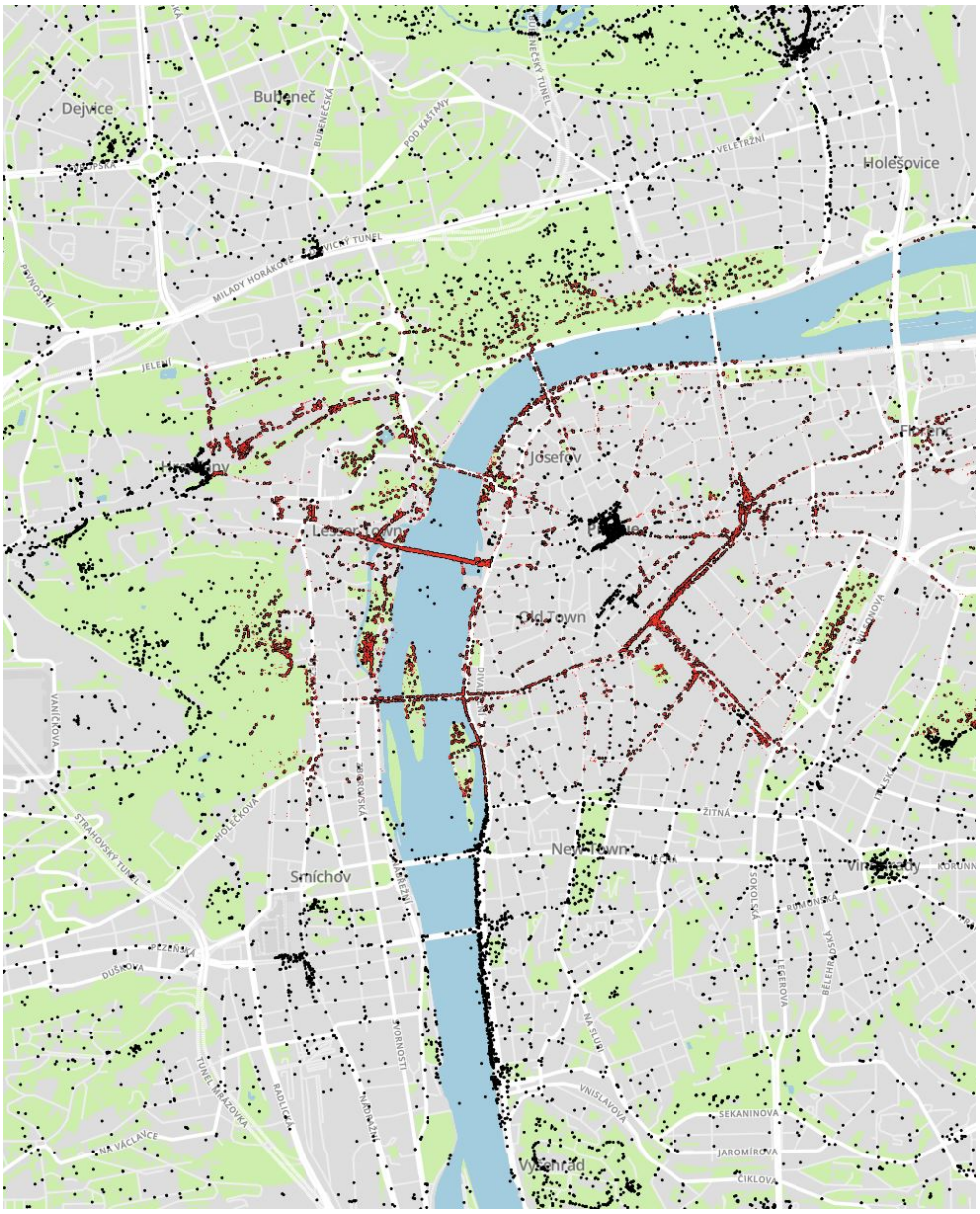


Figure 4.13: On this figure you can see people detected by the best model visualized on a map. Also, note the original annotations visualized in red color. This map covers only part of the area I have made the detections. For maps with another zoom level look at images in appendix B.

Chapter 5

Experiments

In this chapter, I will talk about experiments I have tried when training the prediction model. I experimented with various meta-parameters of the algorithm to find out which setting is producing the best results. But before I dive into experiments, I should explain how we can measure the performance of some model.

5.1 Evaluation

When we perform the task of object detection and localization, we need some technique to measure how well our algorithm is performing. We need to evaluate the accuracy of our prediction model after it is trained. We are looking for some one-numbered metric from which we can see at first sight how good is the prediction model. When performing some other computer vision tasks as classification, it is easier. We want to predict class label for given image which we can then evaluate if it is either correct or incorrect. But we cannot apply this type of binary evaluation to the object detection problem because usually, the predicted bounding boxes are a little bit of the ground truth bounding boxes, so they do not match exactly, and that would mean that they are incorrect. It is almost certain that predicted bounding boxes will never be exact because they are produced (in case of this work) by feature pyramid network which generates anchors (potential bounding boxes) in a deterministic manner, but the real world position of an object is totally random. So that is why we instead use a metric called *intersection over union*.

At first, let me explain what is it *intersection over union*. Then I will show you, how is this used to compute *mean average precision* which is broadly used metrics for object detection prediction models evaluation.

5.1.1 Intersection over union

Intersection over union is an evaluation metric used to measure the accuracy of an object detection algorithm. It is nothing complicated. To compute intersection over the union, we need to know the original ground truth bounding box and the predicted bounding box. For example, see the figure 5.1 where the ground-truth bounding box is shown with green and the predicted bounding box with red. The green square was created by some person while algorithm generated the red one. As you have already notice purpose of the bounding boxes is to store the exact location of some object. Also, it does not matter from which algorithm the predicted bounding box comes, but in our case, we get these bounding boxes from regression head that stands on the top of the whole deep convolutional network. The formula how to compute intersection over union is shown in figure 5.2. As you can see, it is a simple ratio. In the numerator, we compute the area of overlap between the ground-truth bounding box and the predicted bounding box. In the denominator is the area of the union of both bounding boxes.

Figure 5.1: Ground-truth (green) and predicted (red) bounding box.

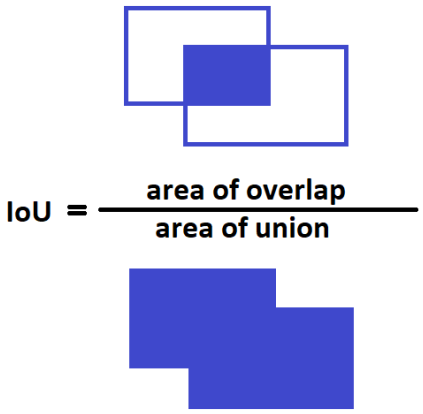
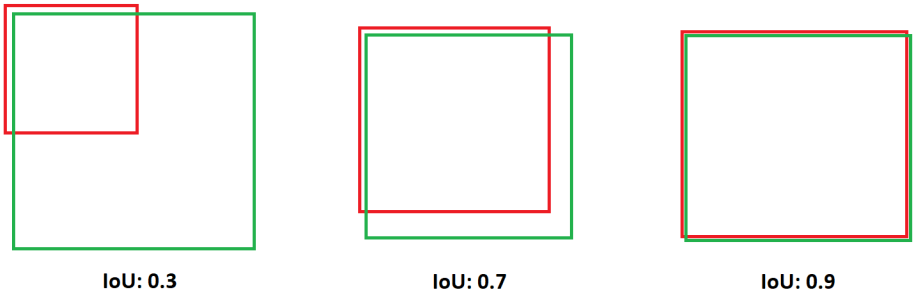


Figure 5.2: Visualization of the intersection over union.

