Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science and Media

# A Privacy-Aware Mobile Object Detection Tool for Urban Surveys

Master's Thesis

Christopher Getschmann
born on 01.11.1990 in Bad Dürkheim

Matriculation number: 116326

**First referee:** Junior-Prof. Dr. Florian Echtler
**Second referee:** Vertr.-Prof. Dr.-Ing. Sven Schneider

Submission date: 29.08.2018

Whenever possible this work is built on top of free and open-source hard- and software. This would not have been possible without:

- The RepRap Project
- OpenSCAD
- Arduino and avrdude
- python, matplotlib and numpy
- tensorflow and the tensorflow object detection framework
- openCV
- labelimg
- d3
- tex and latex
- and many more projects

# Abstract

For urban planning, the tracking of pedestrian flows is valuable information. Existing solutions, however, either rely on labor-intensive manual counting, are prohibitively expensive for short-term analysis, or raise significant privacy concerns by sending image data to a central service. This thesis documents a ready-to-use tool for automated surveys on public streets and places based on low-cost Android smartphones that should enable researchers and citizens alike to gather quantitative movement data. Depending on the phones computing power one of three different Convolutional Neural Networks is used to process camera input directly on the phone. No image data is stored, thereby avoiding any privacy concerns or legal issues regarding video surveillance of public spaces. To evaluate and discuss neural network architectures for mobile object detectors performance measurements for different phones are compared. In addition to the survey tool, a 3D-printable weatherproof enclosure for outdoor placement and a web-based visualization tool to evaluate the gathered datasets are proposed.

# Contents

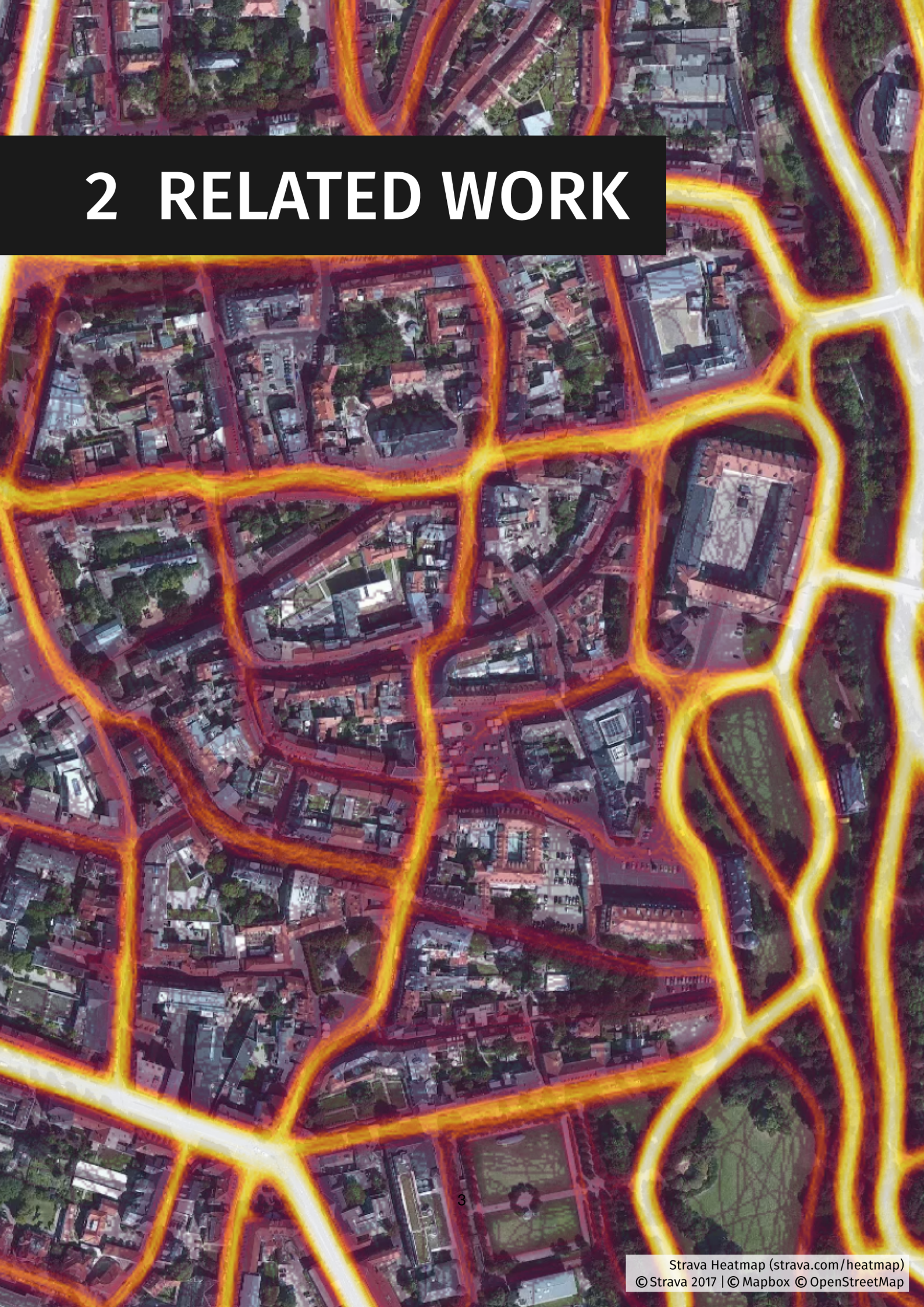
# 1 MOTIVATION

## *1 Motivation*

When analyzing cities and urban space, planning traffic or evaluating development projects, the whereabouts and movement patterns of pedestrians are a key element. While surveying the site in person will give invaluable insights, quantitative data is necessary for reliable decision-making. Usually, this data is gathered only by expensive permanently installed systems or through labor-intensive field work and manual counting. If manual counting is conducted, those short-time samples are usually annualized and timespans of two hours should represent an average of the traffic patterns of a whole year.
A perfect tool to gather the largest amount of relevant information for a quantitative survey would count pedestrians, cyclists and other objects as well as recognize their activities and track their routes when moving. In addition to that, it could estimate age, gender, and ethnicity. Overall it would need to be versatile, precise and easy to use in a variety of places. While this exists and is commonly called a student with a clipboard, it would be beneficial to automatize as much of this as feasible to allow data collection over a large span of time. Such a tool for automatized urban surveys should support planners, researchers and citizens in gathering datasets on public places and streets. It would need to have a low entry barrier in terms of cost and technical knowledge and would be expandable should someone have the need to alter or adapt it to their specific needs.

We propose an app for multi-class object tracking, running on commonly existing hardware in the form of low-cost Android smartphones. It can be placed in public spaces and tracks pedestrian paths visually by recognizing pedestrians in camera images. The live camera data is processed on the device and raw footage is neither saved nor uploaded. This is done to guarantee people in public a freedom of surveillance, at least from our device. The decision to run locally is not undisputed: the architecture of a pedestrian counter is much simpler and will yield a higher accuracy if the computationally demanding task of image detection is run on a different computer or server, rather than directly on the capturing device. But to allow this, all captured image data would need to be saved and transferred. This would result in permanently aggregating and saving large amounts of high-resolution image data, making identifying unique persons easy. We explore the possibility to do all processing locally in order to offer a simpler and decentralized architecture, a better usability and a more privacy-preserving concept.
It can be noted that the scope of this work is broader than necessary to give a simple proof of concept and feasibility. We aim to present a ready-to-use tool and part of this is a 3D-printable enclosure to actually use the phone outdoor on location. The app and corresponding software is open-source to give others the option to adapt and extend it or allow a verification of the produced data. Consequentially hard- and software is fully publicly available and licensed under permissive licenses. The sole exemption of this is the annotated dataset of pedestrian images that is used for evaluating this work and will be used to retrain our Convolutional Neural Network models later. This dataset contains high-resolution images and makes it trivial to identify unique persons.
While currently it is possible to detect tracks and desire paths of pedestrians, the complementary functionality of counting and activity recognition will be added in the near future.
We imagine our app could be used to conduct density measurements, comparisons of traffic patterns and overall usage of spaces. Questions like *"where do people cross the street?"* or *"at which spots do people linger?"* can be answered. In addition to that, it can be used to track resource usage of public infrastructure such as parking spots. If one would be bold, one could even imagine citizens automatically counting parking offenders on bike lanes and use this data to argue in favor of infrastructure changes.

# 2 RELATED WORK



Strava Heatmap (strava.com/heatmap)
© Strava 2017 | © Mapbox © OpenStreetMap

## 2.1 Automated Pedestrian Tracking and Counting

To acquire an overview of the field of measuring pedestrian traffic automatically, it is useful to have a look at the most frequently used and installed technology. Several techniques and tools are common:

### 2.1.1 Infrared Sensors

Infrared sensors, such as Eco Compteurs Pyro Posts (see fig.2.1a)[1], are the most common pedestrian counter. An active IR diode creates an infrared light beam and every pedestrian or cyclists crossing the beam is counted. They have a low accuracy for higher densities but are inexpensive, unobtrusive and low-maintenance. The downside is that the sensor requires a shallow or funnel-like walkway geometry which makes posts with IR-sensors popular for store entrances or trekking-path monitoring and not for open spaces. While historically these IR sensors have not been able to detect the walking direction, now dual-beam IR sensors are available that acquire this data too.

**(a)** Infrared sensor post **(b)** Pressure mats before covering up

**Figure 2.1:** Basic pedestrian sensors (images taken from the Eco Compteur site)

1 https://www.eco-compteur.com/en/products/pyro-range/urban-post

### 2.1.2 Pressure Sensors

Pressure mats (see fig.2.1b)[2] for pedestrians and pneumatic tubes for bicycles are used for both long and short-term measurements but are less constrained by the walkway geometry. However, the downsides of pressure mats are similar to infrared sensors. Precision is very high for single pedestrians but drops for groups and in crowdy settings. To acquire the walking direction, two rows of pressure mats need to be placed.

The main advantage of pressure mats and IR sensors is their low price (especially for IR sensors) when installed permanently. In addition to that, these devices have no impact on privacy and there is no problem of public surveillance.

### 2.1.3 Smartphone Sensors

The usage of smartphone sensors, either active or passive has emerged shortly after the widespread adoption of Wifi-enabled phones. Passive tracking of pedestrians happens by installing static devices in retail stores or public spaces that record Wifi or Bluetooth signals emitted from their phones. Some vendors even offer software to turn own routers into counting devices[3].

In addition to this localized data acquisition, position data is acquired by mobile network operators that track and triangulate users based on cellphone antenna data and sell access to these pseudonymized datasets. In Germany, one of those companies is Motionlogic[4], a subsidiary of the Deutsche Telekom, the countries largest telecommunications carrier.

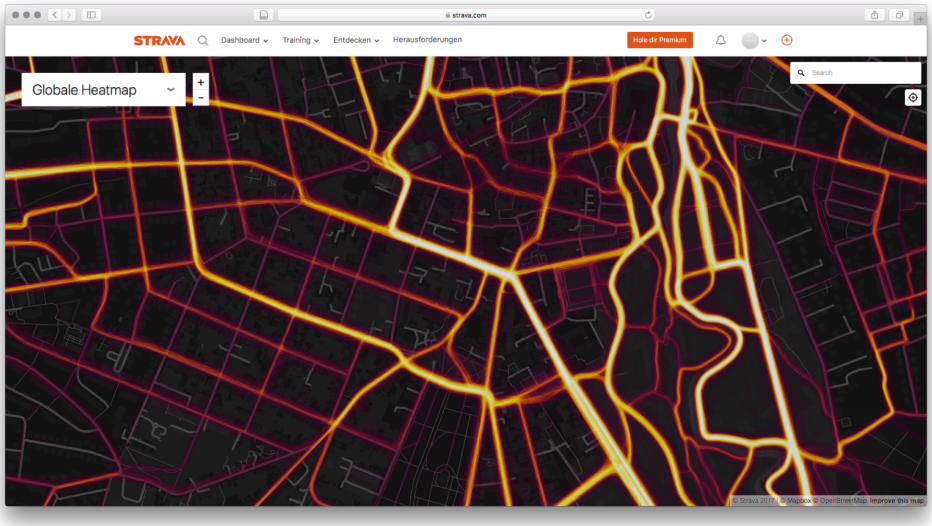



**Figure 2.2:** strava heatmap screenshot

2 https://www.eco-compteur.com/en/products/range-slabs
3 https://bluemark.io/products/
4 https://www.motionlogic.de/blog/en/solutions/

Active tracking of smartphone users happens via installed apps or smart fitness trackers. A notable commercial product is Strava Metro[5], which is the dataset and visualization product for urban planners by the fitness band company Strava. A part of this data is available on the Strava Heatmap (see fig.2.2). However, these datasets incorporate mostly sports activities such as jogging or cycling, which makes it more useful for analyzing these recreational traffic patterns than general traffic. Additionally, this data is biased towards social groups that actually purchase and use fitness tracking devices to quantify their workout.

It should be noted that there are not only commercial products, but also projects where movement data can be donated to create a dataset. One of those instances has been the Cycletracks app in San Francisco[6] which enabled cyclists to share their rides with the San Francisco Transportation Authority. This data was used to help traffic planners understand which impact traffic elements like separated bike lanes have. After the project was finished in 2010, the app has been adapted by 18 other cities at the time of writing.

All these methods have in common that they are able to generate only a ballpark estimate of pedestrian or cyclist traffic since not every person carries an active smartphone or tracker. Access to datasets from carriers or tracker companies is expensive, but offers a higher amount of data in relation to tracked area and time range in comparison to other methods, even though accuracy may be low.

### 2.1.4 Thermal Cameras

When privacy is a concern, often research in automated pedestrian counting is conducted with infrared cameras. In addition to being less privacy-intruding, infrared footage allows better foreground/background separation and improves detection accuracy [1].

Commercial IR-based pedestrian trackers are usually limited in their operational range and mounted overhead like xovies[7], the density.io tracker[8] or FLIRs traffic cameras[9]. When combined with stereo vision, these cameras perform well in dense spaces and can cope with occlusion. Some of these sensors do not count or track every pedestrian or vehicle in their field of view but only count the presence of an object in 2 to 8 preconfigured boxes.

### 2.1.5 Optical Cameras

Optical systems can be divided into monocular and stereo-camera setups. Devices with stereo cameras have the advantage of easier background subtraction and object separation but must be mounted overhead and have only a limited field of view (about 4x4m are typical).

5 https://metro.strava.com/
6 https://www.sfcta.org/modeling-and-travel-forecasting/cycletracks-iphone-and-android
7 https://www.xovis.com/en/xovis/
8 https://www.density.io/
9 https://www.flir.com/traffic/

Monocular optical systems are either permanently installed such as traffic cameras or mobile systems for temporary measurements. Mobile optical systems are the most relevant traffic measurement systems for this work and are thus discussed in depth in a separate section:

## 2.2 Monocular Mobile Object Detectors

### 2.2.1 Miovision Scout


**Figure 2.3:** Miovision Scout camera and processing unit mounted on an extendable pole (Image taken from the Miovision website)

The Miovision Scout [2] is a mobile traffic counter camera. It can be mounted on an extendable pole (see fig.2.3) and captures video footage from the highest point of the pole. Video data is uploaded to Miovision servers and traffic data such as counts, waiting times and trip times is extracted there. The Scout is marketed mainly for creating surveys and measurements of streets and intersections[10]), while technically it could be used for pedestrian counting as well.

### 2.2.2 Placemeter

The company placemeter [3] uses low-cost Android smartphones to gather video data (see fig.2.4) of streets and public places. The camera output from the devices is streamed to a central server where object detection is performed. Customers are billed per video stream and object type (car, pedestrian, etc.). The extracted information includes counts and densities of cars and persons, pedestrian trajectories (entering/exiting a store or building) and vehicle speed (if a car is speeding or traveling within the speed

10 `https://miovision.com/datalink/scout/`


**Figure 2.4:** the placemeter output and Android phones running the acquisition app (images taken from the Placemeter commercial material)

limit). Every phone requires a constant broadband network connection to stream video data and is not supposed to run on battery. Placemeter was acquired by Netgear and the service is no longer available to new customers.

### 2.2.3 Modcam

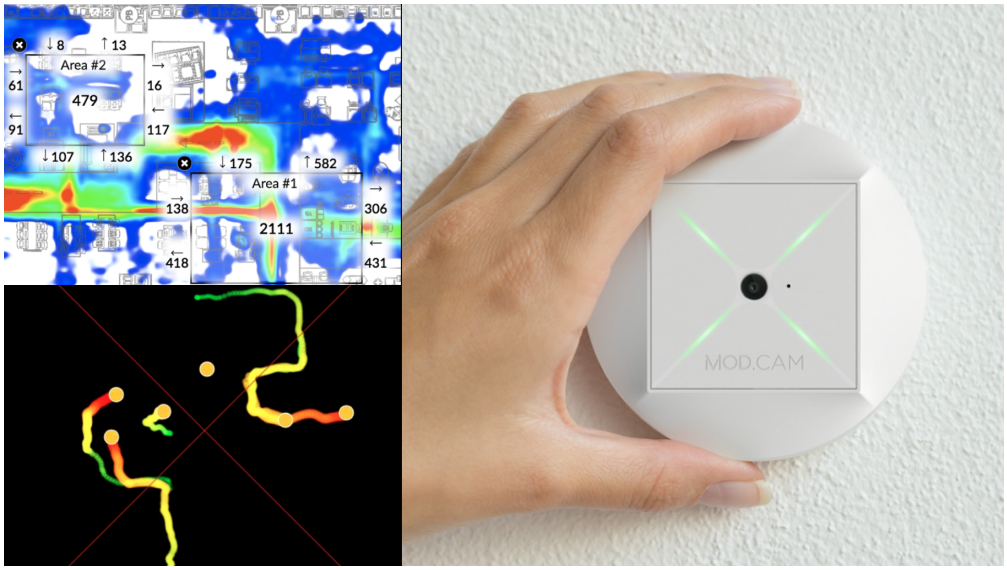



**Figure 2.5:** The modcam device and its heatmap output (images taken from the modcam press kit)

The company modcam [4] is offering a pedestrian counter with the same name, aimed at retail and facility management. It is a monocular device with a fisheye lens. The detection algorithms are running directly on

the device, power is supplied by a wall socket and it is meant to be mounted from above facing down (see fig.2.5). The software outputs counts as well as position heatmaps.