Usually, if an intersection over union score is greater than 0.5 it is generally considered as a *good* prediction.

5.1.2 Mean average precision

Mean average precision is single number metric and it is used in most popular object detection challenges like PASCAL VOC, ImageNet, and COCO. This metric is used because in ordinary datasets there are many classes and their distribution is non-uniform. This usually happens with skewed classes. That is when the occurrence of one class is very rare in the entire dataset. So a simple accuracy-based metric would introduce biases. The other thing is that



(a) Bad IoU. (b) Good IoU. (c) Almost perfect IoU.

Figure 5.3: Examples of some intersection over union scores for various bounding boxes.

we have to somehow take into account the risk of misclassification. For these reasons the *average precision* was introduced. Before I start to talk about average precision I will briefly explain familiar terms *recall* and *precision*. For better understanding let me use an example from Andrew Ng and his Coursera machine learning class. We are predicting if patients have cancer or not.

Predicted	**Actual**	**Name**
1	1	True positive
0	0	True negative
0	1	False negative
1	0	False positive

Precision: From all patient that we predicted that they have cancer, what fraction actually has cancer?

$$\frac{True\ Positives}{Total\ number\ of\ predicted\ positives} = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

In object detection context, precision measures the *false positive rate* or the ratio of true object detections to the total number of objects that the classifier predicted. If we have a precision score of close to 1.0 then there is a high likelihood that whatever the classifier predicts as a positive detection is in fact a correct prediction.

Recall: From all patient that actually have cancer, what fraction did we correctly detect as having cancer?

$$\frac{True\ Positives}{Total\ number\ of\ actual\ positives} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Again in object detection context, recall measures the *false negative rate* or the ratio of true object detections to the total number of objects in the data

set. If we have a recall score close to 1.0 then almost all objects that are in your dataset will be positively detected by the model.

It is very important to note that there is an inverse relationship between precision and recall. Also, these metrics are dependent on the model score threshold that we set (as well as of course, the quality of the model). When computing this normal precision and recall, we use just one value of the threshold. The greater the threshold, the greater the precision and the lower the recall and vice versa the lower the threshold, the greater the recall and the lower the precision. See figure 5.4 to see the relation curve between recall and precision.

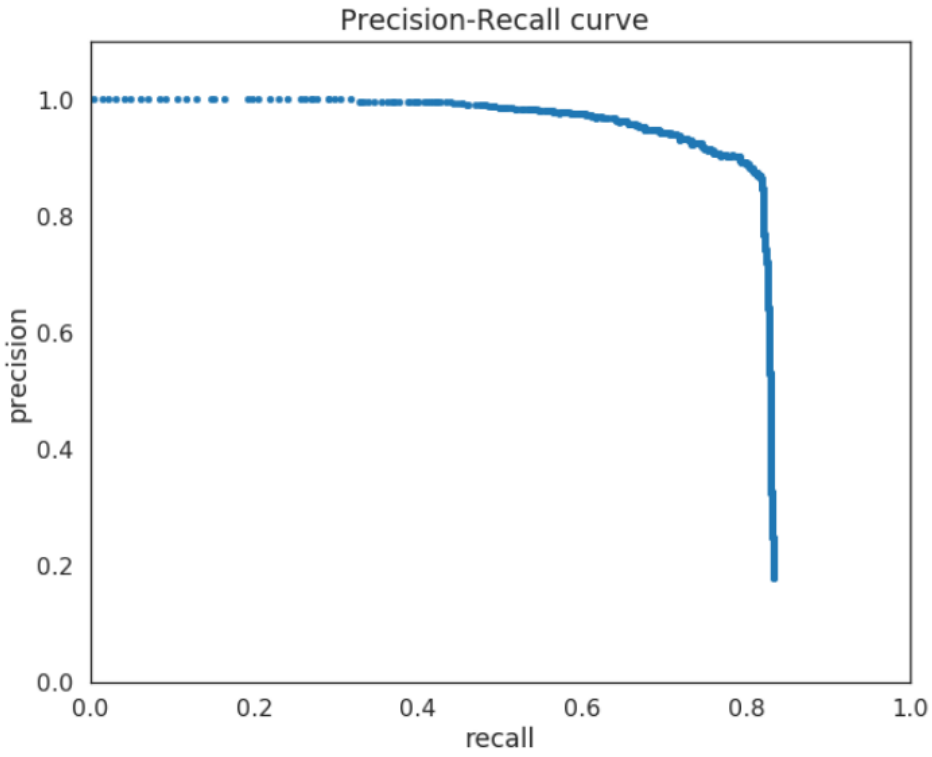


Figure 5.4: Example of the relationship between recall and precision. Each point of this curve corresponds to some value of the threshold. And as we can see with the increasing recall value (lowering the threshold) the precision value is decreasing and vice versa.

Average precision differs in the sense that we use not one value of the, but we compute it for all possible thresholds. To compute the average precision score, we take the average value of the precision across all recall values. This then becomes the single value summarizing the shape of the precision-recall curve. In case of object detection, we sort all the predictions produced by the prediction model by their confidence scores from the biggest to the lowest.

This way we obtain all levels of the mentioned threshold. Also for each of this threshold value we store the amount of *false positive* or *true positive* predictions. Then we can easily compute all values of recall and precision which we then use for computing average precision. See equation 5.1.

$$Average\ Precision = \frac{1}{number\ of\ threshold\ levels} \sum_{Recall_i} Precision(Recall_i) \tag{5.1}$$

You may also ask how we determine if the prediction (more exactly predicted bounding box) is considered as correct (*true positive*) or incorrect (*false positive*). This is where the previously explained intersection over union comes into play. We can set some value of this IoU threshold to determine between correct and incorrect predictions. IoU thresholds vary for each competition, for example in the COCO challenge ten different IoU thresholds are considered, from 0.5 to 0.95 in steps of 0.05. In my experiments, I have usually used the value of 0.5. Now that we have defined average precision and seen how the IoU threshold affects it, the mean average precision (mAP) score is calculated by taking the mean average precision over all classes and/or over all IoU thresholds, depending on the competition. I did not have any complicated averaging because I have usually used one value of IoU and I have worked only with one class (person).

5.2 Experiments

Now that we know how to measure the performance of the algorithm, we can start experimenting. When training the model, I split my dataset into three parts. A did not split the dataset by images but by the number of annotations. This way it can't happen that some part has only images with few annotations while some other have images with dense annotations. The train set contains 60% of all annotations, validation, and test set both 20%.