### 2.2.4 opendatacam

**Figure 2.6:** the opendatacam on location with the web based control interface (images taken from the opendatacam website)

Groß et al. [5] built and documented the opendatacam, a proof of concept for an open source mobile object counter. A Nvidia Jetson Board (an embedded board with a combination of CPU and GPU optimized for power consumption) is used to capture webcam input on-site. The video data is processed in-situ using the YOLO object detection algorithm. Part of the project are stencils to make a rain-protecting bag from plastic sheets (see fig.2.6) and soldering tutorials to build the battery connector and a discharge-protection circuit.

While this serves well as a proof of concept, there are several key problems:
It is costly to purchase; the used Nvidia Jetson TX2 development board is priced at 600 Euro at the time of writing. The enclosure is the minimum required safety protection for a bare board with exposed copper and wiring. It is neither practical nor safe and short-circuits of 12V Lithium-Polymer batteries can cause serious harm. In addition to that, makes the material choice and fabrication technique the enclosure complex to build and prone to errors while assembling. Setting up the software requires technical knowledge and an understanding of software development basics.

## 2.3 Visualization and application of pedestrian data

Capturing data is only half of the work, visualizing these datasets is important to infer information.

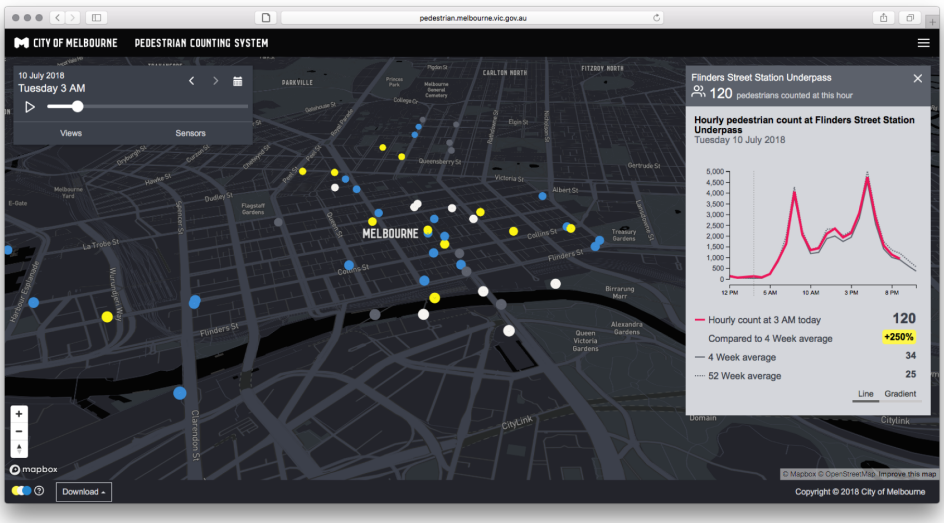



**Figure 2.7:** pedestrian counts in the open data portal of the City of Melbourne

The City of Melbourne installed permanent pedestrian counters in the city center in 2009 and publishes this data as a part of their open data portal[11]. The data is aggregated into one-hour bins and shown as a line plot averaged over 4 and 52 weeks (see fig.2.7). This can be used as an example of how raw counting data can be visualized and published.

## 2.4 Convolutional Neural Networks and Pedestrian Detection

The task of detecting objects in camera images is one of the fundamental challenges for our survey tool. Academic work relevant for this is discussed in depth in the detection chapter (see chapter5.2).

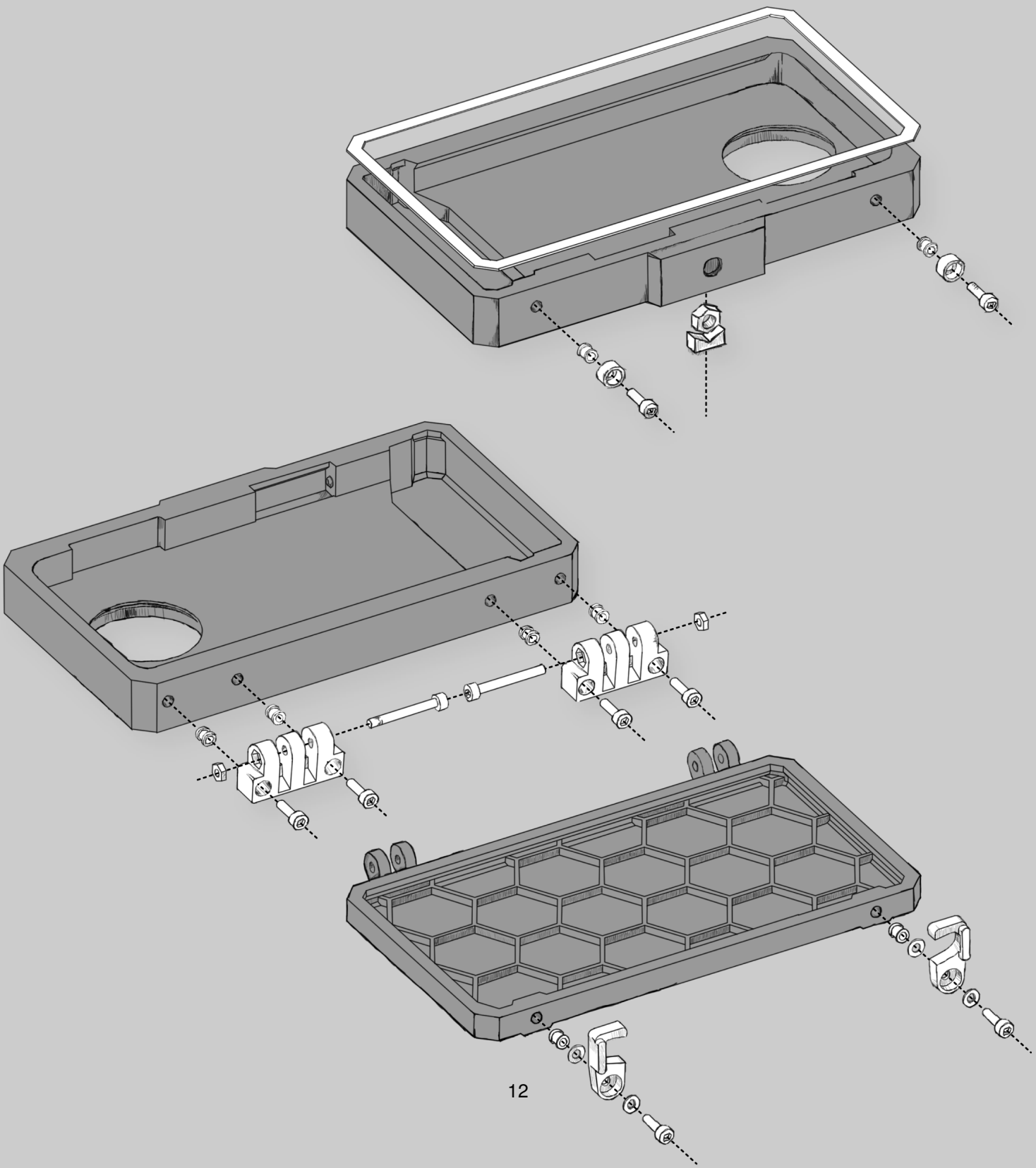
## 2.5 Conclusion

All in all, infrared and pressure sensors allow high-fidelity pedestrian counts for very small areas whereas smartphone based datasets offer low-fidelity data for very large areas. In addition to that, stereo and infrared-vision cameras allow better counting in small areas but are not applicable for larger spaces, such as public places. Monocular camera systems such as the Placemeter service or Miovision Scout make tracking objects on larger public spaces possible although they are expensive. However, none of the above-mentioned research or products is aligned with our approach of offering a ready-to-use dataset acquisition tool for research. None, with the exception of the opendatacam or the Cycletracks app, is open-source or in any way suitable for low-cost data acquisition. While Placemeter did follow the same low-cost approach in terms of hardware choice, privacy was not a concern in the architecture of their

11 http://www.pedestrian.melbourne.vic.gov.au/

system and it may not have been technically feasible because mobile hardware did need to come a long way since 2012 and on-device detection requires significant computational resources. The opendatacam concept is identical in its purpose and shares approach and some design decisions with our app, but it lacks versatility.

# 3 ENCLOSURE



To place the smartphone unsupervised on a public place for several hours or days a weatherproof enclosure is required. While placement behind a window often allows a simpler setup, only by using an enclosure many locations become viable. However, to the best of our knowledge, there is no off-the-shelf waterproof smartphone enclosure which fits most android phones.

In accordance with the overall objective of creating a simple and easy-to-replicate solution, we propose our own enclosure using 3D printing as the preferred way of fabrication.

**(a)** front **(b)** back

**Figure 3.1:** the fully assembled printed enclosure on location

## 3.1 Requirements

We identified several key requirements:

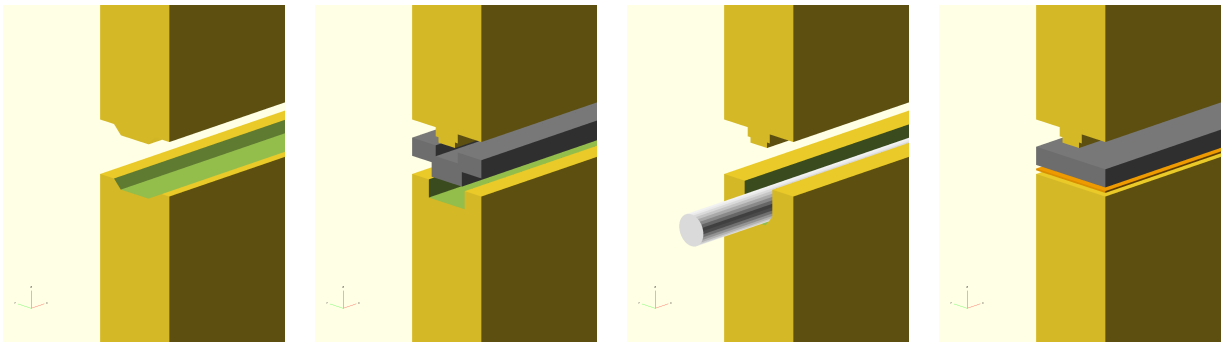
- weather-proof and suitable for outdoor use
- easy to reproduce on a low-end 3D printer
  - **No dual extrusion** printer should be necessary, no soluble support structures required
  - favourably **no flexible materials** which may require a specialized extruder or uncommon plastic
  - All non-printed parts need to be easy to source and should be preferably **standard metric screws** and fasteners
- usable for a wide range of smartphones and situations

- easy to use on location, no additional tools for opening or closing after assembly

## 3.2 Design decisions

To be weatherproof the enclosure requires a watertight or at least a water-repelling sealing mechanism. Rainwater should not enter the enclosure or, if it does, at least not pool around the electronic components. In accordance with the requirements stated above the sealing mechanism should be easy to print and should not require specialized o-rings or gaskets if possible.

To design a sealing mechanism which works for a phone enclosure, we tested four designs by submerging in water up to a depth of 10 centimetres for 30 minutes. [1]



**(a)** no seal **(b)** printed flexible seal **(c)** flexible filament **(d)** adhesive and rubber

**Figure 3.2:** cross section of the enclosure showing different sealing techniques

The different designs:

**No seal** [fig. 3.2a]

The trivial approach, an edge between top and bottom without any seal, does not offer suitable protection against water drops hitting the enclosure from above even when perfectly sanded and compressed by several screws.

**Printed gasket** [fig. 3.2b]

Another option is using a gasket printed from flexible filament[2] as a seal. The material used for testing is Colorfabb NGEN semiflex, which – unlike more flexible filaments – is printable on a standard printer without a specialized extruder (hardness: Shore 95A). However, we were not able to achieve a print with a surface finish that was smooth enough to form a tight seal. While this approach was theoretically the most promising, the resulting gasket performed worst among all experiments. We assume that these results

1 While the IP68 standard requires no minimum submersion depth, usually a depth between 1 and 1.5 metres is specified by manufacturers. Even though we did just test at 30 cm, technically our enclosure is IP68 compliant.
2 filament is the term for the line of plastic that is fed into the printer

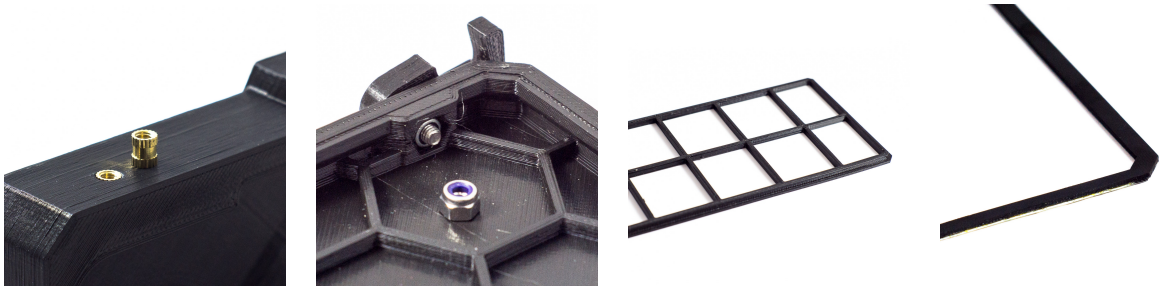
could be improved by not printing the bottom part of the enclosure and a semiflex gasket separately but by combining both parts when designing the enclosure. During printing the printer can be paused at a specific layer height to switch to the semiflexible filament. This allows printing with two materials without having to rely on a dual-extrusion printer and could eliminate warping and bed adhesion problems of the semiflex filament which were one of the main reasons for the subpar printing quality.

**Single strand of flexible filament** [fig. 3.2c]

Another approach is to leave a cavity and fill it with a single line of non-printed flexible filament, like an o-ring. The filament used for this test is Recreus Filaflex[3], the most flexible filament available at the time of writing (Shore 70A). While this technique seals sufficiently on the straight sides of the enclosure, the pressure does not suffice at the corners. A tiny amount of water is leaking.

**Rubber sheet with adhesive** [fig. 3.2d]

The most reliable seal was achieved by using a gasket cut from a sheet of 1mm thick rubber (hardness: Shore 65A). The cut can be done with a hobby knife and a printed stencil (either printed on paper or 3D printed, see fig.3.3c) or on a laser cutter (see fig.3.3d). The raw material is easy to source. It can either be regular rubber combined with an adhesive layer on the backside or superglue and any soft polymer sheet, such as neoprene, mousepad-foam, anti-sliding mats for cars or silicone insulation sheets. These seals are still tight after several months of usage and we recommend this solution for the enclosure, even though this results in a more complex assembly procedure.



**(a)** heat insert **(b)** nyloc nut **(c)** stencil to cut rubber sheets **(d)** adhesive and rubber

**Figure 3.3**

**A word about print quality:** A well-calibrated printer can achieve a surface quality and tolerances that allow the outer hull to be watertight even if screws pass through it (as long as self-tightening nuts are used, so-called nyloc nuts, see fig.3.3b]). The degree of precision that is required for this is hard to achieve, especially with some of the more difficult to print materials, such as PETG.[4] This is the reason why all screw connections are made from heat-insert knurled threads (see fig.3.3a). These do not require to pass through

3 Recreus Filaflex filament `https://recreus.com/en/14-filaflex-ultrasoft-70a`

4 Polyethylenterephthalat-Glycol (or PETG) is PET mixed with Glycol. As a material for 3D printing it withstands higher temperatures and is more resistant to impacts than other common filaments (such as PLA) but is more prone to printing errors while still being printable without heated printer enclosures or specialized print surfaces.

the outer wall to be held by a nut from the inside. The downside of knurled threads is adding a non-standard part to the bill of materials and a slightly more complicated assembly process, as these need to be heated up and inserted with a soldering iron. The alternative is to use captive nuts: this are standard nuts, but inserted in an already finished cavity during the printing process and enclosed in material after being overprinted. Compared to this technique heat inserts are easier to print and more reliable.

The camera hole is sealed by placing a standard 37mm UV filter in the hole and sealing it with acrylic glue or regular superglue. 37mm screw-in UV filters are common and easy to purchase, the connection between glass and metal thread is waterproof and these filters provide excellent glass quality compared to alternatives like acrylic glass.

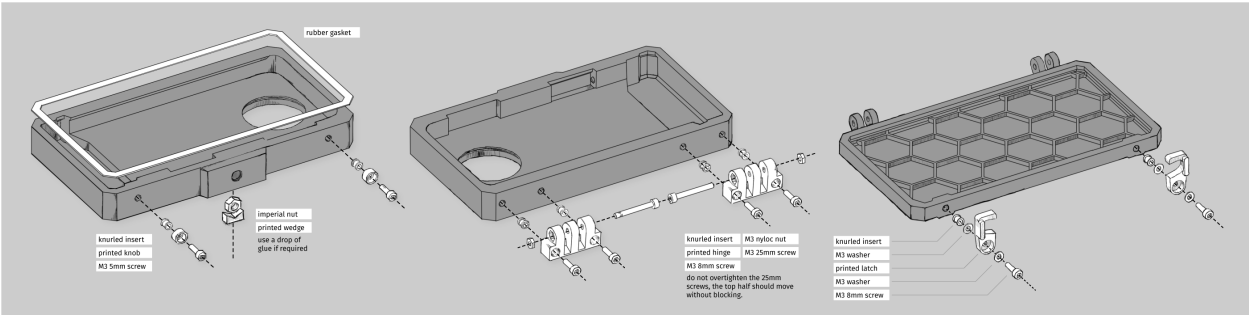



**Figure 3.4:** assembly instructions

Another option in theory to improve water-resistancy is adding one or several layers of paint or another coating. To determine if this would make a difference, several enclosures have been sprayed and sanded with primer and acrylic color. This did indeed seal minor imperfections in the layers of the outer wall of the print and additionally improved the aesthetic appearance. However, on a reasonably well-calibrated printer and if printed with three perimeters (the recommended print settings regarding the number of walls) the walls are sufficiently sealed and adding paint has only a purely aesthetical impact.

One of our other design goals is that the enclosure should be easy to open without any tools, thus the back is attached by hinges and hooks, not by standard screws or anti-theft screws reaching through the whole enclosure. This mechanism makes it harder to apply enough pressure for the seal to be watertight but has several other advantages. It is faster to operate and more convenient while reducing the chances of accidentally changing the viewing direction of the device while closing the top. This could make the device more susceptible to theft or vandalism, but this was not considered as relevant for the design of the enclosure. If the device is accessible to everyone and not placed on a lamp pole or balcony or other elevated position, no amount of reasonable engineering could protect a 3D-printed enclosure against a determined thief or even casual vandalism. By applying enough force it is possible to break the enclosure free of its mount, even if secured with a metal cable. Thus, except keeping a simple appearance to avoid the attraction of attention, no extra measures have been taken to protect the device.

Assembly after printing the parts is easy and can be explained in three simple drawings (see fig.3.4).

### 3.2.1 Mounts

There are several ways to place the enclosure, depending on the location. If a balcony is used, the recommended mount is a clamp with a tripod screw, widely available as photography or lighting equipment. If a metal facade or pole is available, our printable magnetic mount for metal surfaces (see fig.3.5) works well. On other locations regular tripods or flexible stands can be placed. To summarize: in most situations a suitable method for fastening the enclosure is available.

**Figure 3.5:** Magnet mount

### 3.2.2 Thermal protection:

One of the upper boundaries of detection performance of the whole system is set by thermal stress due to computational load and battery heat up. Therefore exposure to direct sunlight and the insulating properties of the enclosure are a factor to consider for this problem. If the enclosure is not printed using Polylactide filament (PLA), but with PETG or similar materials, direct sunlight exposure will not be a problem for the enclosure itself. If PLA is used, exposure to direct light on a hot day or due to contact with the phone itself the material will exceed its glass temperature of 45°C and become soft. Due to the poor heat transfer properties of the used plastic (and the fact that the process used for 3D-printing produces pockets of insulating air), the phone is sufficiently protected from the environment, but at the same time the enclosure will trap the waste heat of the phone as well. Another factor to consider for this is filament color. When it comes to heat absorption printing in black will produce the worst performing enclosure. We did this nevertheless since we consider a black enclosure as the most unobtrusive variant, but it should be noted that our temperature measurements represent the worst case. During usage, it has become clear that overheating is a problem for the phone. When running on maximum computational load, even without direct sunlight, some phones did reach temperatures above 45°C which would harm the enclosure if PLA would be used. Actions taken to reduce the thermal stress of the device are discussed separately in the software chapter 4.3.6.

### 3.2.3 The drawback of the chosen approach:

In favour of simplicity, a single enclosure does not work for arbitrary smartphones but needs to be printed specifically to fit a single model. To support any model of a given size, the enclosure would need moving parts and springs to apply pressure. This makes printing the enclosure and sourcing the required non-3D-printed parts more difficult. The 3D model used to print an enclosure is generated from a script via OpenSCAD[5], an

5 openSCAD: "The Programmers Solid 3D CAD Modeller" `http://www.openscad.org/`

open source parametric 3D modeling software. This makes it easy to add a new smartphone by extending an additional file with the exact dimensions of the required cavity, but it still requires basic knowledge of the software and probably one or two test prints.

### 3.2.4 Alternatives to self-printed enclosures

Using a simple smartphone holder can be sufficient if there is no risk of rain or dew during the installation and if there is no intention to disguise the appearance of the smartphone. These holders offer a standard tripod screw[6] which can be used to mount it to a clamp on a balcony, a suction cup on a window or directly on a tripod. Usually if purchased from online marketplaces, there are only two to three distinct variants and either of these will suffice for a standard setup.

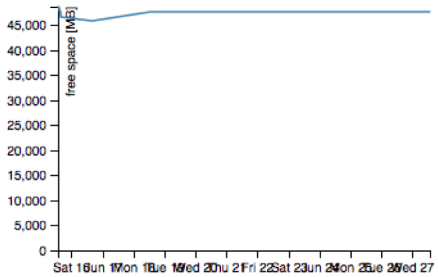
## 3.3 Further extensions to the enclosure

To increase the battery life and thus the observation time window of an installation, external power is required. This can be done by using an off-the-shelf USB power bank which can be placed in an alternative top-half of the enclosure. Another, more sophisticated approach is a custom design for a smartphone power supply, using standard 18650 Lithium-Ion cells and a custom driver board. This board can be controlled from the smartphone via a UART serial connection over USB-OTG where the board acts as a USB client, but still charges the phone. The benefit of this solution is a slightly increased battery life due to higher power conversion efficiency (the phone can decide when to be charged because lithium polymer batteries are less efficient if charging happens when close to maximum capacity). The mechanism for power line signalling (if the host or client supplies current) differs from the OTG standard on all three tested phones, but the design for a universally compatible proof-of-concept board is part of the source code in the digital addendum to this thesis (path */enclosure/powerbrain2*), although not discussed in more detail in this work.

The full bill of materials, printer settings and documentation on assembly is part of the documentation (see either the digital addendum of this thesis at the path */docs* or the documentation in the Github repository of the app: `https://volzotan.github.io/despat`).

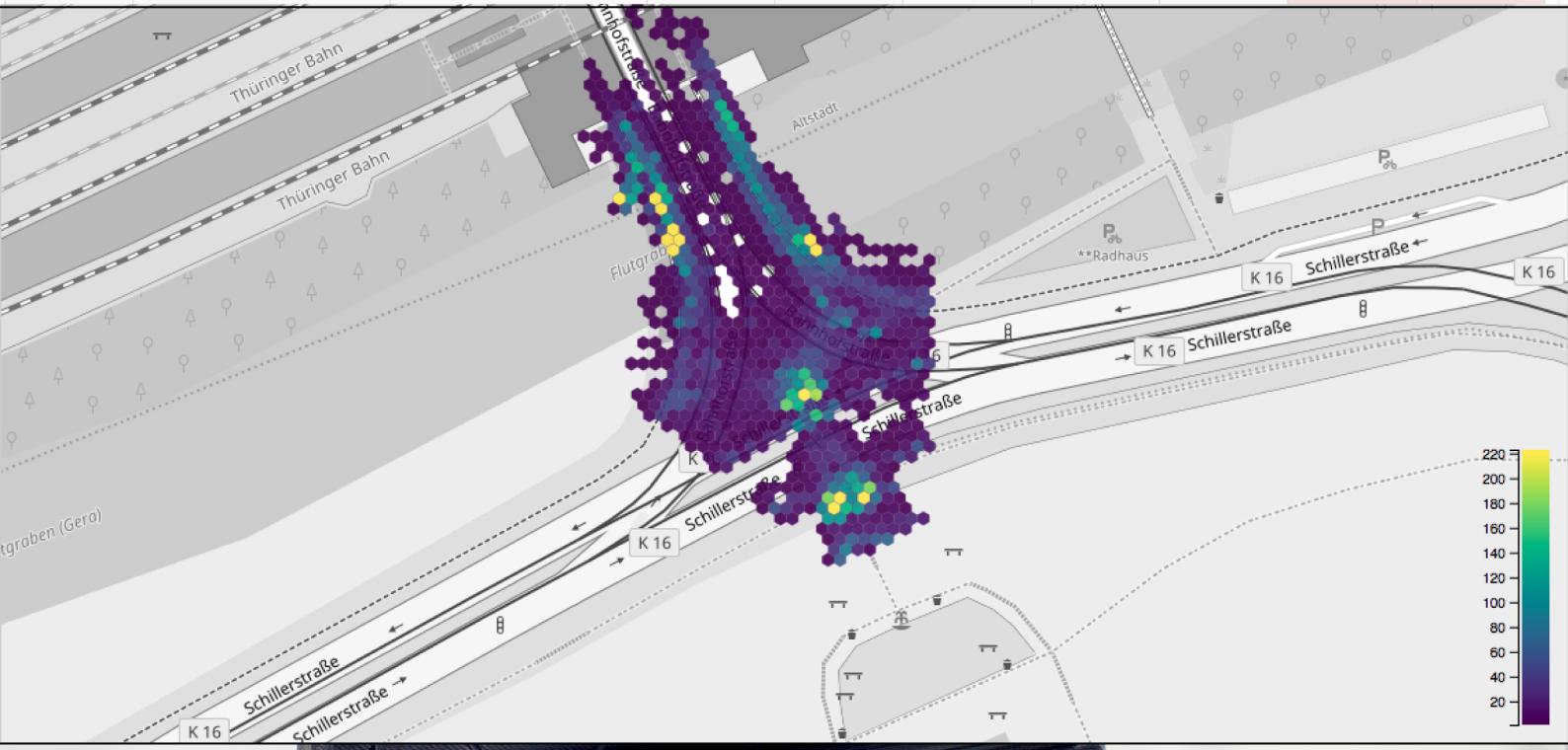
6 an imperial UNC (unified national coarse) screw, 1/4 inch in diameter and 20 threads per inch

last hour | today | all time


# 4 SOFTWARE

| | | | | | images in memory | free space internal | free space SD-card | battery built-in | charging |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 7032A963DAD0DC1A | 2018.06.21 - 18:22:34.783 | ZTE A2017G | | 1723 | 47564 | | 71 | □ |
| 14 | 7032A963DAD0DC1A | 2018.06.21 - 18:06:28.943 | ZTE A2017G | | 1704 | 47600 | | 19 | □ |
| 13 | 7032A963DAD0DC1A | 2018.06.21 - 17:01:07.029 | ZTE A2017G | | 1667 | 47676 | | 23 | □ |
| 12 | 7032A963DAD0DC1A | 2018.06.18 - 13:21:35.282 | ZTE A2017G | | 1652 | 47693 | | 27 | □ |
| 11 | 7032A963DAD0DC1A | 2018.06.16 - 16:10:41.294 | ZTE A2017G | | 1646 | 47719 | | 6 | ⊠ |
| 10 | 7032A963DAD0DC1A | 2018.06.16 - 15:01:56.205 | ZTE A2017G | | 1614 | 45884 | | 93 | ⊠ |
| 9 | 7032A963DAD0DC1A | 2018.06.15 - 16:05:16.396 | ZTE A2017G | | 1545 | 45918 | | 100 | ⊠ |
| 8 | 7032A963DAD0DC1A | 2018.06.15 - 16:05:15.297 | ZTE A2017G | 689 | 1391 | 46683 | | 6 | □ |
| 7 | 7032A963DAD0DC1A | 2018.06.15 - 15:17:28.463 | ZTE A2017G | 689 | 1391 | 46683 | | 6 | □ |
| 6 | 7032A963DAD0DC1A | 2018.06.15 - 14:57:12.924 | ZTE A2017G | 402 | 817 | 47504 | | 11 | □ |
| 5 | 7032A963DAD0DC1A | 2018.06.15 - 14:40:27.069 | ZTE A2017G | 281 | 575 | 47848 | | 14 | □ |
| 4 | 7032A963DAD0DC1A | 2018.06.15 - 14:25:39.004 | ZTE A2017G | 180 | 373 | 48125 | | 16 | □ |
| | | | | 91 | 195 | 48372 | | 16 | □ |




The full process of gathering a dataset consists of capturing the scene with the app, while optionally monitoring the phone with via a server. To evaluate the gathered dataset afterwards a visualization tool is provided and recommended but not mandatory.

Before describing the architecture and discussing design decisions, it is useful to get to know the process of recording a dataset with the app from a users point of view in detail:

## 4.1 Workflow

The very first step is to determine the best location for placing the camera to survey the scene. To minimize occlusions, a viewpoint angled at 30 to 90 degree to the ground plane is recommended. If a smartphone with a high-resolution camera is used, an extensive public place can be captured. If only a street needs to be monitored, a low-end smartphone will suffice and placement on a balcony or on the ledge of a window facing downwards is recommended.

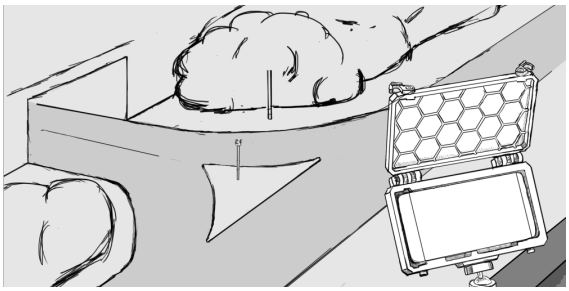


**Figure 4.1:** placement

On location, the user can place the phone in a clamp or an enclosure and start the app. At the first start, the app will ask for permissions to take and store images before displaying the main screen.
The main screen always displays the current camera preview if idle or the last capture while a dataset is recorded.


**Figure 4.2:** Start screen

Before recording for the first time, it is recommended (but not mandatory) to customize some settings. In the settings menu, the device name can be changed to something which is easily recognizable. The device name is part of the exported dataset and visible if data is synced to the server.

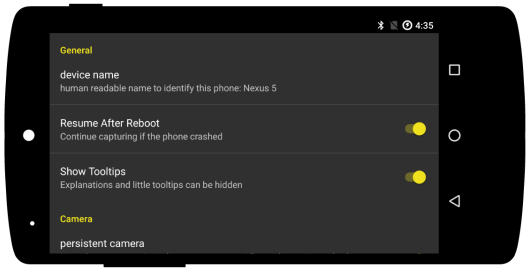



**Figure 4.3:** The settings pane

Once the phone is correctly oriented, the capturing can be started by hitting the REC button. While the screen is active, the last captured frame and additional information are displayed.


**Figure 4.4:** Recording

After stopping the capture, the app runs postprocessing operations in the background and the dataset can be inspected. For every dataset, the averaged image can be viewed, as well as additional information such as metadata, temperature data, and detected objects.
Before the data can be exported, the image needs to be mapped to a map. For this, at least 4 corresponding points between image and map need to be selected by the user.

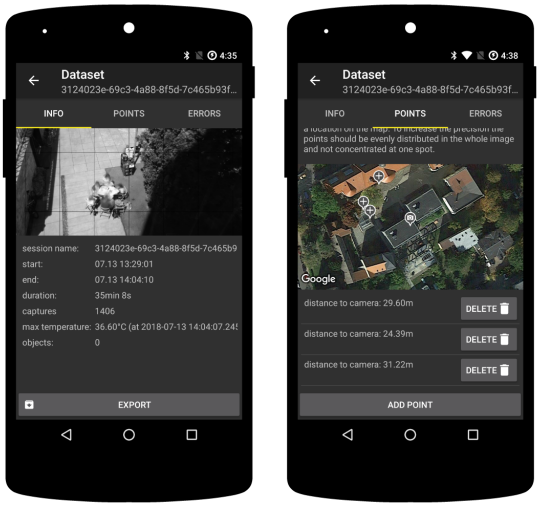



**Figure 4.5:** Dataset detail view

This can be done directly in the app. The map used for geocoordinate selection is the Google Maps Satellite layer, a georeferenced satellite photo.

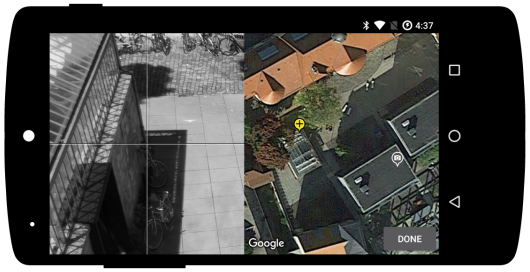



**Figure 4.6**

Once at least four points have been mapped, the complete dataset can be exported as a ZIP archive. The Android share function allows to save the archive on the phone, attach it to an email or place it in a dropbox folder.

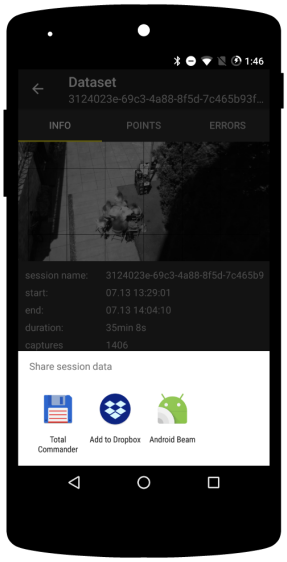



**Figure 4.7**

## 4.2 Architecture

## 4.3 The App

Android as a development platform is the first choice considering we want to make use of inexpensive smartphones and need to access the camera in the background without user interaction (this is simply not possible on iOS devices).