First, let me note that the training of deep neural networks takes quite a long time. For instance, the training on multiple graphics cards on ImageNet dataset which is one of the biggest available image datasets may take even several weeks. This extreme is fortunately not my case because my dataset is not that huge. But still, the training takes some not negligible time because it is done only on one GPU on my personal computer (see the exact parameters of the computer I used in table 5.1). The time of the training differed with the configuration of the network and also with the modifications I did to the images from the dataset. I will describe each of these configurations, and I will present average precisions each of it achieved. Before I start talking about the experiments, I want to say that to train the model for one epoch with Faster

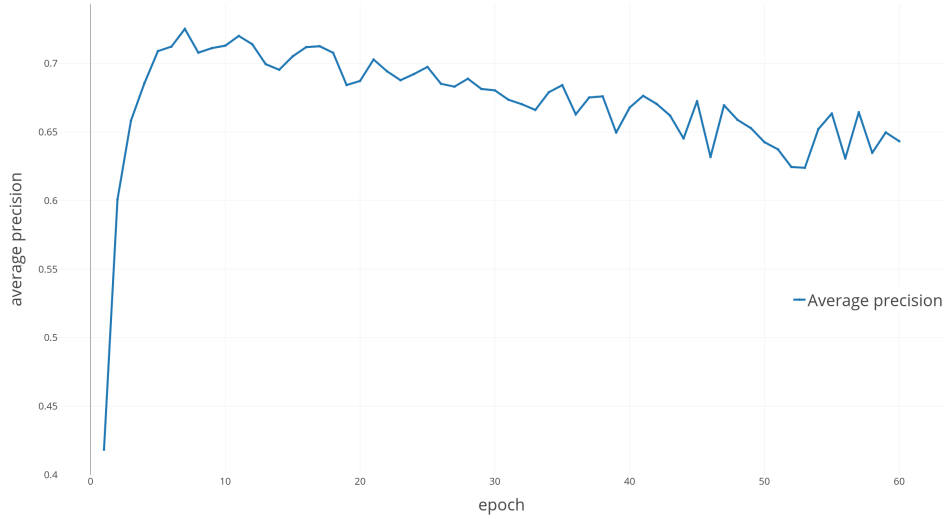
R-CNN it took over one hour which is, considering that my plan was to do at least 50 epochs with every network configuration, quite long time. Just to remind what exactly epoch is. The epoch in deep neural network terminology is when you feed every sample from the dataset to the network during the training. So when I trained the model for thirty epochs, it means the model saw all images from the dataset also thirty times. Fortunately, the RetinaNet reduced this time to bearable twenty minutes. It allowed me to train the models with about thirty epochs through the night. But as it turned out thirty epochs of training weren't sufficient. When I used data augmentation (about which you can read more in the next section), it needs to train the network for about 80 epochs to achieve the best average precision. On the opposite side when I trained without augmenting the data, the model started to overfit just about after seven epochs (see figure 5.5). I attribute that to the fact that even though the dataset contains about 16k of annotated objects, it is still quite small in the scale of deep neural nets standards.

In general, the process of training was following. I started the training with the mentioned 30 epochs. Then I visualized the loss on train set after each epoch together with the validation loss. On the figure B.1 in the appendix you can see how it usually looked like. Another thing I was checking was the average precision after each epoch. I visualized it as well. Look at the figure 5.8.

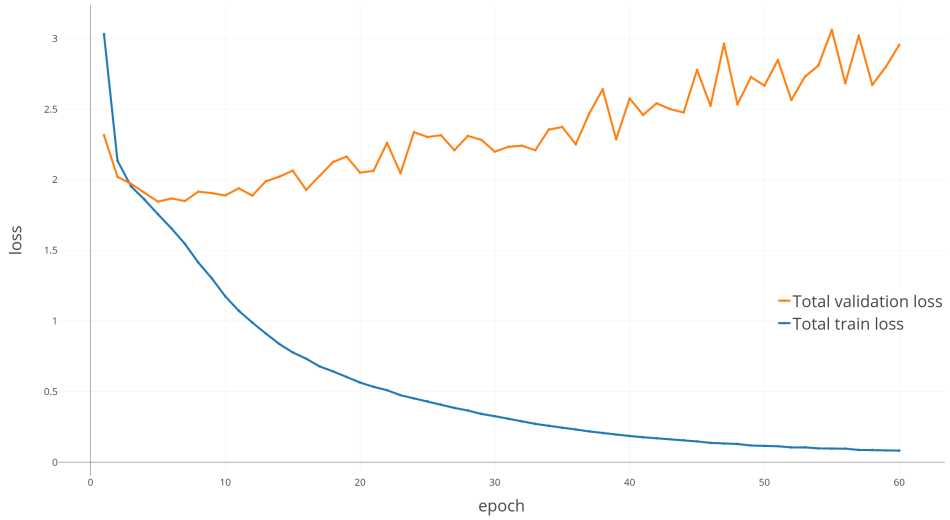
As you can notice the validation loss is decreasing approximately till the 7th epoch and then it starts to increase. Parallel the average precision on the validation set is peaking also around the same epoch.

As described in chapter 4.3 I used two sizes of training images for training. Although I proposed RetinaNet modifications to better fit the dataset in 4.3.2, it turned out to produce much worse results (almost 20% of the average precision) than the unmodified network trained with smaller images. That's why I trained mostly with these small images.

For most of my experiments, I initialized the model with pretrained ImageNet weights. Then during the training, I tuned these parameters for my own dataset without freezing any layer of the network. It is common to freeze all weights of the convolutional part of the network and train only the fully connected layers. This technique is called *transfer learning* and allow us to use the already trained encoder and fit it to our desired classes. I also tried to experiment with this by freezing different layers of the network, but the result was much worse than when all parameters were trained. This is probably due to big differences between objects from ImageNet and my dataset. In ImageNet classical photos are containing 1000 classes of various objects like vehicles, dogs, etc. These objects are very different than people on aerial images. As optimizer I chose the Adam [22] optimizer with initial learning rate if 1×10^{-5}. The learning rate then dynamically decreases during the training.



Average precision.



Losses on train and validation sets.

Figure 5.5: On this figure you can see information about the training of a network with ResNet50 backbone without data augmentation over 50 epochs. As you can notice the validation loss starts to grow after about 7th epoch. At the same time, the average precision stars to decrease. The 7th epoch is when the model started to overfit on the train data.

Part	Description
CPU	3.3 GHz Intel(R) Core(TM) i5-4590
GPU	NVIDIA GeForce GTX 970 4GB
RAM	8 GB
OS	Windows 10 64 bit

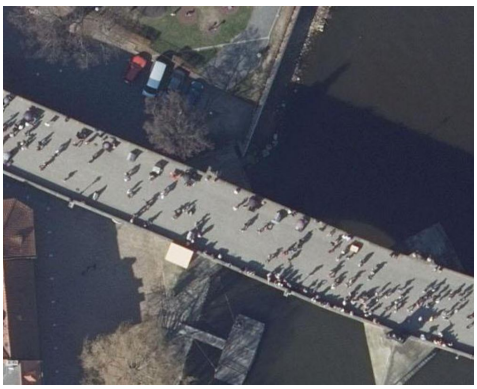
Table 5.1: The parameters of the computer used for the training.

Data augumentation

When training deep neural networks, it is common practice to do the data augmentation. It is an easy way of creating new data without the need of annotation new images. By various transformation of the original images, you can create new training samples and extend the existing dataset. I experimented with data, and you can read about it in this section. I have used these types of image modification listed below. You can see all of them as well as the original not augmented image on the figure 5.6.