**About alternatives to smartphones**

When using smartphones as an embedded platform for computer vision tasks it may be easy to disregard alternatives from early on. While this is reasonable, we will nevertheless have a look at other choices and explain why it makes sense to turn them down. Instead of a smartphone (a highly integrated combination of camera, processor, antenna, and battery) different boards and devices could do the same job.

The Nvidia Jetson[a] is an embedded processor with GPU in a small package. While the processing power of such a platform exceeds smartphones, a combination of display, jetson board, battery and (optionally) a cellular antenna is above 1000 Euro purchase price only in components.

A more inexpensive combination would be a Raspberry Pi Zero[b] with a camera, battery and a custom board for power supply. While this is in the price range about 200 Euro, every single component performs worse than a smartphone of the same price. The ZTE Axon 7[c] for instance, has twice the processing power and four times the camera resolution for about the same overall cost. The computing

power of an Raspberry Pi board could be improved in the future by additional hardware like external processing[d], the main problem of higher cost and a tinkering-like approach remains.

However, the most important argument should not be only cost of purchase, but ease of use in terms of reproducibility. To run an automated survey with a smartphone, Android hardware may already present and only the installation of the app is required. When using embedded systems like a Jetson board or a Raspberry Pi, the single components need to be assembled and always require an additional power supply. This may be custom hardware which needs to be purchased, assembled and may require soldering. This raises the entry barrier in a way which makes it nearly infeasible to suggest any alternatives to smartphones.

*a* `https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/`
*b* `https://www.raspberrypi.org/products/raspberry-pi-zero/`
*c* `https://www.zteusa.com/axon-7`
*d* Google announced graphic processors and specialized hardware for machine learning operations in Raspberry Pi compatible hardware: `https://aiyprojects.withgoogle.com/edge-tpu`

Our App is targeted at Android phones starting with Android 6.0 (API level 23, Marshmallow, released in 2015) and supports devices up to Android 8.0 (API level 26, Oreo, released in 2017). At the time of writing this corresponds to about 64.4 percent of all active devices (which have the Google Play Store installed[1]). The minimum Android version can further be lowered by rewriting small parts of the App. Critical parts such as the camera code are already ready to run on older version due to our compatibility layer as described in the next section.

Processing and computation of the whole capturing process are managed each by an own Service. The relevant functionality is split into five distinct parts:

### 4.3.1 Image Capturing

One major consideration in app architecture is how the Android camera is handled and how a reliable capturing process can be achieved. To capture images at a fixed interval the app needs to trigger an image capture at actually fixed intervals (and not when the system schedules it). Additionally, the app needs to open the camera in the background without any user interaction or an active display.

Images are captured by an Android Foreground Service, a special construct in Android which is despite its name a background service (including a forced notification on the lockscreen). The service is the sole owner of to the camera and is responsible for metering, triggering and saving images.

The service can run in two different modes: **persistent camera** and **non-persistent camera**.

If the camera is not persistent, the camera module is closed after each capture. The camera is powered down and the app releases its wakelock, allowing Android to let the processor enter sleep mode. Before releasing the wakelock, an event is scheduled using the Alarm Manager to wake up in time for the next capture. Upon waking up, the camera is initialized and a full metering run is started to get an Autofocus lock as well as determine exposure and white-balance settings. This may take up to 1.5 seconds, depending on

1 Google Distribution Dashboard `https://developer.android.com/about/dashboards/`

hardware and scene. If this was not successful, the nearest approximation of the correct settings that could be calculated before hitting the timeout is used and images are captured anyway. By allowing the device to sleep in between captures, more than a day of battery life can be achieved. This comes at the price of a minimum shutter interval in this mode of 6 seconds.

If the camera is persistent, it is initialized at app startup and runs permanently in preview mode. While the camera is in the background, the buffer used by the preview is empty but the metering functions are still active. This means that the processor, as well as the camera module, are powered up and active. When the camera is triggered, images can be captured in an instant as the camera is active and autofocus/auto-exposure have already finished. This increases power consumption and heats up the device, but allows shutter intervals of less than 6 seconds. Depending on interval length and hardware, 3 to 10 hours of battery life are common in this mode among tested phones.

Both modes rely heavily on resource usage in the background and Android has several restrictions to this which are getting increasingly stricter with each release:

Starting with Android 4.4 (Kitkat), events triggered by the Alarm Manager, Androids abstraction layer for the Real Time Clock of the device, has been changed and is deliberately inaccurate.
Starting with Android 6.0 (Nougat) the Doze mode has been introduced. Doze mode starts at a random point in time after the display is turned off and the device stopped moving. In this state all events by the Alarm Manager are delayed to trigger several of them at once and they do not fire more than once every 9 minutes.
Starting with Android 8.0 (Oreo) background processes are not able to open the camera if they do not run a Foreground Service which makes itself visible to the user. However, all of these restrictions can be circumvented or disabled in a way that allows our app to run on the current Android version (8.0, Oreo).

In addition to that, device vendors add own restrictions to their Android versions, which can be disabled by the user or not. As an example: ZTE does not allow the Alarm Manager to be triggered after 90 minutes of inactivity if the user did not exempt apps from their custom "Power Manager" in a special Android settings pane. These restrictions are mostly undocumented, differ vastly per vendor, may require manual intervention and need to be tested on new phone types.

Another major problem are inconsistencies between devices in the quality of the vendor implementation of the Camera interface. Android offers the deprecated Camera API 1 and the newer and recommended Camera API 2. Since the camera subsystem is hardware dependent and needs to be supplied by the phone vendors, the Camera API 2 is just an update on the Android side. Hence, part of the API 2 is a legacy layer which maps operations and settings from the newer API to calls which the camera subsystem on older phones is able to provide. On at least one tested device, the Motorola E2, the implementation of the Camera API 2 legacy layer for background processes is inherently broken. Image buffers are not freed and after a random number of captures the camera subsystem crashes. To deal with these issues, we added our own compatibility layer which allows our app to optionally rely only on the Camera API 1, which is slightly slower but still enables us to use these low-budget phones.

These obstacles in relation to issues with the Camera API have been major setbacks during development.

### 4.3.2 Detection

After capturing every image needs to be analyzed for pedestrians and other tracked objects. Processing of the buffered images is done in batches by the detector Android service. This runs once a minute when the phone wakes up and takes care of the costly process of initializing and running the detection algorithm on every image. After the algorithm has processed the image, no image data is retained. The detection algorithm is discussed in detail in its own chapter.

### 4.3.3 Homography Transformation

The detection algorithm returns bounding boxes for detected objects. These need to be mapped to a single point on a map. Before transformation, bounding boxes are reduced to a single point. For cars, this is the center point of the bounding box. For pedestrians and bicycles, this is the center of the lower bounding box boundary (e.g. the feet of a person). In order to transform these points in the camera input to coordinates on a map, a Homography transformation is required. The transformation is done by multiplication with a homography matrix, which in turn needs to be computed. For this at least four corresponding points between image and a map are required and any additional points could be used to calculate an estimate of the overall error of all points. The homography transformation implementation of openCV[2] is used for this in our App. The points need to be manually selected on a map and an image of the scene, this is explained in the next section. Geocoordinates are saved as latitude and longitude in their decimal representation based on the World Geodetic System 1984 (WGS84). WGS84 is used by GPS vendors, Google Maps, and OpenStreetMap and thus our choice since no coordinate transformation must be done.

### 4.3.4 Image Averaging

The corresponding points to compute the homography matrix need to be selected manually but doing that on location is not sensible. Touching the phone while mounted, if it is even accessible at all, may move the mount or change the camera orientation which introduces an error to the coordinate transformation. Storing an image and selecting the points after the capturing has ended is possible, but yields the problem that footage of unique pedestrians would need to be saved. However, a single image where no pedestrians have been detected could be saved but it is not guaranteed that always an empty image will be captured or that it contains no false negatives.

Thus, we propose to average all captured images to create a mean image with no non-static objects. Using this approach even persons standing or sitting motionless will blur after processing only a few images. Captured images are averaged by adding their pixel values to a matrix and dividing the matrix by a scalar. This has the advantage that the images do not need to be saved and collected individually for averaging, but can be iteratively added. However, this is expensive in relation to memory as we hold these images as an openCV matrix, which is as an uncompressed bitmap in memory. To reduce the memory footprint, the images are downsampled by 4x and converted to grayscale before adding their pixel values to the matrix.

2 https://opencv.org/

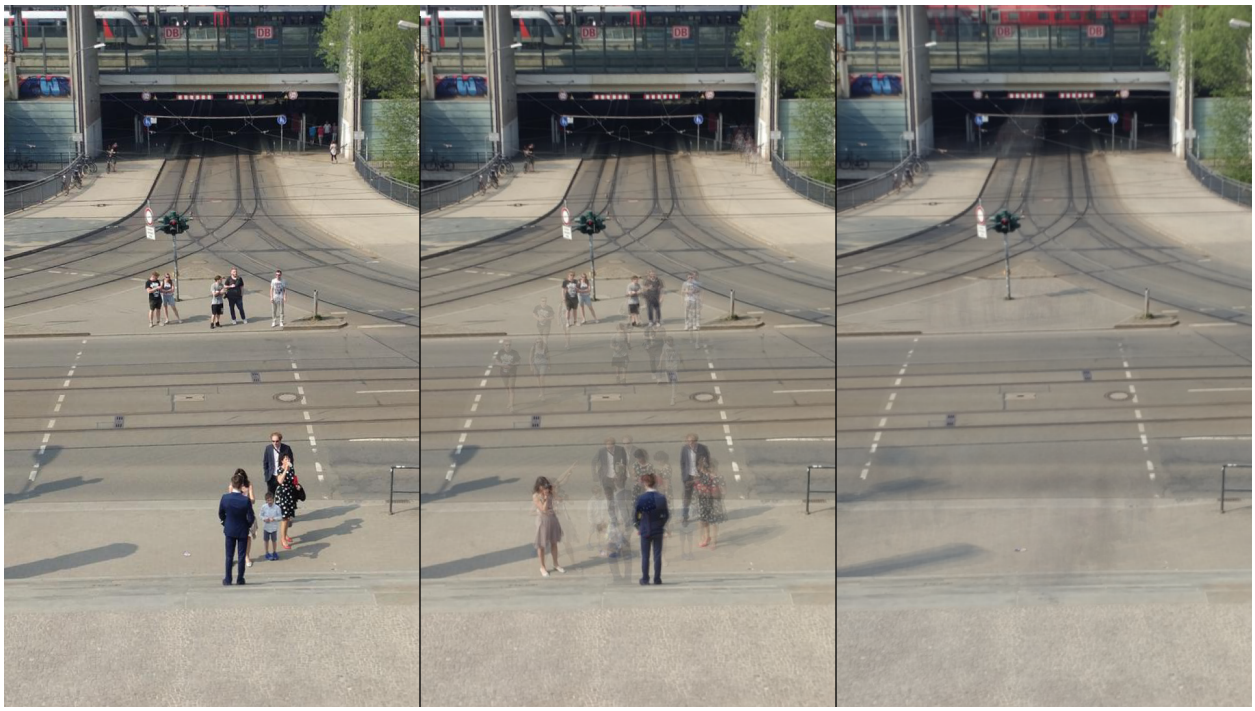


**Figure 4.8:** The image averaging process at start, at 20 seconds and at 2 minutes capture time

Up to 256 images are averaged per recording session, all further images are dropped. To achieve a fully pedestrian-less scene, a minimum of 30 to 50 images is usually sufficient (this depends on the amount of movement in the scene and the pixel size of the non-static objects).

Prior to summing pixel values in a 16bit unsigned integer matrix and dividing by scalar before export, we tested several other techniques. Normalizing pixel values from $[0, 256]$ to $[0, 1.0]$ and storing in a double matrix introduced visible banding artifacts due to rounding errors. Storing fractions by dividing before summing produced even worse results. Since basically uncompressed bitmap values are stored in the matrix, a data structure with a compression scheme should be a promising approach to reduce memory consumption, but no work has been done to improve this yet.

### 4.3.5 Data Export

The complete set of data from a single capturing session can be exported as a ZIP-archive by using the Android share functionality. This allows to save the file on the device, in a dropbox folder or directly attach it to an email. The ZIP-file can be opened on any device and consists of three files:

- a CSV file (comma separated values) with header information. It contains every single detection, including confidence values, class and coordinates
- a json file with session metadata such as device name and homography points
- the averaged image as JPEG

This archive can either be read by our visualization tool or the CSV files can be opened with a spreadsheet software such as Excel.

### 4.3.6 Performance and thermal issues

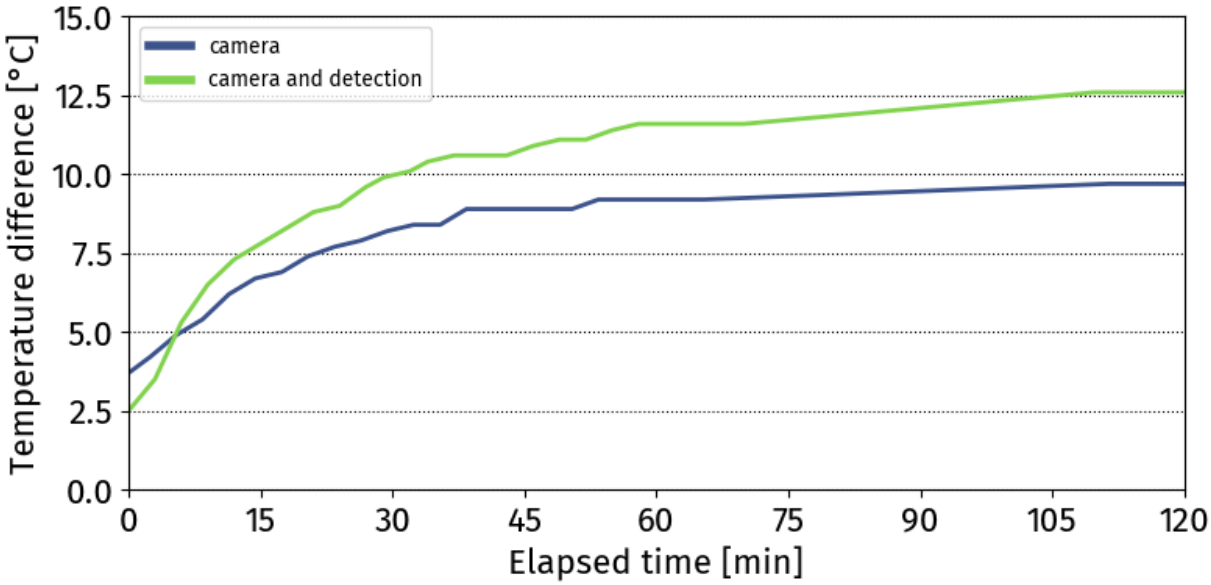
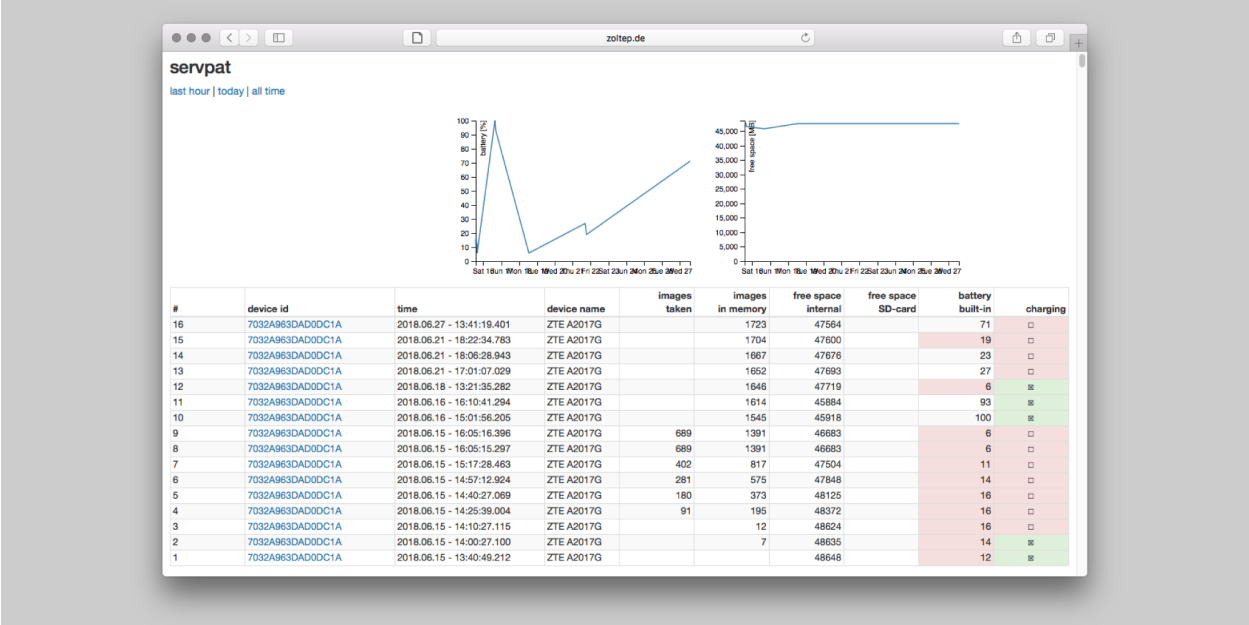



**Figure 4.9:** impact of different actions on phone temperature

When exploring the maximum performance several Android phones could deliver, it became obvious that the phones are not limited by the computational capacity of the processor itself but rather by their temperature. Most phones do not have a built-in temperature sensor but repurpose the battery temperature sensor for estimating their device temperature. While it can be assumed that the battery sensor underreports the actual device temperature, it is the best data available. When running the camera subsystem permanently (camera is active and providing a preview datastream), battery temperature increases by 10 degrees centigrade above ambient. This is caused solely by power draw which heats up the battery and the waste heat generated by processor and camera chip. If actually processing the image and running the detection algorithm at the very modest rate of one image per minute, the temperature increases further to 13 degrees above ambient. When running a phone constantly at full load, processing time increases as soon as the processor is throttled. Obviously, the heat management capacities of smartphones are not designed to support running the processor at full load over extended periods of time. This sets device temperature as the upper boundary of performance rather than unused computing time on the processor. It could be determined empirically that about 50 to 70 percent utilization keeps most phones within safe working temperatures. However, all these tests have been conducted with an ambient temperature of about 30 degrees centigrade and direct sunlight. It can be expected that the device is able to provide more computation time when deployed in wintertime, however, when ambient temperature drops below zero the heatup of the phone is even beneficial to protect the battery from cooling down and loosing capacity.