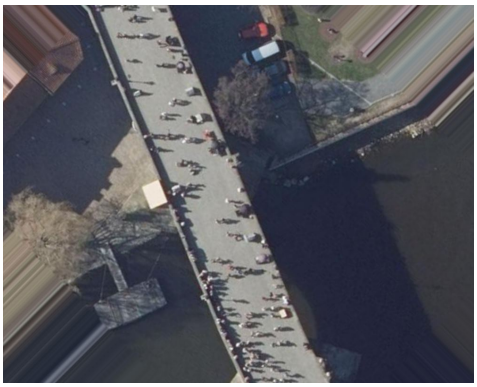
- **Flipping**: I set the algorithm to flip each image during the training in both axis with a 50% chance.
- **Rotation**: I rotated the original images in random range from -90° to 90°. This range covers all possible image positions when used simultaneously with image flipping.
- **Translation**: Another augmentation is translation. This modification shifted each image from zero to 10% of the image size to the left or right or top or down.
- **Scaling**: All the previous modification can be understood as creating a new source of data because the structure is retained. With the scaling, it is a little bit different. Because we work on a dataset that has some set precision we cannot just freely change the size (resolution). I decided only to set scaling augmentation to a tiny level. From decreasing the size by 5% to increasing it by 5% of the original image.
- **Shearing**: Shearing is the strongest augmentation type because it changes the overall ratio of the image by beveling it by a certain amount of degrees. I did not use this augmentation type. I just state that it exists and it can be helpful when detecting a different kind of objects.

As mentioned in the previous section when I trained without the data augmentation the model started to overfit early after few epochs. Even though it achieved high average precision, I decided not to use models trained like this because it is only good to make a prediction on the dataset it was trained which is not wanted. I want to have a model that generalizes to the real world



(a) Original image.

(b) Image flipped over both axis.



(c) Image rotated by 45°.

(d) Image shifted down and right.

(e) Image with increased size.

(f) Sheared image. I did not use this one.

Figure 5.6: On this figure you can see original images and five types of augumentation which can be applied during the training to increase the amount of data.

(different datasets). On the other hand, when I trained the network with augmented images, the validation loss oscillated little bit up and down, but what is important, in general, it decreased. Same for the average precision achieved on the validation dataset. It also wobbled, but it converged. I arrogate these oscillations to the structure of the validation set images being different than the structure of the augmented and modified images which were used during the training. Every image from the original dataset was photographed at the same hour of the day, so, for instance, the shadows are roughly directed in the same way for every person. This causes that the network is learning this as a feature and when we rotate and otherwise modify the images during training it afterward doesn't match with the unmodified images from the validation set. When we think about this, it is not a bad thing. When we would like to make the predictions on some other images also with little bit different structure than is our train dataset we should try to set up the augmentation to match this other dataset. This hypothesis confirmed when I tried to test the model in images from a new dataset. This dataset is from the same source as the dataset I used for training but photographed approximately one year later. Even though the images in this dataset are almost identical in the sense of structure (people changed), there are few minor differences. For example, the shadows have a little sharper angle, and they are shorter. See the difference on figure 5.7. Notice the different shadows cast by the lamps in the middle of the picture. This change caused that with the model trained without augmentation I wasn't able to detect almost any person, and on the other hand with the model trained on augmented data it worked perfectly. This demonstrates the power of data augmentation, although we have annotated images only from one source we can tune the training and get model suitable for predictions on a dataset from different sources. This is why I mainly focused on training only with data augmentation.

Backbone selection

Another thing I could change was the network's backbone architecture. For the majority of my experiments, I used ResNet [35]. There are three variants: ResNet50, Resnet101, and Resnet152. The number marks the number of layers for given architecture. A tried to train the first two of them, and the results didn't differ much (see figure 5.8). The only thing that was different was the amount of time it took to train each of them. Resnet50 needed about 18 minutes per epoch, and the Resnet101 trained one epoch over about 25 minutes. When I tried to train Resnet152, I faced a problem with memory, but I expect that the benefit of using it wouldn't also be not that significant. Another backbone was MobileNet [41] which is designed to work on mobile devices, so its main intention is to be fast and effective. I didn't do many experiments with this backbone because it produced worse results than ResNet. Also, I should note that the training when using MobileNet128 as backbone

(a) Example image from old dataset. (b) Example image from new dataset.

Figure 5.7: Example of little differences between old annotated dataset and updated dataset from Geoportal. Notice different angles and lenghts of the shadows.

took almost the same time as for Resnet50. The MobileNet128 needed only 2 minutes per epoch less. On the other hand, the inference time is where MobileNet shines. It is suitable for real-time prediction, but it is nothing I need. The last backbone I tried was DenseNet [42]. But with this backbone, I wasn't able to even start the training due to its high memory requirements. This is caused because DenseNet is much deeper network than ResNet or MobileNet and it has a huge amount of parameters.

One class vs. two classes

Another thing I had to decide was the decision what annotations to use for the training. When I was annotating the dataset, I introduced six pseudo-classes for these annotations with the idea that I can filter out some annotation - most likely the worst ones of people in shadow or somehow obscured or poorly visible. I have done some experiments where I trained the model using annotations of different classes. For instance, I used only the first three classes (the best ones) and discarded the classes of worst annotations. My idea was that the model could work better if it is trained only with these good classes. But this assumption was wrong because when we want to use the model for predictions, our goal is not to find only well visible people but we want to detect as many people as it is possible. After I realized this, I trained the model for two classes, the first one same as before (only good annotations) and the second class consisting of annotations of badly visible people. But this also didn't work that well. And again I realized that this division only introduces artificial complication because the model has not only to determine if something is a person but also assign it one of these classes. It makes no sense as we don't care much about if the people are clearly visible or not but

we simply want to detect all of them. After all these missteps I ended up using all annotations as one and only class. And I think that this way the model is most general and is suitable for a real-world application.

Focal loss parameters

I also tried to change the meta-parameter γ of the focal loss, but the initial experiments showed no average precision improvements, so I decided not to continue with it.

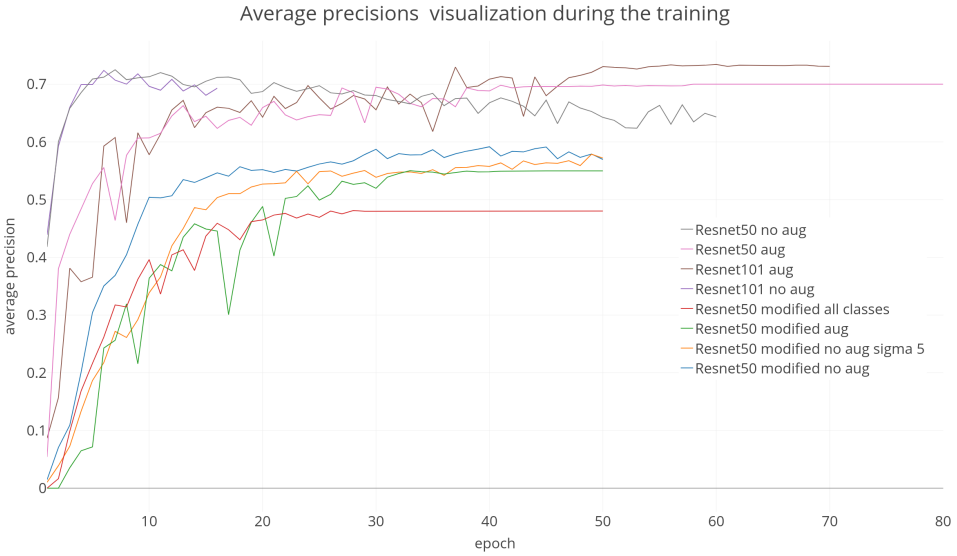


Figure 5.8: The visualization of the average precision during the training. You can see some of the different settings I haved used during the training.

Backbone	Augmentation	Classes	γ	Frozen Layers	Initial Weights	Dropout
Resnet50	flipping	cleanest	2	all	ImageNet	yes
Resnet101	translation	clean	3	some	none	no
Mobilenet128	rotation	not clean	4	none		
Resnet50 mod	scaling	uncertain	5			
		bench or sitting				

Table 5.2: Summary of the all the parameters I could have experimented with. Any tried many of different combinations of the setting, and you can read more about the most important in the text. I did not describe every possible configuration because I usually stopped training when the specific configuration yielded bad results. (The best configuration: Resnet50 with heavy augmentation of all types, using all classes, with default γ equal to 2, no frozen layers, ImageNet as initial weights and no dropout)

5.3 Results

In table 5.3 you can find the resultant average precisions measured on the test dataset. Notice that the best average precision has the Resnet50 without data augmentation. Even though it achieved the highest score on test set I think that it is due the model is overfitted. Unfortunately, the only dataset which I could use for testing is from the same source of data as the training set. I believe that if I would have used test set consisting for example of images from the images taken one year later, both of the models trained without data augmentation would have average precision several times lower. Another assumption is that opposite models with data augmentation would remain its average precision also on this different test set.

Figure 5.9: On this figure you can see the image transformation when I wanted to do the prediction on the original high resolution images. It is the same as when I prepared data for the training.

Model	Average precision
Resnet50 (no augmentation)	0.6953
Resnet50 modified (no augmentation)	0.5012
Resnet101 (no augmentation)	0.6640
Resnet50 (with augmentation)	0.6571
Resnet50 modified (with augmentation)	0.4753
Resnet101 (with augmentation)	0.6881

Table 5.3: Average precisions of the best models from different network settings. All these configurations had default γ, were trained on annotations of all classes without any frozen layers with ImageNet as initial weights and utilizing no dropout layers. I do not list average precisions of other not so successful configurations.

Figure 5.10: On images where people are nicely visible the model has no problem detecting them. For example when the background is coherent or people are't that close to each other.

(a) (b)

Figure 5.11: In pictures where there is a large number of people close to each other the model struggles to detect all of them or sometimes detects one person with multiple bounding boxes.

(a) (b)

Figure 5.12: Another difficult images for the model are the ones where the people aer in the shadow. To my surprise the model handles this difficulty quite well. But, of course, it doesn't detect everything.

Chapter 6

Conclusion

In this work, I deeply studied the most recent techniques and algorithms for object detection. Also, I successfully applied these state-of-the-art algorithms to detect people from aerial images. I introduced ways how to achieve that they become suitable for working with high-resolution aerial images. With my approach, it is possible to detect people who appear as a little dots on the original high-resolution images. The model is also not lagging behind when detecting people close to each other and it is also able to partly detect people in shadow as well as if they are partially obscured. I have done many experiments during which I trained the network with different kind of data. I had to process thousands of images when preparing them for training. I also experimented with various settings of the network itself. My best prediction model was based on the Resnet101 backbone and achieved the average precision of almost 0.7 which I found out as a good result. To achieve this, I had to create my own dataset which I annotated using my annotation utility. The creation of the dataset is another big part described in this thesis. This dataset will be available for further experiments. Last but not least, I used the final model to detect people on new image dataset and saved all the detection in a format suitable for further work. I also used this data to create heatmap symbolizing the people density on the area of the Prague city center and its surroundings.

6.1 Future work

As a future work, I propose some of the possible improvements for getting more precise prediction model. Usually, in deep learning, it is hard to say what exact change or improvement of the pipeline would cause that we get better results, so I will describe possible things to try not ordered by priority.

Let's start with the training data. The creation of the dataset itself was quite extensive, and I didn't put that much effort into making it perfect because I needed to start working on the prediction algorithm. I think that some kind of revision of the dataset would be beneficial. For instance, I be-

lieve that I forgot to annotate a lot of people or reversely that I annotated some objects that are not people. This would need to be revised and corrected as well as the distribution of the annotations into introduced classes could be reworked. This would require to go through all annotations and simply reassign the class we want this object to be without any changes to the bounding box. This relates to another point of future work namely creation of two new pseudo-classes: *person* and *person in the shadow*. With these classes, it would be possible to focus on different settings when detecting well or poorly visible people. Another improvement related to the dataset is to get more data. Maybe annotate some other images from the current data source or to annotate new images from another source. This way we can get a dataset that is more general because it would contain images of a different kind. Also, one thing that came to my mind was to use videos shot from a stationary drone. With these videos, it is then quite simple to detect moving objects and assign them proper classes (car, person, etc.). Afterward, we could split the videos into images and lower the resolution so we can easily obtain brand new dataset. This is just an idea which may not work, but I think it would be nice to try it. I found annotated videos that were used in [43]. Although I managed to achieve very good precision we could maybe improve it when using of some new object detection algorithm. For instance, we could use YOLOv3 [19] or any other newly published algorithm. These new algorithms (or improvements to the current one) are published with high pace and usually improve both the precision as well as the speed. Next thing that is possible to do is to combine my prediction model for people detection and enhance it with a model that detect for instance buildings. This would improve the accuracy of the result by simply verifying if every detected person is not in the area where some building is (detected by the other model). This way we could eliminate false predictions of antennas, chimneys or some other objects that similarly cast the shadow as people do and which cause troubles for the model. This approach is also applicable for example for water.

Bibliography

[1] Ren, S.; He, K.; et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *ArXiv e-prints*, June 2015, 1506.01497.

[2] Lin, T.-Y.; Goyal, P.; et al. Focal Loss for Dense Object Detection. *ArXiv e-prints*, Aug. 2017, 1708.02002.

[3] Krizhevsky, A.; Sutskever, I.; et al. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, edited by F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, pp. 1097–1105. Available from: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[4] Stanford. CS231n Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/2016/syllabus, [Online; accessed 9-March-2018].

[5] Giordano, S.; Bris, A. L.; et al. Fully automatic analysis of archival aerial images current status and challenges. In *2017 Joint Urban Remote Sensing Event (JURSE)*, March 2017, pp. 1–4, doi:10.1109/JURSE.2017.7924620.