## 4.4 Monitoring Server




| # | device id | time | device name | images taken | images in memory | free space internal | free space SD-card | battery built-in | charging |
|---|---|---|---|---|---|---|---|---|---|
| 16 | 7032A963DAD0DC1A | 2018.06.27 - 13:41:19.401 | ZTE A2017G | | 1723 | 47564 | | 71 | □ |
| 15 | 7032A963DAD0DC1A | 2018.06.21 - 18:22:34.783 | ZTE A2017G | | 1704 | 47600 | | 19 | □ |
| 14 | 7032A963DAD0DC1A | 2018.06.21 - 18:06:28.943 | ZTE A2017G | | 1667 | 47676 | | 23 | □ |
| 13 | 7032A963DAD0DC1A | 2018.06.21 - 17:01:07.029 | ZTE A2017G | | 1652 | 47693 | | 27 | □ |
| 12 | 7032A963DAD0DC1A | 2018.06.18 - 13:21:35.282 | ZTE A2017G | | 1646 | 47719 | | 6 | ⊠ |
| 11 | 7032A963DAD0DC1A | 2018.06.16 - 16:10:41.294 | ZTE A2017G | | 1614 | 45884 | | 93 | ⊠ |
| 10 | 7032A963DAD0DC1A | 2018.06.16 - 15:01:56.205 | ZTE A2017G | | 1545 | 45918 | | 100 | ⊠ |
| 9 | 7032A963DAD0DC1A | 2018.06.15 - 16:05:16.396 | ZTE A2017G | 689 | 1391 | 46683 | | 6 | □ |
| 8 | 7032A963DAD0DC1A | 2018.06.15 - 16:05:15.297 | ZTE A2017G | 689 | 1391 | 46683 | | 6 | □ |
| 7 | 7032A963DAD0DC1A | 2018.06.15 - 15:17:28.463 | ZTE A2017G | 402 | 817 | 47504 | | 11 | □ |
| 6 | 7032A963DAD0DC1A | 2018.06.15 - 14:57:12.924 | ZTE A2017G | 281 | 575 | 47848 | | 14 | □ |
| 5 | 7032A963DAD0DC1A | 2018.06.15 - 14:40:27.069 | ZTE A2017G | 180 | 373 | 48125 | | 16 | □ |
| 4 | 7032A963DAD0DC1A | 2018.06.15 - 14:25:39.004 | ZTE A2017G | 91 | 195 | 48372 | | 16 | □ |
| 3 | 7032A963DAD0DC1A | 2018.06.15 - 14:10:27.115 | ZTE A2017G | | 12 | 48624 | | 16 | □ |
| 2 | 7032A963DAD0DC1A | 2018.06.15 - 14:00:27.100 | ZTE A2017G | | 7 | 48635 | | 14 | ⊠ |
| 1 | 7032A963DAD0DC1A | 2018.06.15 - 13:40:49.212 | ZTE A2017G | | | 48648 | | 12 | ⊠ |

**Figure 4.10:** data view for a capturing session on the monitoring server

Telemetry information and gathered detection data can be synced to a server (see fig.4.10). This is useful for monitoring device state and battery charge on installations lasting several hours or days and have network access. The web-based data visualization can be linked to an active recording session if syncing is enabled and corresponding points have already been selected. Additional features like tracking a phone through the monitoring server in case of theft may be added later. Although the app already allows syncing with the monitoring server, it is not yet ready for a use outside of a development environment and should be considered work in progress.

## 4.5 Visualization Tool

The exported data of our app can be processed with spreadsheet software such as Excel and visualized with Google Maps, but neither of these tools performs well on the task of visualizing pedestrian density data. Since data acquisition is pointless without means of examining and evaluating, we present our own visualization tool:

Our visualization tool displays density data for arbitrary classes on a binned heatmap or in a raw scatter plot on top of a map layer. The map layer is able to load map data and layers from Google Maps or OpenStreetMap. All layers are interactive and can be turned on or off. The map allows panning and zooming; the heatmap parameters regarding bin size and opacity can be changed, too. Data can be filtered by time, object class and recording device. Datasets gathered in parallel from different phones or sequentially from the same device can be combined.

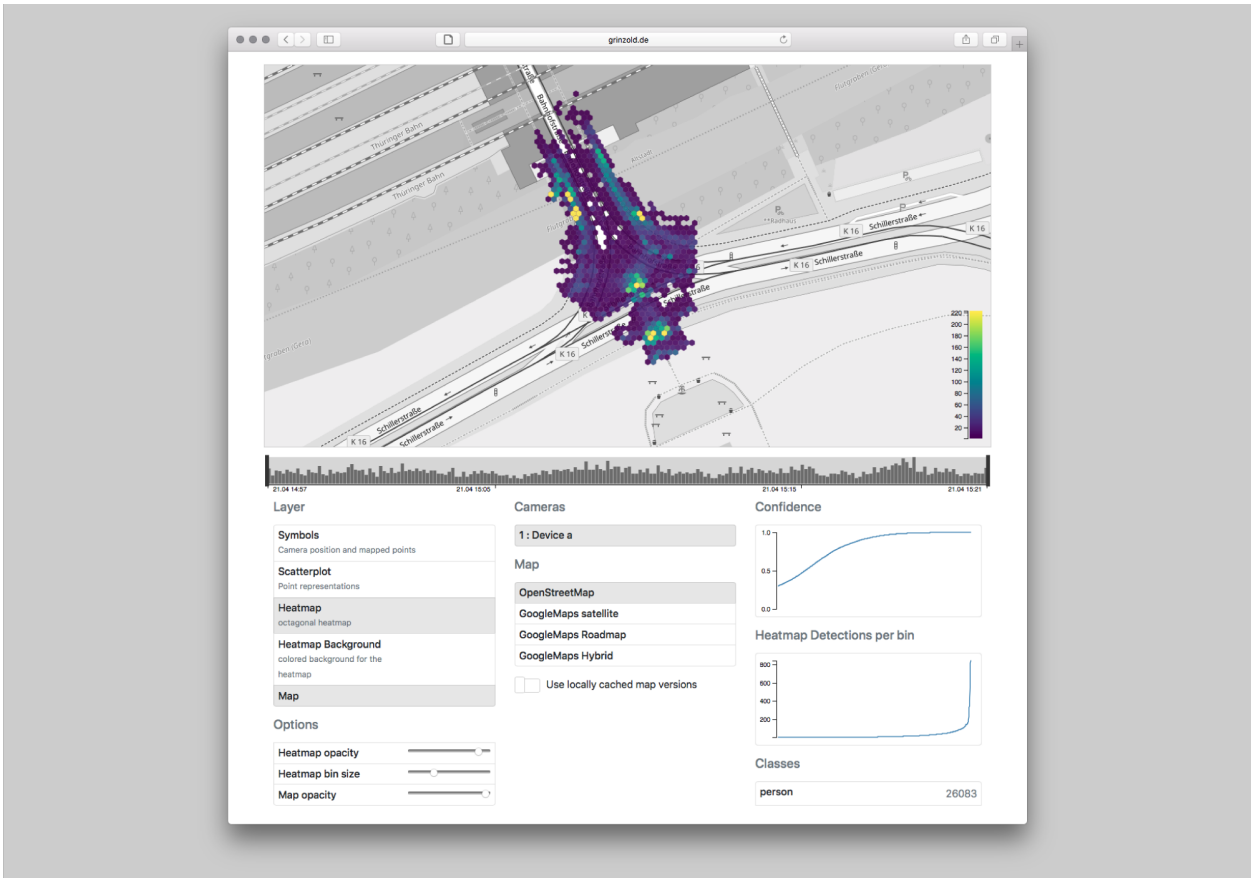



**Figure 4.11:** The web based view of our visualization tool

We did choose the heatmap with octagonal bins over the traditional heatmap for various reasons:

To display object locations and travel paths a heatmap like visualization is required. The traditional heatmap introduces the problem of choosing a blur value. Not enough blur and the viewer has the perception of a near infinite precision of localization. Too much blurring and information are lost. Binning map positions, however, allows having clear and distinct borders for single bins (or tiles if that is a more descriptive term). They are easy to visually parse and doesn't give the false presumption of infinite locational accuracy. In addition to that, the overlaying scatterplot can show clearly where every single detection happened. The octagon is the best geometrical choice since its geometry is efficient to compute and is the highest-order polygon that allows tiling without overlapping.

**Figure 4.12:** hexagon bins

**A word on colormaps**

Choosing the right colormap for the data at hand is one of those problems that are considerably harder than they seem at first glance. The default coloring scheme for heatmap-like visualizations has been

matlabs 'jet' for several years. The problem about jet is that it is not perceptually uniform [6]. The visual distance between two pairs of points in the color range does not necessarily reflect the difference of the underlying data. This results in the effect that jet creates in some data ranges the perception of contrast and hotspots in the visualization where only minor differences exist in the data.

The colormap for our visualization tool is viridis, a colormap from purple over green to yellow. Alternative colormaps which are perceptually uniform such as magma or inferno contain the color red, but all of these colormaps were rejected during testing. Some of our testers explained their rejection with the statement that these coloring schemes would be 'uncomfortable'.

A typical interaction with the visualization tool can be described like this:

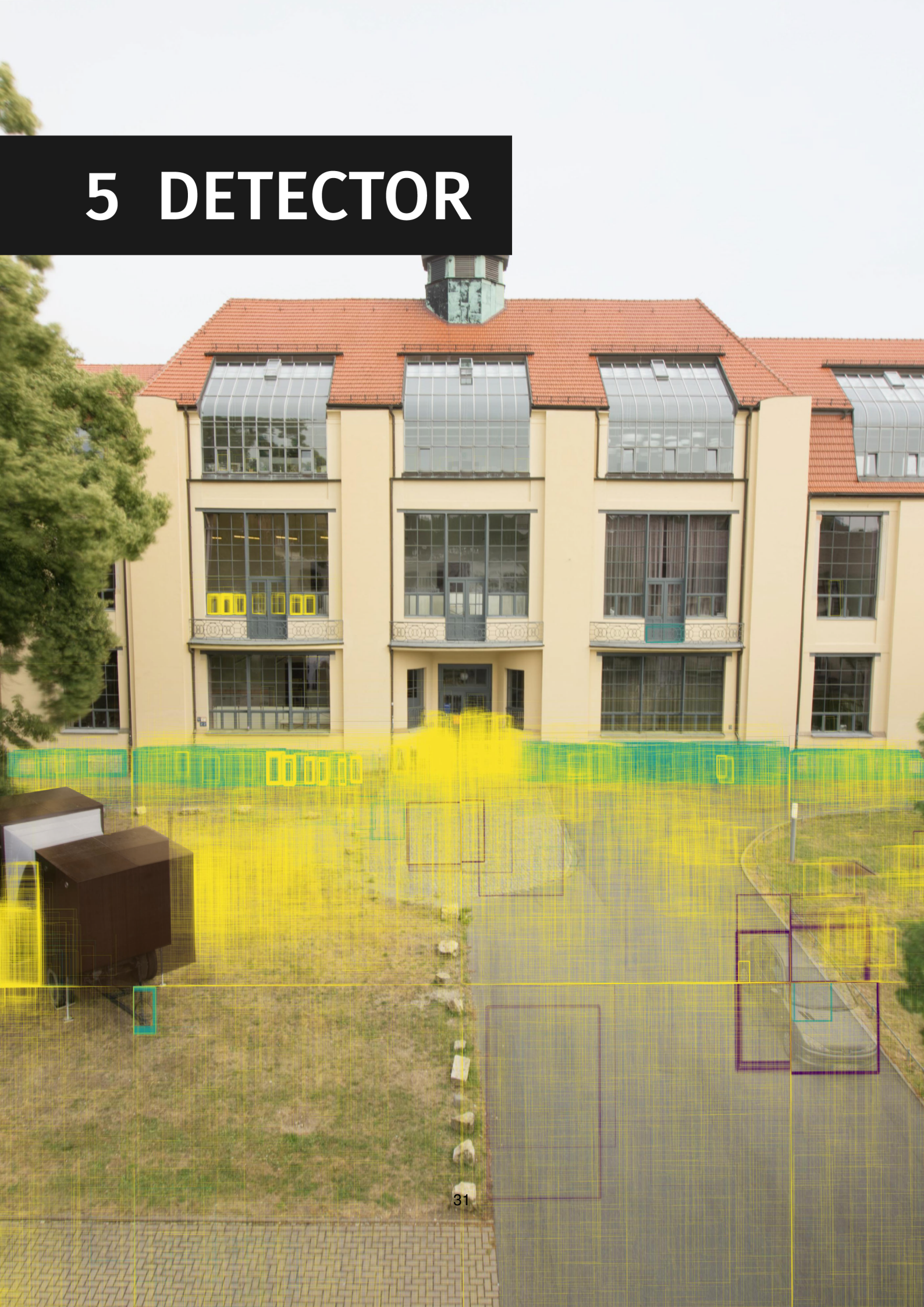
A dataset is gathered at an intersection. The data is uploaded to a dropbox folder and loaded by the visualization tool. The user deselects the first and last minute on the timeline, as this may contain erroneous data where the recording is already running but the phone is moved while placing or removing it from its mount. Since the intersection is spacious and the user examining the data is interested in how people enter and leave the waiting zones, the bins of the heatmap are too coarse. The user reduces the size of the bins to get a more fine-granular view and decreases the opacity to compare hotspots of waiting persons with the shape of the underlying streets.

The visualization tool is built as an SVG vector graphic, loaded in the browser. The javascript library D3[3] is used to interactively manipulate elements of this vector graphic to filter data points. It is currently already possible to save and extract the SVG from the website, but we plan on adding this as a more user-friendly feature in the future. While visualizing the data directly on the mobile device would be consistent with the overall goal of creating a complete all-in-one app for automated surveys, this would introduce several problems. The main problem is that complex user interfaces are hard to get right on small screens. While we assume that an – at least preliminary - visualization directly on the device is beneficial, the scope of this thesis had to be limited at this point.

The last not yet mentioned part of the software is the detection algorithm. It is discussed extensively in the next chapter.

3 https://d3js.org/

# 5 DETECTOR



## 5.1 A very recent history of pedestrian detection algorithms

One of the most prominent feature extraction and detection algorithms is the Histogram of Oriented Gradients (HOG) [7]. HOG can be used as an example to discuss a whole class of algorithms based on similar approaches and is thus described in detail: HOG is moving a sliding window at different scales along the image input. Every window is resized to 64 by 128 pixels and HOG features are extracted. This works by subdividing the image into overlapping 8x8 cells and computing the gradient for every cell (edges in the image are detected). The gradient directions are quantized into 8 bins and normalized with their neighbours. This results in 3780 values per window which are fed into a Support Vector Machine. The Support Vector Machine outputs either the label person or background for each window.

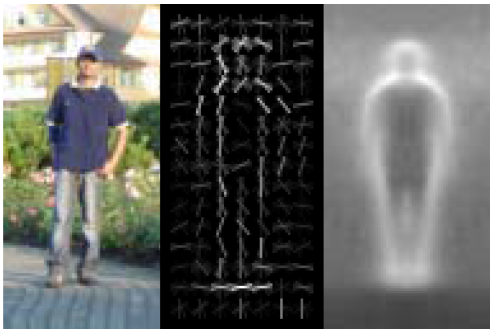


**Figure 5.1:** HOG: input image (left), visualized extracted features (middle) and average of the classifiers training data (right). Taken from the paper by Dalal and Triggs [7].

Improvements have been made after HOG by applying filters or extracting different features and using alternatives to Support Vector Machines as classifiers. An example of this are Filtered Channel Features (ICF) [8]. Other approaches are based on identifying single limbs (Deformable Parts Model) [9] to handle occlusion cases better.

Basically, these techniques use a deterministic algorithm without learnable parameters (called handcrafted features) to extract information from an image and a trained classifier which decides using the extracted features if a pedestrian is present in the given part of the image. This was the state of the art across all pedestrian benchmarks up until the ImageNet Large Scale Visual Recognition Competition in 2012. The convolutional neural network AlexNet [10] was a turning point, scoring best by a margin of 10.8 percentage points to the submission placed second. This and further work proved that convolutional neural networks are able to outperform other methods for the task of image recognition and subsequently it was shown that this could be transferred to image detection as well.

Before neural networks for pedestrian detection will be discussed, it is useful to have a look at convolutional neural networks in general.

## 5.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a subtype of an artificial neural network. Neural networks are algorithms that apply functions on input data based on collections of connected nodes, so-called artificial neurons. These nodes represent activation functions that combine outputs from other nodes and learnable parameters (called weights) to compute their own output. Usually, these nodes are stacked in layers and

data is passed from layer to layer. A CNN is a neural network that applies learnable convolution filters to image input data.

To understand the structure of CNNs, it makes sense to explain the architecture of an image recognition network in comparison to the more specialized object detection networks.

A typical layer in a CNN consists of several convolution kernels for filtering, an activation function and weights for the function. The input of such a layer is the output of the previous layer or the initial image data. This input is filtered with each convolution kernel and the result is fed into the activation function. A popular example for this is the activation function of the Rectified Linear Unit ( $f(x) = max(0, x)$ ). These outputs are called feature maps. During the training of the network, the filter kernels and the weights are adjusted to allow the network to produce the desired output, in case of a recognition network this is a high score for the correct class.

The whole CNN may be built by using several of these layers which produce outputs of decreasing width and height but higher depths, followed by fully connected layers (layers where a neurons output influences not only some but every neuron from the next layer). In between the convolutional layers may be additional layers of different kinds which perform operations such as pooling/subsampling on the data to further reduce the size, but do not contain learnable parameters. For fully connected layers the two-dimensional feature maps are reshaped to vectors and fed into nodes where every node is connected to every other node of the following layer. These last fully connected layers are meant to map convolutional filter outputs to classes.

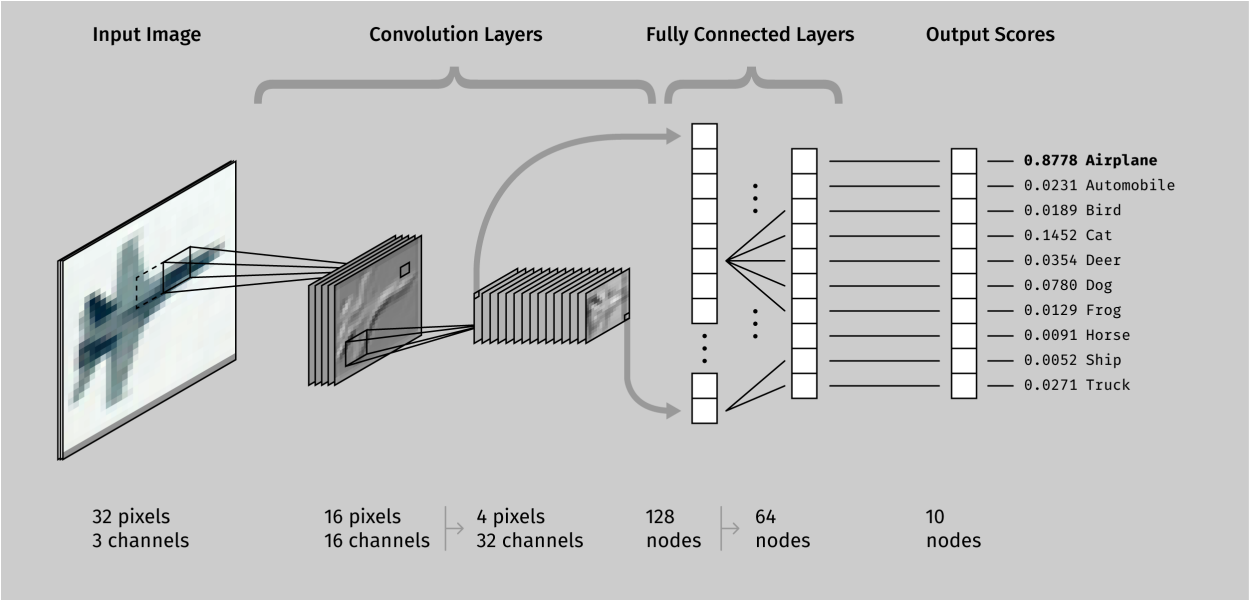



**Figure 5.2:** Simplified architecture of a Convolutional Neural Network (image data taken from the CIFAR-10 dataset and ConvNetJS)

To get a general overview of the structure of an image recognition network a tiny example with a certain amount of brevity and unsharpness is explained in the following (see fig. 5.2):

RGB Image data with 32x32x3 pixels is fed into the network. 16 convolution filters with a size of 5 and a step size of 2 are applied. This results in 16 feature maps with half the width and height of the original input. This is fed into an activation function which combines the 16 filter outputs and a set of learned weights (one per pixel). The output of the activation function has the same dimensions. The next layer has 32 convolution filters, resulting in 32 feature maps with 4 by 4 pixels output size. These feature maps are reshaped into a 512 element long vector, which is the input for the fully connected layer afterward. The fully connected layer computes an output based on the reshaped vector and its weights.

Due to the subsampling of convolution filters with high step sizes or discrete pooling layers, every region in the feature map of deeper layers is a result of a much larger area in the original image. This is called the receptive field and will be relevant when shortcomings of image detection architectures for tiny objects will be discussed.

The final layer usually does not use an activation function but a loss function to produce class scores. In the case of a cross-entropy (Softmax) loss, the output of the last layer can be interpreted as the networks confidence that the object is part of this class (this can be seen in the figure).

When discussing CNN architectures for object detection and their shortcomings in the next section, two details are relevant:

The input and output size of CNNs is fixed. There are exceptions when CNNs are combined with other network types, but a CNN where data is passed from one layer to the next without any processing has a fixed size image input and a fixed size feature map output.

The output of the first layers of a CNN usually have a higher resolution and contain low-level features such as edges and circles. By applying more convolutional layers and adding pooling/subsampling/max layers, the outputs have a lower resolution but contain information corresponding with higher-level features, these can be eyes, fur or scales for example if a model is trained to recognize animals.

## 5.3 Object Detection Networks

The main difference between an image recognition network and image detection is that a recognition network needs to output only a single class score for the dominant object in the image. A detection network needs to output class and location of an unknown number of objects.

The trivial approach would be to add a sliding window at different scales and feed the cropped and rescaled image region of the window into the recognition network, similar to algorithms like HOG. The major problem about this technique that it is very expensive in regard to computation time.

A notable algorithm is Region-CNN by Girshick et al. [12] which approached this problem by using a proposal algorithm on the original image as a first stage and extract and rescale only these image regions to feed them into the recognition network.

The evolution of this approach is Fast R-CNN [13] by Girshick which improves detection speed by not reapplying the convolution filters of the recognition networks first layer over and over again. The first stage of convolution filters are applied to the whole image and the regions proposed by the proposal algorithm are extracted from the output of the first layer. This reduces computation time, but still, the bottleneck of the proposal algorithm remains. While the CNN portion can run on a graphics processor (GPU), the proposal algorithm is not accelerated in the same way.

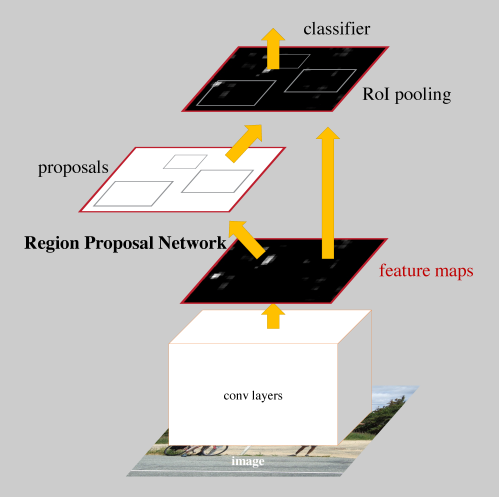



**Figure 5.3:** Faster R-CNN base and proposal network (taken from the paper by Ren et al. [11]

One approach to solve this problem is to use a CNN for generating region proposals too. This is done by the successor of Fast R-CNN, logically named Faster R-CNN [11], in the following referred to as FRCNN. FRCNN is split into two stages. The first stage is a network without any fully connected layers, just a set of convolutional filters that can be run on the image input in original dimensions that produce an output of feature maps of which are not fixed in size. Region proposals (where could be objects in the image?) are generated by feeding this output into a second network, called the Region Proposal Network in a sliding window fashion. The region proposal network is trained on anchors, a fixed set of boxes with different scales and aspect ratios, and outputs for every anchor an estimation how likely this anchor hit an object or background and how the anchor bounding box should be changed to include the whole object. This information is used to extract Regions of Interest (ROIs) from the output of the first stage network. These regions are resized and fed into the second stage, a network used solely for object recognition [see fig.5.3].

It should be noted that the concept of anchors is FRCNNs replacement of the image pyramids or filter pyramids employed by other algorithms to detect objects at different sizes. More recent research reintroduced this to FRCNN in a computationally less expensive way, called Filter Pyramid Networks (FPN) [14], discussed in the section about detector improvements (see 5.5).

To train FRCNN the first stage network and the classifier are created by splitting an image recognition network[1]. The region proposal network is then trained on the combination of first stage and second stage. Afterward, the proposal stage is fixed and only the classifier is trained on first stage and proposal network. This makes training of FRCNNs a difficult task.

Another approach in contrast to FRCNNs two-stage architecture is handling bounding box regression and classification in a single network:

Single Shot Detectors (SSD) by Liu et al. [15] generate for each image a total of 8732 pairs of class score and bounding box coordinates. These outputs are directly derived from the feature maps of different network layers without intermediate object or region proposals [see fig.5.4]. This is realized by aggregating pixels on

1 It is common to take apart the architecture of an image recognition network to adapt this for detection. The network that is repurposed is called the base network.

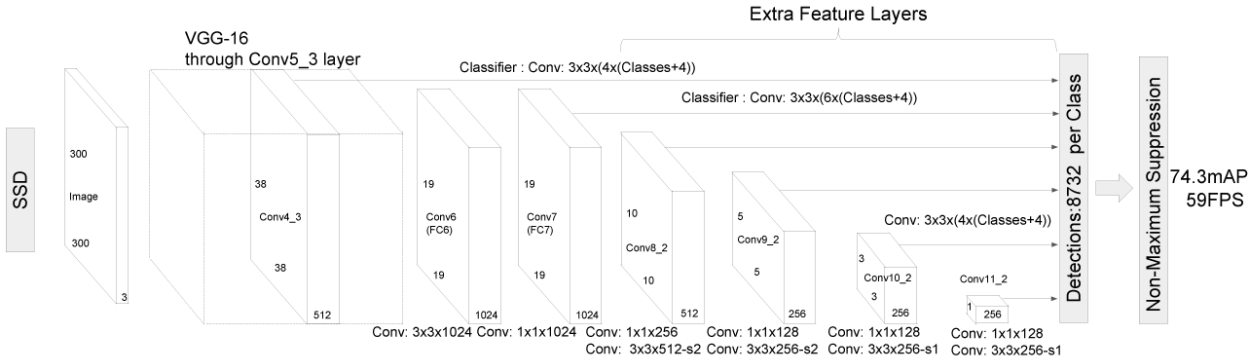



**Figure 5.4:** SSD (taken from the paper by Liu et al. [15])

feature maps of different sizes to boxes of different size and aspect ratio, similar to FRCNNs anchors. For each box, a category score and a bounding box offset are outputted.

There are other architectures not mentioned previously like YOLO (You Only Look Once) [16]. YOLO and its successors are single stage detectors similar to SSD, but while SSD uses information from layers of all sizes for detections, YOLO relies only on the last and smallest layer. This is due to a optimization for inference speed, not for localization precision or detection of small objects. YOLO has not been evaluated for this work since there exists no off-the-shelf implementation for mobile phones.

### 5.3.1 Comparison

The most prominent advantage of FRCNN is that proposal generation is run on the input image in original dimensions without resizing. Relevant for detecting the presence of small objects are the anchor scales and not if a high-resolution input image has been resized. Additionally, the detections are translation invariant since the Region Proposal Network is used in a sliding window fashion on the base network output. An object is detected with the same reliability in the upper left corner and in the center of the image.

While FRCNN theoretically should perform well on small objects, this is a problem for SSD due to its fixed anchor positions. Small objects that are near to each other need to be recognized each by their own anchor box. In cases like an image of a flock of birds or a school of fish SSD will produce false negatives due to a saturation of all anchors in the image region.

SSDs have the advantage of a higher inference speed. Due to their architecture, they can be trained directly like recognition networks with the difference that the loss is a weighted sum of confidence and localization error. No alternating training and fixing of proposal or classifier stage is required as it is the case for FRCNN.

Variants of FRCNNs and SSDs are considered the state of the art at the time of writing and FRCNN/FPN based networks score best on the challenging COCO leaderboard[2].

2 http://cocodataset.org/#detection-leaderboard

## 5.4 Datasets

### 5.4.1 Existing datasets

Whenever machine learning algorithms are used, datasets are as relevant as the algorithms itself. When it comes to pedestrian detection there are datasets which are specifically created to train and evaluate detection algorithms for persons in urban scenes and general datasets for object recognition and detection of multiple classes. Relevant ones of the latter class are:

**ImageNet** [17], an general purpose dataset for object recognition, consisting of 14 million images annotated with 21841 terms. The corresponding challenge is the ImageNet Large Scale Visual Recognition Competition.

**PascalVOC 2007 - 2012** (VOC07-12) [18], the dataset for the Pascal Visual Objects Challenge. VOC supports multi class object detection and features 20 classes in ~10,000 images.

Microsofts **Common Objects in Context** (COCO) [19], is a dataset for object detection consisting of ~200,000 images with 1.5 Million objects labelled from 80 classes. These include "person", "bicycle", "car" and "truck".

Datasets which have been compiled to specifically detect pedestrians:

- Caltech Pedestrian Detection Benchmark [20]
- KITTI Vision Benchmark Suite [21]
- INRIA Person Dataset [22]
- ETHZ datasets [23]
- TUD-Brussels [24]
- Daimler [25]

All of the datasets above have in common that they are specifically created to facilitate pedestrian detection for autonomous driving. Image data are videos or frames from videos of dashcam-like cameras at a low resolution (in the case of Caltech 640x480 pixels), captured from a moving vehicle. The pedestrians in these images are large compared to the total image (60 pixels average height at Caltech) and sparse at ~1 person per image on average. None of these datasets is sufficiently similar to a static pedestrian detection setup mounted on an elevated viewpoint to be used for training or evaluation.

There are two (notable) other datasets:

The PEdesTrian Attribute (PETA) [26] dataset, consisting of 19,000 images of pedestrians from frontal and elevated viewpoints. However, PETA contains only cropped images, useful for pedestrian attribute recognition (such as carrying a backpack, male/female, etc.), but not for pedestrian detection.

The CityPersons dataset [27], features 5000 images with 35,000 bounding boxes from 27 cities. Image acquisition is similar to the Caltech dataset, a moving vehicle with a dashcam. While CityPersons is an improvement over Caltech and other automotive pedestrian datasets in terms of data diversity and size, it still suffers from the same problems as every other dashboard-recorded video frame sequence, discussed in the next chapter.

All of these pedestrian focused datasets have several disadvantages for our application.

### 5.4.2 Problems

The main problem of Caltech and KITTI is summarized by Zhang et al. [27]:

> "Both datasets were recorded by driving through large cities and provide annotated frames on video sequences. Despite the large number of frames, both datasets suffer from low-density. With an average of ~1 person per image, occlusions cases are severely under-represented. Another weakness of both dataset, is that each was recorded in a single city. Thus the diversity in pedestrian and background appearances is limited."

CityPersons improves in this points, but all datasets have several problems in common. On all datasets the image data is low resolution and features few or no elevated viewpoints. Additionally, most of them have no static camera on image sequences.

To the best of our knowledge, there is no comprehensive dataset of urban scenes that features high-resolution images or images from a static viewpoint from above. Hence, we have decided to evaluate the performance of our system on a self-compiled minimal set of test data.

### 5.4.3 Own Dataset

**Figure 5.5:** averaged images of all four scenes

Our dataset consists of 160 manually annotated images from 4 different scenes (40 per scene), featuring four observation positions with the preferred viewing angle of 30 to 60 degrees (see fig. 5.5). The 2378 person bounding boxes in the evaluation batch of this dataset have an average size of 83x193 pixel and a non-standardized average aspect ratio of 1:2.33 [see fig. 5.6]. While the Caltech pedestrian bounding boxes have an average height of 60 pixels, our objects can be considered small in relation to the overall resolution

of our images (11 to 20 megapixels) [3] In addition to *person* the classes *car*, *truck*, *bus* and *bicycle*, although only the class person will be used for the evaluation.

Ground truth annotation of this dataset has been done manually using labelImg [28]. Bounding boxes are covering the whole extent of the object without padding, even occluded areas [4].

Caltech and other datasets included the concept of 'ignore regions' such as extremely crowded parts of an image. In these regions no detections of the algorithm are rated as FP or FN. Our evaluation dataset contains these annotations, but they are currently ignored. Thus, performance is slightly underreported.

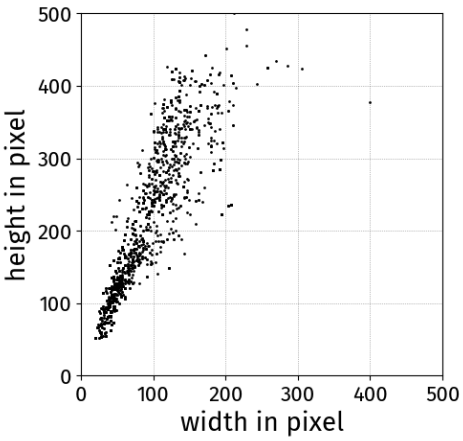



**Figure 5.6:** person bounding box size

The size of this dataset is small with a number of only 3495 bounding boxes in total (2378 in the class person). This limits the ability to generalize from this evaluationset but even this small dataset gives a better estimation of the overall performance than any of the previously mentioned datasets despite their vastly larger size.
To improve our dataset for evaluation purposes it would be more important to add additional scenes, rather than increasing the number of annotated images per scene.