[6] Jean, N.; Burke, M.; et al. Combining satellite imagery and machine learning to predict poverty. *Science*, volume 353, no. 6301, 2016: pp. 790–794, ISSN 0036-8075, doi:10.1126/science.aaf7894, http://science.sciencemag.org/content/353/6301/790.full.pdf. Available from: http://science.sciencemag.org/content/353/6301/790

[7] Deng, Z.; Sun, H.; et al. Toward Fast and Accurate Vehicle Detection in Aerial Images Using Coupled Region-Based Convolutional Neural Networks. *IEEE Journal of Selected Topics in Applied Earth Observations*

and Remote Sensing, volume 10, no. 8, Aug 2017: pp. 3652–3664, ISSN 1939-1404, doi:10.1109/JSTARS.2017.2694890.

[8] Cao, L.; Wang, C.; et al. Vehicle detection from highway satellite images via transfer learning. *Information Sciences*, volume 366, 2016: pp. 177 – 187, ISSN 0020-0255, doi:https://doi.org/10.1016/j.ins.2016.01.004. Available from: http://www.sciencedirect.com/science/article/pii/S0020025516000062

[9] Audebert, N.; Le Saux, B.; et al. Segment-before-Detect: Vehicle Detection and Classification through Semantic Segmentation of Aerial Images. *Remote Sensing*, volume 9, no. 4, 2017, ISSN 2072-4292, doi:10.3390/rs9040368. Available from: http://www.mdpi.com/2072-4292/9/4/368

[10] Maggiori, E.; Tarabalka, Y.; et al. Convolutional Neural Networks for Large-Scale Remote-Sensing Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, volume 55, no. 2, Feb 2017: pp. 645–657, ISSN 0196-2892, doi:10.1109/TGRS.2016.2612821.

[11] Meynberg, O.; Cui, S.; et al. Detection of High-Density Crowds in Aerial Images Using Texture Classification. *Remote Sensing*, volume 8, no. 6, 2016, ISSN 2072-4292, doi:10.3390/rs8060470. Available from: http://www.mdpi.com/2072-4292/8/6/470

[12] Oliveira, D. C. D.; Wehrmeister, M. A. Towards Real-Time People Recognition on Aerial Imagery Using Convolutional Neural Networks. In *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*, May 2016, pp. 27–34, doi:10.1109/ISORC.2016.14.

[13] Blondel, P.; Potelle, A.; et al. Fast and viewpoint robust human detection for SAR operations. In *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, Oct 2014, ISSN 2374-3247, pp. 1–6, doi:10.1109/SSRR.2014.7017675.

[14] Tijtgat, N.; Ranst, W. V.; et al. Embedded Real-Time Object Detection for a UAV Warning System. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, Oct 2017, pp. 2110–2118, doi:10.1109/ICCVW.2017.247.

[15] contributors, W. Orthophoto — Wikipedia, The Free Encyclopedia. 2018, [Online; accessed 9-March-2018]. Available from: https://en.wikipedia.org/w/index.php?title=Orthophoto&oldid=829402448

[16] He, K.; Gkioxari, G.; et al. Mask R-CNN. *ArXiv e-prints*, Mar. 2017, 1703.06870.

[17] Redmon, J.; Divvala, S.; et al. You Only Look Once: Unified, Real-Time Object Detection. *ArXiv e-prints*, June 2015, 1506.02640.

[18] Redmon, J.; Farhadi, A. YOLO9000: Better, Faster, Stronger. *ArXiv e-prints*, Dec. 2016, 1612.08242.

[19] Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *ArXiv e-prints*, Apr. 2018, 1804.02767.

[20] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[21] He, K.; Zhang, X.; et al. Identity Mappings in Deep Residual Networks. *ArXiv e-prints*, Mar. 2016, 1603.05027.

[22] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, Dec. 2014, 1412.6980.

[23] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*, Feb. 2015, 1502.03167.

[24] Uijlings, J. R. R.; van de Sande, K. E. A.; et al. Selective Search for Object Recognition. *International Journal of Computer Vision*, volume 104, no. 2, 2013: pp. 154–171. Available from: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013

[25] Zitnick, C. L.; Dollár, P. Edge Boxes: Locating Object Proposals from Edges. In *Computer Vision – ECCV 2014*, edited by D. Fleet; T. Pajdla; B. Schiele; T. Tuytelaars, Cham: Springer International Publishing, 2014, ISBN 978-3-319-10602-1, pp. 391–405.

[26] Pinheiro, P. O.; Collobert, R.; et al. Learning to Segment Object Candidates. *ArXiv e-prints*, June 2015, 1506.06204.

[27] Pinheiro, P. O.; Lin, T.-Y.; et al. Learning to Refine Object Segments. *ArXiv e-prints*, Mar. 2016, 1603.08695.

[28] Girshick, R.; Donahue, J.; et al. Rich feature hierarchies for accurate object detection and semantic segmentation. *ArXiv e-prints*, Nov. 2013, 1311.2524.

[29] Girshick, R. Fast R-CNN. *ArXiv e-prints*, Apr. 2015, 1504.08083.

[30] Lin, T.-Y.; Dollár, P.; et al. Feature Pyramid Networks for Object Detection. *ArXiv e-prints*, Dec. 2016, 1612.03144.

[31] Anderson, C. H.; Bergen, J. R.; et al. Pyramid Methods in Image Processing. 1984.

[32] Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2372-2, pp. 886–893, doi:10.1109/CVPR.2005.177. Available from: http://dx.doi.org/10.1109/CVPR.2005.177

[33] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, volume 60, no. 2, Nov. 2004: pp. 91–110, ISSN 0920-5691, doi:10.1023/B:VISI.0000029664.99615.94. Available from: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[34] Lin, T.-Y.; Maire, M.; et al. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, edited by D. Fleet; T. Pajdla; B. Schiele; T. Tuytelaars, Cham: Springer International Publishing, 2014, ISBN 978-3-319-10602-1, pp. 740–755.

[35] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, Dec. 2015, 1512.03385.

[36] Fizyr. Keras RetinaNet. https://github.com/fizyr/keras-retinanet, 2018.

[37] Yhenon. Keras Faster R-CNN. https://github.com/yhenon/keras-frcnn, 2017.

[38] Seznam.cz. Mapy.cz. http://www.mapy.cz, [Online; accessed 9-March-2018].

[39] Geoportal. Orthophotomap of Prague. http://www.geoportalpraha.cz/cs/opendata/A1324401-980B-44C0-80D6-5353AFEC437E, [Online; accessed 9-March-2018].

[40] Mapbox.com. Mapbox.com. https://www.mapbox.com/, [Online; accessed 27-April-2018].

[41] Howard, A. G.; Zhu, M.; et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv e-prints*, Apr. 2017, 1704.04861.

[42] Huang, G.; Liu, Z.; et al. Densely Connected Convolutional Networks. *ArXiv e-prints*, Aug. 2016, 1608.06993.

[43] Robicquet, A.; Sadeghian, A.; et al. Learning Social Etiquette: Human Trajectory Prediction In Crowded Scenes. 2016, european Conference on Computer Vision (ECCV). http://cvgl.stanford.edu/projects/uav_data/.