## 5.5 Object detection for a mobile device

To decide which architecture is most suitable for our detector on a mobile system several requirements should be stated:

Precision and **speed** are always a tradeoff. For the task of recording pedestrians, paths precision is a higher priority than speed, for counting pedestrians it is vice versa. For both modes is true that faster networks reduce the battery consumption and increase the total observation time.

**Upper limit for memory consumption**. On many smartphones processing power is not the limiting factor but rather the memory transfer speed and memory capacity. A big network may perform well on modern phones with 4 or 2 Gigabytes of RAM but will generate Out-Of-Memory errors on older devices with less memory.

3 This is relevant in regard to the architecture of the neural network used for detection. SSDs require resizing of the whole input image so the original resolution of an object becomes less important. The first stage of Faster R-CNN operates on the original image data and benefits from a high-resolution input.
4 as stated by Zhang et al [29] the learning process benefits from this setting and accordingly the COCO bounding boxes have been set for our pretrained network

Pedestrians are rather **small objects** in images of public spaces. The network needs to detect objects at a lower scale reliably. This is our main problem which needs to be addressed.

To test several networks on Android we use the tensorflow Mobile Framework [30] in combination with the tensorflow Object Detection API [31]. This allows us to rely on the extensive collection of pretrained networks in the tensorflow model zoo. These networks have been trained on COCO and allow to detect objects from the 80 classes of this dataset without further transfer learning.

While we rely on the tensorflow Mobile Framework to run networks on Android phones there is the more advanced option of the tensorflow Lite Framework. The Lite Framework supports Androids Neural Networks API (available at Android 8.1 and higher) which promises to accelerate certain common operations for neural networks and quantized layers (faster network layers with integer weights in contrast to floating point weights). While certain frameworks may benefit from this in the future, at the time of writing this is still experimental and important operations required for common object detection network architectures have not been implemented yet.

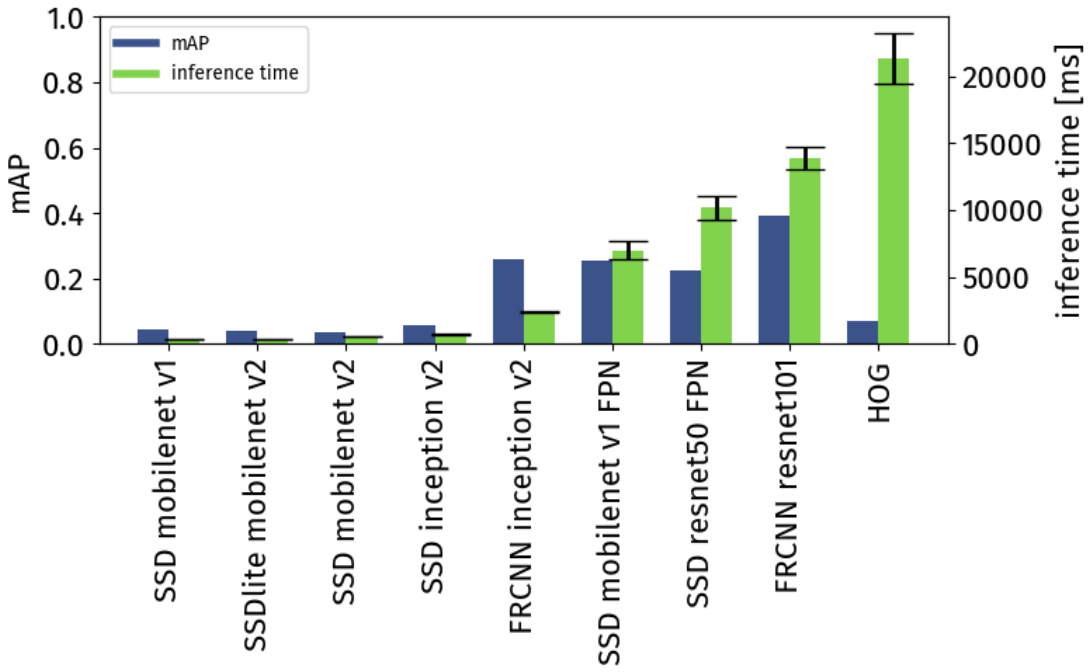



**Figure 5.7:** Comparison of detection algorithms on the full image (higher is better for precision (measured in mAP) and lower is better for inference time)

We compare HOG and several SSD and FRCNN models with different base networks (see fig.5.7). Since our evaluation dataset is sparse in COCO classes such as *bicycle* or *car*, we consider only the class *person*. The left bar (blue) represents average precision as mAP (higher is better, the metric is explained in detail in the evaluation section6.1.1). The right bar (green) represents inference time, the complete processing time of the network from input to output without initialization time (loading the model happens only once).

Time measurements were done a ZTE Axon 7, a mid-level priced smartphone. The networks will run faster or slower on different devices but will have about the same runtime relative to each other.

The different networks score as expected with large differences. In terms of runtime compared to detection precision, HOG performs worst by a large margin and will be discarded for further evaluations. In terms of speed, mobilenets in both versions [32] [33] as a base for SSD outperform the larger FRCNN networks.

The lower performance of SSD-based models compared to FRCNN networks can be explained by their fixed input. The SSD FPN models require rescaling to 640 pixels, all other SSD models to 300 pixels before processing. Although FRCNN networks perform better, they have a higher memory consumption and are slightly slower. The biggest network, the FRCNN NAS variant requires about 4.5 gigabytes of memory and is unsuitable for use on a mobile device. The precision of all models, especially the mobilenet SSD model, are without further improvements too low to detect pedestrians in our high-resolution images.

There are several approaches to improve the performance of both architectures for small objects:

Liliang Zhang et al. [**?**] investigate FRCNNs Region Proposal Network and the Fast R-CNN classifier. They identify a too high step size when applying the convolution filters on high-resolution layers as the main problem as well as too small features maps on the classifier. They propose increasing the receptive field by convolution with only every n-th pixel (called á trous convolution) instead of applying filters with a high stride.

Shanshan Zhang et al. [27] investigated several changes to FRCNN in order to improve pedestrian detection specifically on the Caltech dataset. They propose up-scaling of the input image (Caltech is only 640 by 480 pixels), quantized scales for the Region Proposal Network and different training techniques (using Adam instead of Stochastic Gradient Descent) among others. Results show that up-scaling the image has the biggest impact on precision while using smaller step sizes for the filters make the least difference.

Eggert et al. [34] propose a different anchor generation scheme to create anchor boxes at different scales to offer a better overlap with small objects.

One approach to improve the precision at small scales at SSDs is Deconvolution. Fu et al. [35] propose to add after the last and smallest layer new layers with increasing size. This hourglass-shape of a network is done by deconvolution operations. Detections are made based on the deconvoluted layers of increasing size instead of the convolutional layers of decreasing size.

Cao et al. aim to improve the deconvolutional approach by their Feature-Fused SSD [36], introducing summing operations to fuse layers of different size in order to improve precision on small objects.

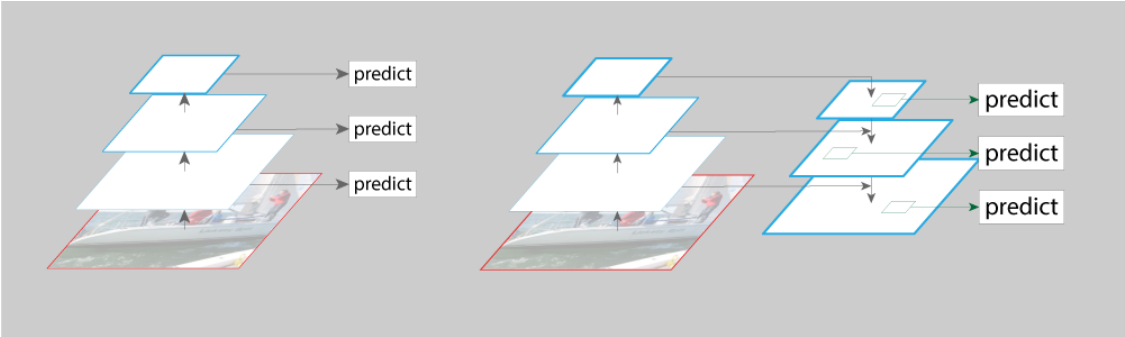



**Figure 5.8:** left: SSD architecture, right: FPN (figure taken from the FPN paper by Liu et al. [14])

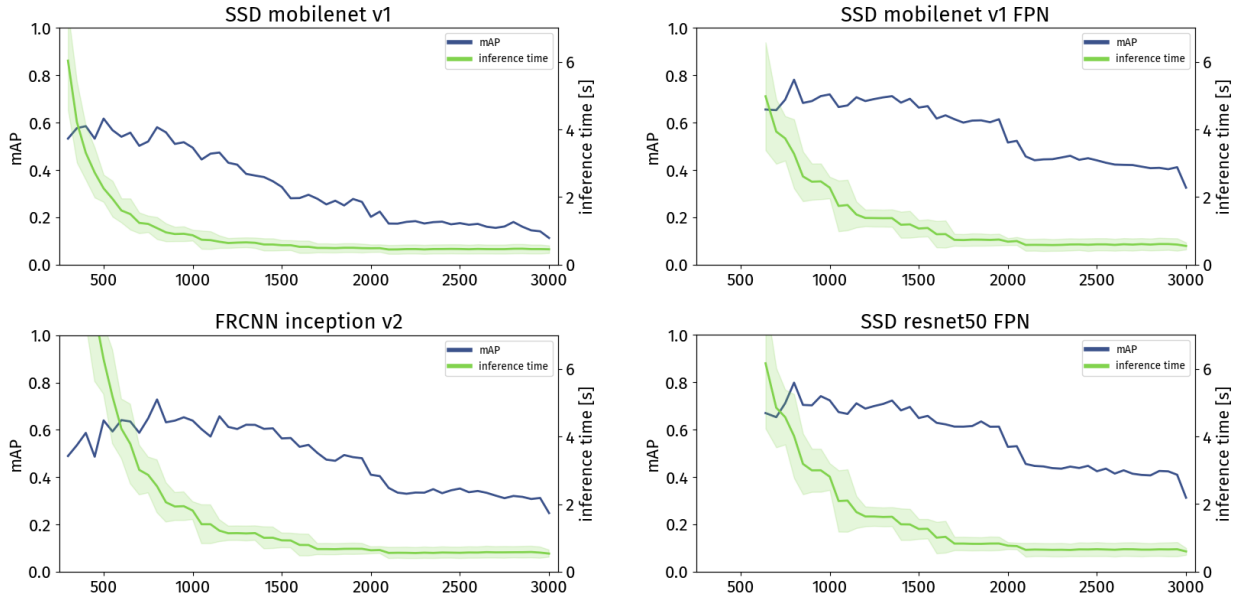
Liu et al. propose Feature Pyramid Networks (FPN) [14] to improve the networks ability to detect objects at multiple scales. One problem with a classical CNN architecture is that the first layers have a high-resolution which is necessary to detect small objects but contain little semantic information (for example only edges). The last layers contain more semantic information (or high-level features) but have a lower resolution. While SSD uses these early, high-resolution layers directly for detections, their low semantic information density may be obstructing in generating high precision detections. The concept of FPNs is to enrich the high-resolution layers by passing semantic information back to the early layers [see fig.5.8]. This is done by upsampling the last layers output layer by layer and merging this with each layers feature map with 1x1 convolutions. Based on this resulting feature pyramid the detections are generated.

To choose the best performing approach for the task of detection small pedestrians in high-resolution images several points must be taken into consideration:

Training an own model that performs well on larger input sizes and uses deconvolution would exceed the scope of this thesis (training a larger SSD model is planned as a future work). Upscaling the whole image as proposed is not an option on a mobile platform, as this increases memory consumption and is only suitable for FRCNN networks. However, an effect similar to upscaling without the same memory requirements can be achieved by applying detection networks in a sliding window fashion.

Our approach is to split the high-resolution input image in non-overlapping tiles of constant size and let the network detect objects in every tile separately.

To evaluate if this is a viable option, a benchmark is required:




**(a)** Precision of various network in regard to tile sizes. Inference time was measured across the whole evaluation set while the network ran on a GPU (Nvidia GeForce GTX1050Ti). The FPN models have a larger input size and are evaluated starting at 640 pixels.

When comparing different sizes for non-overlapping tiles, most networks precision begins to decrease above a certain pixel size. The mobilenet-based SSD has a fixed input size of 300 pixels and shows the best tradeoff between accuracy and speed at image tiles about twice its input size [see fig.5.9a]. If FPNs are used (at 640 pixels input size), the image tiles can be increased to three times their input size before the models begin to show a strong decrease in precision. This shows that architectures with FPNs have an advantage for detecting small objects in our evaluation dataset, while differences between the base networks (resnet50 or mobilenets) can be neglected for the SSD FPN architecture.

However, when splitting the image into tiles several problems are introduced:

- **Time:** the detection time is increased by the number of tiles. So even when networks with a higher inference time are used, the total runtime for all image times may be lower if fewer tiles are required for the same precision.
- **Maximum size:** the network can detect no objects larger than a tile. This is a problem in cases where a shallow viewing angle has been chosen and it is necessary to detect big objects in the foreground and tiny objects at the far end of the image.
- **Artifacts:** detection at the edges of tiles will introduce errors. When objects cross tile borders they may not be recognized at all or multiple times if the network performs well. This will result in artifacts in a scatterplot of the detection data (see the evaluation chapter for more information about this 6.1.1)
- **Reduced field of view:** when a fixed tile size is used, the remaining area around image borders is not fed into the network. All ground truth boxes in these areas are counted as a false negative.

The problem of missing or double detections at tile borders could be addressed by adopting a sliding window with overlapping regions. This would require a strategy to deal with double detections, such as applying Non-Maximum Suppression again (that is already a part of SSD). However, this would ultimately result in a complete reimplementation of sliding window techniques used for algorithms such as HOG. We assume changing network architectures to support larger input sizes is the more reasonable approach.

For the time being, however, we adopt tiling and resizing of the image without overlapping regions. While FRCNN outperforms SSD variants in terms of precision, even small FRCNN networks have a higher memory consumption than every SSD model. When choosing SSD models, we recognize that FPN networks have a clear advantage over standard variants. The two FPN networks, SSD with mobilenets v1 and resnet50 as base networks perform at the same level while the resnet network has a slightly higher inference time and binary size. While FPN SSDs perform well on larger tiles than regular SSDs, their inference time is considerably higher, especially on mobile devices. Due to this, we settle on the SSD mobilenet v1 network as our default choice as the object detection network for our app. Since this network offers the highest speed at the expense of precision, we call this the "Low-Fidelity" option. Phones that offer sufficient computing power can optionally use the FRCNN inception v2 model (the Mid-Fidelity choice) or the SSD mobilenet v1 FPN network (High-Fidelity) for increased precision.

## 5.6 Improvements on the despat detector

While the pretrained SSD mobilenet FPN network from the tensorflow model zoo performs well when used with tiling, there are several ways to improve the detector.

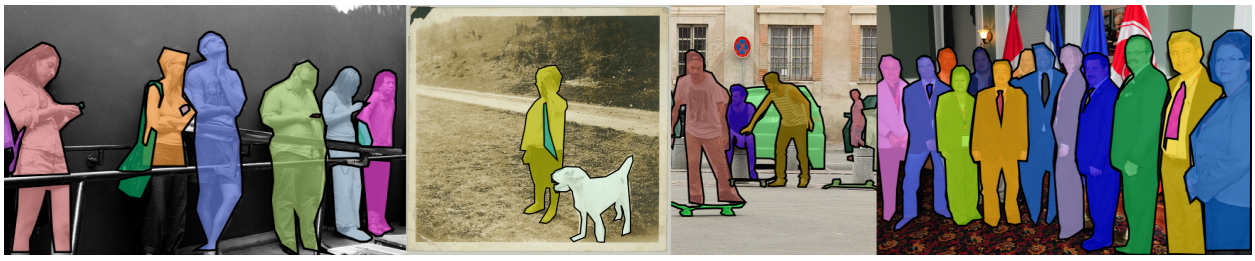


**Figure 5.9:** examples from COCOs person class (with segmentation masks)

### Transfer Learning

The model is using initialized weights from the ImageNet image recognition dataset and has been pretrained on COCO and its 80 classes. The main differences of COCO compared to our dataset is image resolution, object size and that the class person shows predominantly a frontal view [see fig.5.9]. The model is able to generalize, but we assume that the network precision could be increased by transfer learning. If only a subset of the 80 classes are required (person, bicycle, motorcycle, car, truck) and small instances of persons from

30-90 degree angle are used, the network should perform better for our use case. Before we test this, we plan to increase the size of our dataset, but this exceeds the scope of this thesis.

## Background Subtraction

The detector treats every image independently from previous detections. These could be used in post-processing to reduce the number of false positives. Another approach is to improve detector precision by background subtraction. First tests have shown that we are able to reduce the number of false positives by 10 percent if using OpenCV's implementation of the Mixture of Gaussian (MOG) algorithm on a downscaled version of the input image. If an object is detected where no detections have been registered in previous images and no change of background is reported by the background subtraction algorithm, this detection can be penalized and its confidence reduced. This has not yet been integrated into our app but is part of the future work on this project.