Appendix A

Contents of CD

- `readme.txt` ..... the file with CD contents description
- `data` ..... the data files directory
  - `annotations` ... the directory of annotations from the annotation tool
  - `export` ..... the directory of annotations used for training
  - `images` ..... the directory of all images
    - `original` ..... the directory with all images of original resolution
    - `big` ..... images used during the annotation
    - `middle` ..... images used to train modified network
    - `small` ..... images of the smalles resolution
  - `jgws` ..... the directory with files about the geospatial information
  - `mapbox` ..... the directory with files containing predictions used for mapbox map visualization
  - `models` ..... the directory with the models
    - `aug` ..... trained models when using data augmentation
    - `noaug` ..... trained models when using data without augmentation
    - `inference` ..... models used for inference
  - `stats` .. the directory with all files containing stats from the training
- `retinanet-experiments` ..... the directory with the RetinaNet
- `retinanet-middle` ..... the directory with the modified RetinaNet
- `scripts` ..... the directory with all the scripts used

Appendix B

Additional images

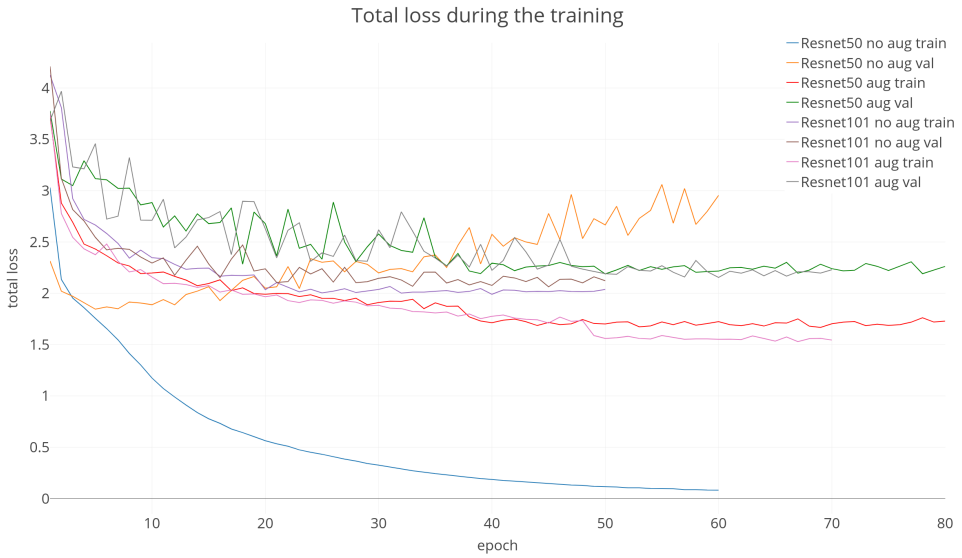


Figure B.1: The visualization of the totat loss (regression + classification) on train and validation set during the training.

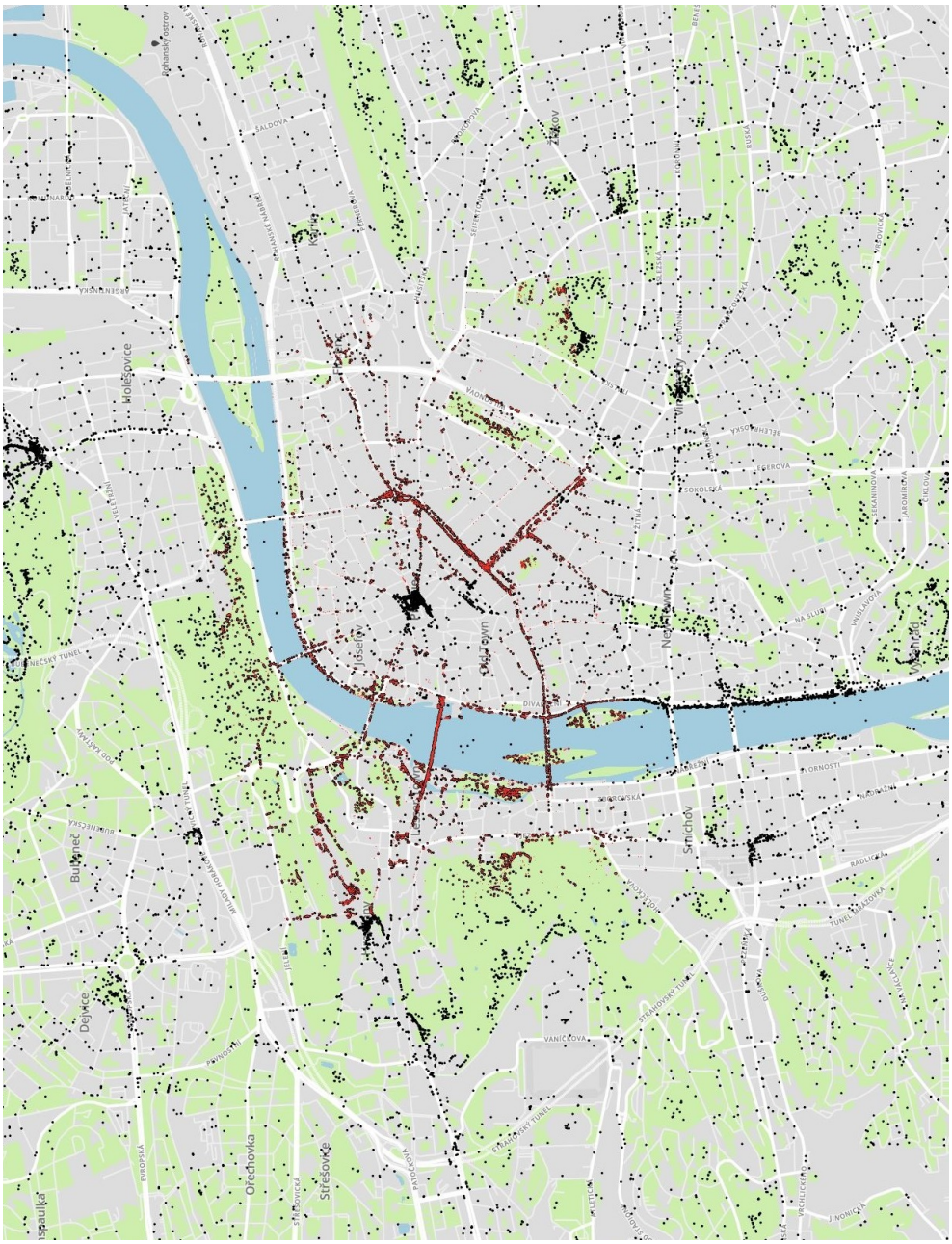


Figure B.2: On this figure you can see all people detected by the best model visualized on a map. Also note the original annotations vizualised in red color. (the map is rotated, the north is on the left side)

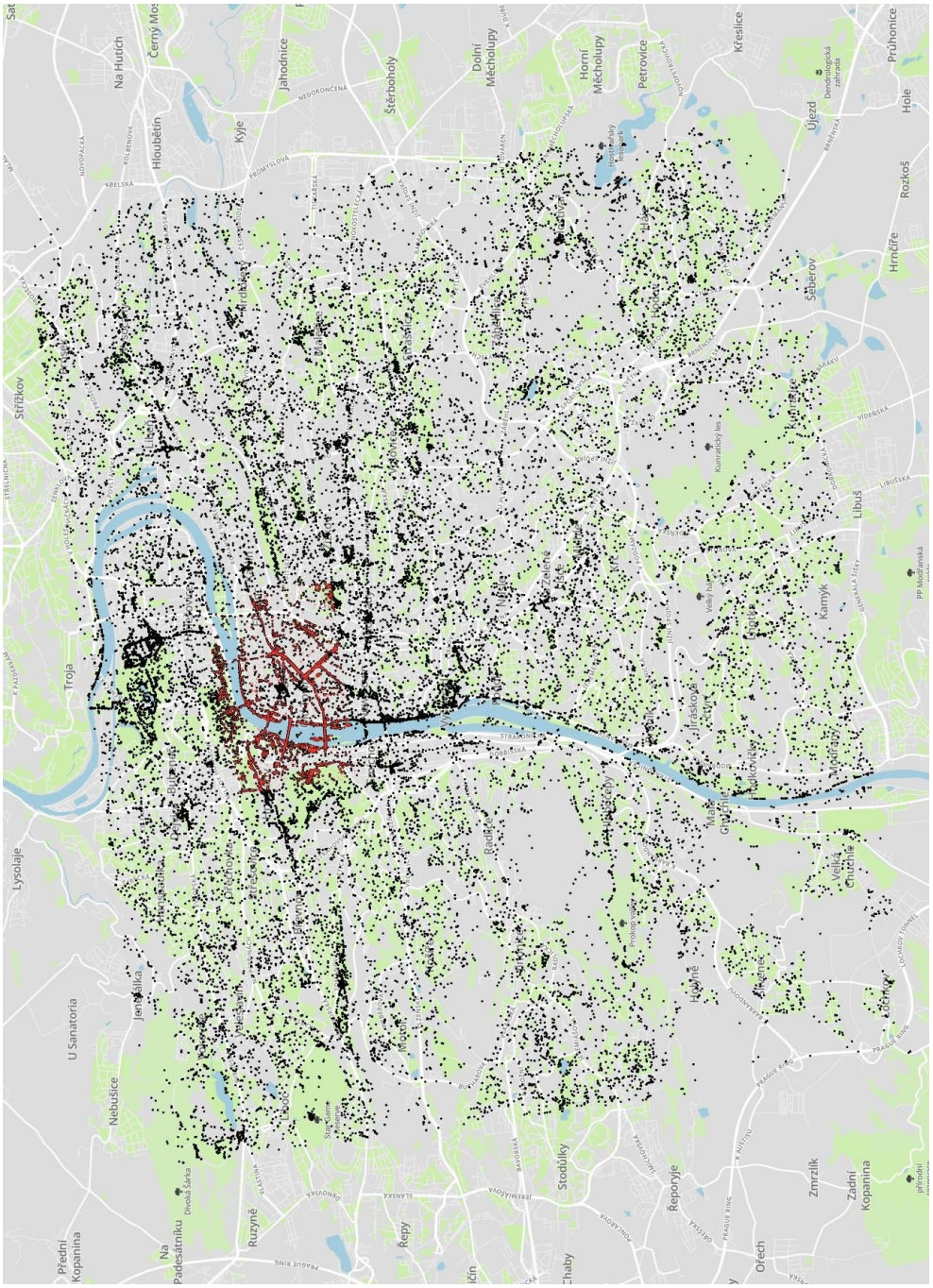


Figure B.3: On this figure you can see all people detected by the best model visualized on a map. You can see the whole area on which I have made predictions in comparison with the original data shown in red.

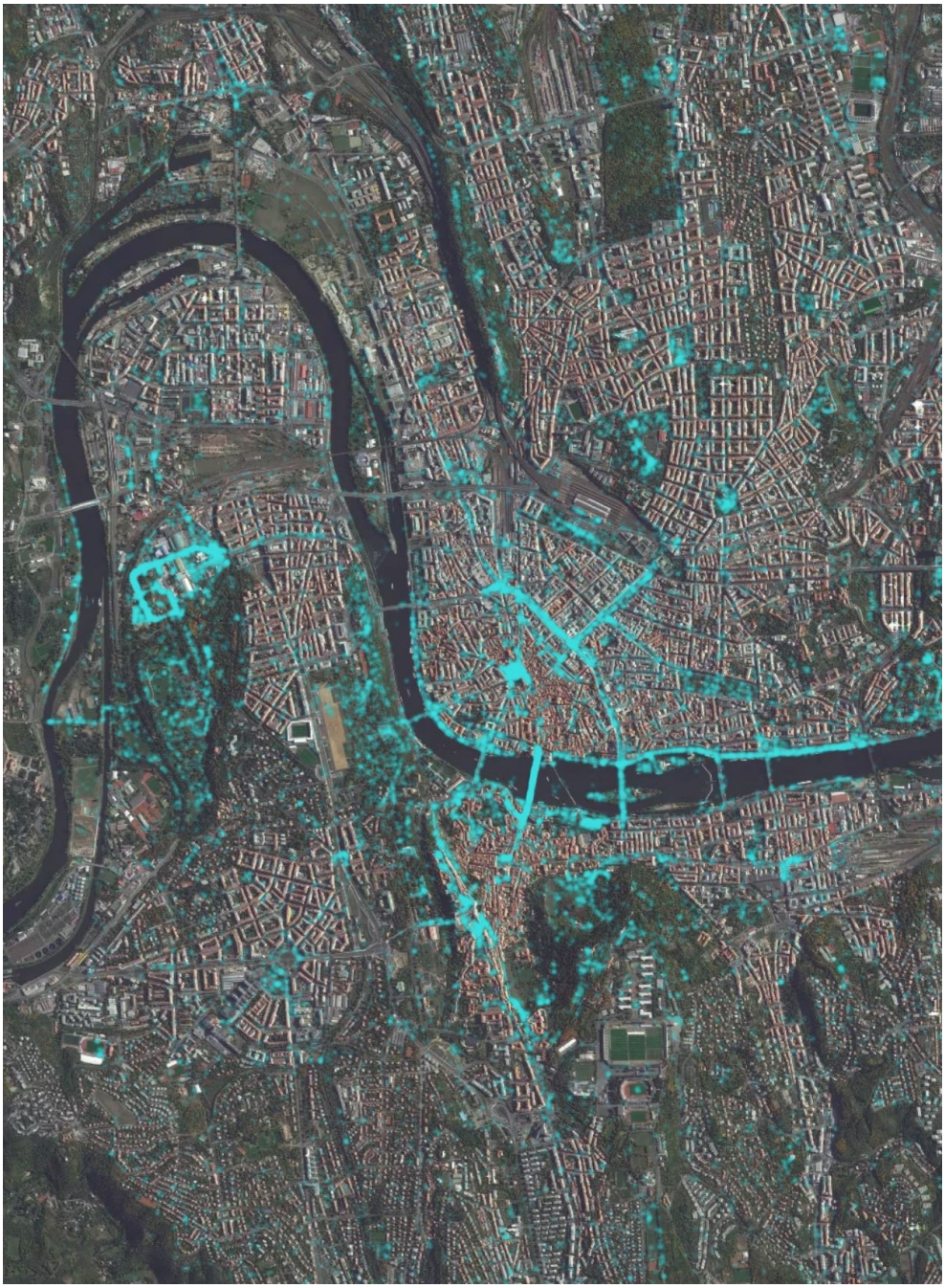


Figure B.4: This figure shows detections visualized as a really simple heatmap. The more the color is saturated the more people is on the given area.