## Ignore Regions

Not all parts of an image may contain pedestrians. When starting a new session, the user could mark ignore regions such as sky or buildings. When the image is subdivided into tiles, all tiles that are part of an ignore region can be skipped during detection. This would reduce processing time as well as reducing false positives from reflections in mirror facades or people visible through windows.

Additional improvements are discussed with other future work in the corresponding chapter.

# 6 EVALUATION

## 6.1 Evaluation metrics

Evaluation of this work should be two-fold, we will discuss the precision and speed of the detector network as well as visualization results. While we did conduct interviews with potential users to assess other issues, a workshop for usability improvements is part of the future work.

Before precision and speed can be measured the corresponding metrics need to be discussed.

### 6.1.1 Precision

The detector returns a bounding box (BB) and a score or probability per class for each detection. To assess the performance of the detection network, detection bounding boxes need to be mapped to ground truth bounding boxes and a metric is required to evaluate the ratio of resulting true positives, false positives and false negatives (TP, FP, and FN). The ground truth bounding boxes in our evaluation dataset have been set manually, according to best practices, as described in the dataset chapter (see 5.4.3).

The prevalent method for mapping bounding boxes is Intersection over Union (IoU): the pixel area of the intersection of two bounding boxes is divided by the pixel area of their union (see fig. 6.1). A BB is rated as a true positive match if it the area of overlap with a previously unmatched ground truth BB of the same class exceeds 50 percent. In recent research (COCO) not a single threshold value is used but a range of overlap areas between 50 and 95 percent in 5 percent steps. Results are calculated for every single threshold and averaged afterwards. This altered metric is more sensitive to tiny localization errors of networks and favours large objects, which is less relevant for pedestrian detection and thus we decided to use the 0.5 threshold in this evaluation.

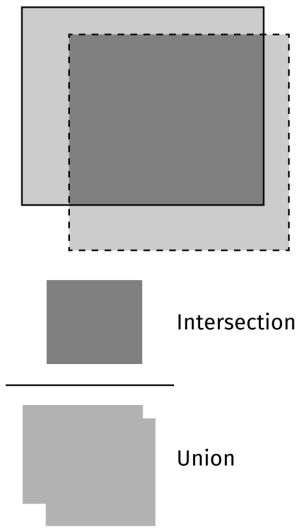



**Figure 6.1:** Intersection Over Union

Commonly used metrics to evaluate the ratio of TP, FP and FN are the ratio of False Positives Per Image (FPPI) to Miss-Rate (MR) and the Mean Average Precision (mAP). First, precision and recall are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

For the VOC 2007 - 2010 evaluation mAP is defined as the interpolated averaged maximum precision at 11 uniformly-spaced recall thresholds:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1.0\}} p_{inter}(r) \tag{6.3}$$

$$p_{inter}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{6.4}$$

This averaged precision AP of every class is averaged again. While VOC uses the abbreviation AP, COCO and other literature refers to the averaged precision over all classes as the mean average precision (mAP or map).

COCO and VOC 2012 calculate the mAP value in a slightly different fashion as the area under the precision-recall-curve. Everingham et al. [37] state as the reason for this change that "this interpolation was [...] too crude to discriminate between the methods at low AP". The mAP curve is derived from calculating the precision and recall values for every single bounding box (and all prior), ordered by a decreasing confidence threshold.

**mean average precision in plain english**
We look at the bounding boxes with the highest confidence score first and add one after another to our list. For all bounding boxes in our list we calculate precision and recall. Usually precision will be high because the detections with the highest confidence are usually right, but recall is low because just a few of all true positives have been found yet. As the confidence score is lowered and more detections are evaluated, the number of false positives will be higher and precision drops, but the amount of found true positives will be higher as well and thus recall increases.

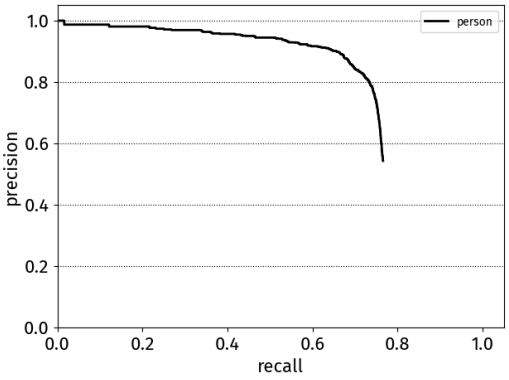



**Figure 6.2:** Example curve

This metric has the advantage that the overall detection performance of a network can be described with a single number.

The alternative to mAP is the ratio of False Positives Per Image to Miss-Rate. The caltech pedestrian benchmark, for instance, is evaluated using FPPI/MR as the absolute number of FN is relevant in autonomous driving. If the algorithm is missing a pedestrian, it is of minor importance if the person was alone or in a crowded scene. While FPPI/MR gives a total number of false positives, mAP values are always in relation to the total number of objects. When it comes to pedestrian tracking for urban planning, evaluating on the basis of mAP is reasonable, since it is acceptable and expected that the algorithms precision will drop when running on crowded scenes.

To evaluate the performance of our detection network we rely on the COCO mAP (interpolation of the curve for every detection, not just at 11 points), but define 0.5 as our only overlap threshold for bounding box

matching. Thus our metric is tolerant against small localization errors which may even increase due to tiling, but still have a fine-grained mAP value which is necessary to differentiate between the precision of different networks for different tile size settings.

### 6.1.2 Localization

We will not evaluate localization errors in the evaluation of this work. As long as an IoU of 0.5 can be achieved, detections are counted as valid and the difference to a full overlap is small. It can be assumed that the error introduced in post-processing by incorrect point selection for the homography transformation outweighs the small localization errors of single bounding boxes. However, the major problem about the homography error is on the human side of things. The resolution of map material is crucial and the worse the image quality, the harder to find precise point correspondences between image and map. We use GoogleMaps as the provider of satellite maps which offers higher resolution image material than any other global provider, but still the resolution is barely enough. This is caused in part by US Law[1] which limits the maximum resolution of satellite images made available to the public. In the future work chapter we discuss approaches to allow users to make more precise point selections to reduce the homography transformation error.

### 6.1.3 Speed

The runtime of the network and additional pre- and postprocessing operations is important for the app in terms of temporal resolution and battery life. The smaller the required time for processing a single image is, the more datapoints can be gathered or the earlier the device can return to battery saving sleep mode.

When evaluating inference time measurements it should be noted that all computation was done on the CPU of the smartphone as no GPU support was available at the time of writing. This may change in the near future and it can be assumed that some phones in the mid to high-cost section will improve their performance with Androids Neural Networks Interface.

## 6.2 Performance of the detector

When discussing the mean average precision of our detector it is necessary to set this in relation to other alternatives. The detector, running on limited hardware in terms of processing power, memory and battery is compared to a reference network (the best-performing CNN executed on a GPU) and to ground truth. Ground truth is compiled by a human without time constraints and access to the whole image and the complete image series. When comparing neural networks to human precision usually the playing field is leveled and the human has to perform under similar constraints as the algorithm. This can be realized by presenting only crops of region proposals for classification to the human and setting a timer for any action. As an example

1 The Land Remote Sensing Policy Act `https://www.congress.gov/bill/102nd-congress/house-bill/6133`

for the sole task of classification, Russakovsky et al. [38] state that on their ImageNet dataset the human error is about 5.1 percent.

The low-fidelity network (SSD mobilenet v1) is compared to the mid-fidelity one (FRCNN inception v2) and the high-fidelity network (SSD mobilenet v1 FPN). Our reference network is the best-performing available network from the tensorflow model zoo, trained on COCO and executed on a GPU (FRCNN NAS with dynamically sized tiles at 1/6 image size). The ground truth referenced here is the unaltered ground truth from the evaluation dataset and is not a human performance estimation.

The phones used as examples have been chosen from all price ranges. The Motorola Z1 features a 13 Megapixel camera and is priced around 500 Euro, representing the high-end device class. Mid-range phones are represented by the ZTE Axon 7, available at about 200 to 250 Euro. Low-end phones such as the Motorola 5GS are available for 150 Euro and extremly inexpensive phones (named the ultra-low-end class) are even cheaper at about 80 Euro.

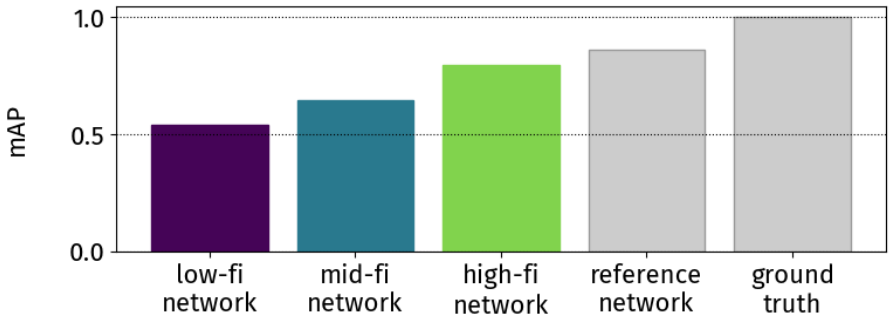



**Figure 6.3:** Mean average precision of our three networks in comparison to the best performing reference network and ground truth

| Network | SSD mobilenet v1 @ 600px (low-fi) | FRCNN inception v2 @ 800px (mid-fi) | SSD mobilenet v1 FPN @ 800px (high-fi) | FRCNN NAS @ 1/6 (reference) |
|---|---|---|---|---|
| mAP (person) | 0.54 | 0.65 | 0.80 | 0.86 |

**Table 6.1:** mean average precision values of our three mobile networks, compared to the reference network

On our evaluation dataset the low-fi network achieves a mAP of 0.54 which is not extraordinary (this is discussed in more detail in the next section). However, the mid-fi and high-fi networks achieve an average precision of 0.65 and 0.80 respectively, which compares very well to the reference network, reaching 0.86 points. An average precision of 0.80 on our very challenging dataset, without any additional postprocessing correction, can be noted positively. In addition to that it should be stated that the reference network runs about 30 times slower than our high-fi network (when both are executed on a GPU).

The lower precision of the low-fidelity network can be explained when considering the time-precision tradeoff and comparing it to the other networks. It runs at 3.5 seconds at the high-end and 5.5s at the mid-range phone. This allows a capture rate of 5 to 6 images per minute without exceeding the safe range of fewer than 50 degrees centigrade on the phone. The performance on the low-end phone is worse for the whole image at 13.3s than 4.6 on the ultra-low-end phone, while the processing speed per tile is about equal at

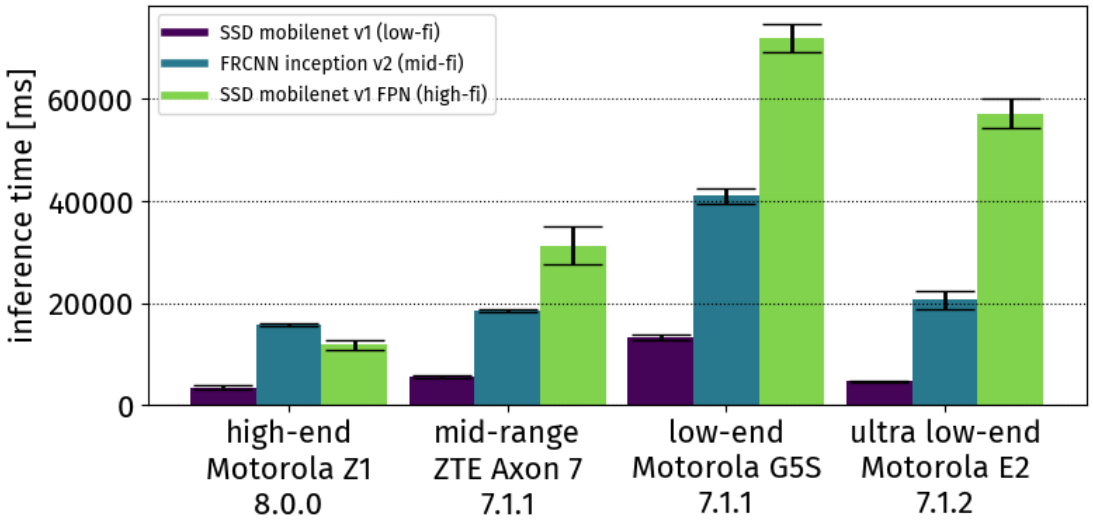



**Figure 6.4:** Runtime per image of the networks on different phone types (error bar is standard deviation)

| Phone | Category | Resolution (Megapixel) | Low-Fi Time Tile [ms] | Low-Fi Time Image [ms] | Mid-Fi Time Tile [ms] | Mid-Fi Time Image [ms] | High-Fi Time Tile [ms] | High-Fi Time Image [ms] |
|---|---|---|---|---|---|---|---|---|
| Moto Z1 | High-End | 13 | 228 | **3426** | 2624 | **15744** | 5879 | **11759** |
| ZTE Axon 7 | Mid-Range | 20 | 197 | **5516** | 2302 | **18423** | 5217 | **31303** |
| Moto G5s | Low-End | 16 | 665 | **13312** | 6832 | **40994** | 17988 | **71955** |
| Moto E2 | Ultra-Low-End | 5 | 711 | **4627** | 10331 | **20662** | 57170 | **57170** |

**Table 6.2:** Runtime per tile and per full image for four different smartphone categories

665ms mean and 771ms respectively. This is a result of a higher camera resolution on the low-end phone and needs to be taken into consideration. While some phones will be slower in detection, their maximum in precision is higher due to their advantage in resolution. However, a gain in resolution cannot always be directly translated to gains in precision, in part due to inexpensive optical systems, sensor size and resulting poor image quality.

The mid-fidelity networks inference time is 15.7 and 18.4 seconds on high and mid-range phones. Again the speed difference between those phones can be neglected for the task of detection. The marginal gain in inference speed cannot be utilized when the device overheats. Low and Ultra-Low-End phones finish processing after 40 and 20 seconds, setting the maximum capture interval to about one image per minute.

The high-fidelity network is actually faster on the high-end phone than the mid-fidelity model (due to fewer tiles). It takes 12 seconds, compared to 31 seconds on the mid-range device. Although it is possible to run the high-fidelity network on the low-end phones, they are prohibitively slow at about 62 and 58 seconds per image.

It should be noted that all these measurements were conducted with networks running at maximum resolution and are the upper boundary. If a phone is placed on a balcony surveying only a narrow street, distance from the camera to pedestrians is low and their size in the image is considerably higher. This allows to increase

the tile size and reduces the number of overall tiles, thus reducing runtime (however, these settings are not yet user-definable in our app).

## 6.3 Failure Analysis

When analyzing the ratio of FP to FN it becomes evident that the detector misses more pedestrians than it makes up or detects twice. Manual analysis shows that most FN can either be contributed to occlusion (especially pedestrians walking in groups) or small size. This can be improved by image tiling, which introduces several other problems on its own. One is an increased computational load. Another one is detecting objects at tile borders (see fig.6.5). If an object is cut by a tile, a well-performing network will detect the object twice and one bounding box will be evaluated as a false positive. A less well performing network may not detect an object at all on the border of a tile and a false negative is the result. In all cases the bounding box is trimmed at the tile border.

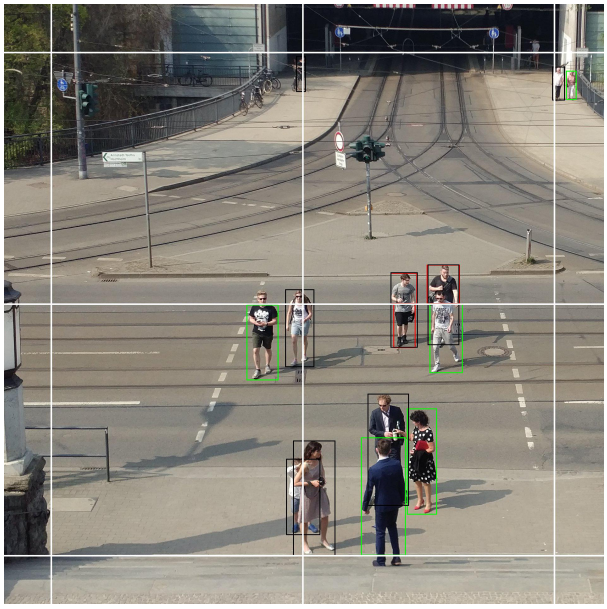


**Figure 6.5:** Common detection errors due to tiling (TP: green, FP: red, FN: black)

This may or may not be visible when evaluating actual visualized datasets. When comparing the Low-Fi networks output to a dataset generated by the Mid-Fi network, it becomes apparent that tiling artifacts are indeed visible in the visualization (see fig.6.6). However, this is only a problem at the small tile sizes of the Low-Fi network.

While in the image lines of hotspots are clearly visible, these lines correlate with curbstones and street borders. Obviously, people fan out when waiting for the traffic light to turn green. However, one of these lines is not visible in the visualization based on Mid-Fi network data. This line can be contributed solely to the effect of tile borders. Detections are cut above and below the border line in between tiles and thus artificially amplifies value in those bins. This effect is less pronounced on datasets gathered with a steeper viewing angle. Additional examples are part of the discussion in the next chapter.

### 6.3.1 Potential Problems in the Evaluation

There are three potential problems or improvements:

Our dataset contains **ignore regions**, bounding boxes describing areas in images where the detection network is expected to fail. This includes heavily occluded areas or streets meeting the horizon. In these

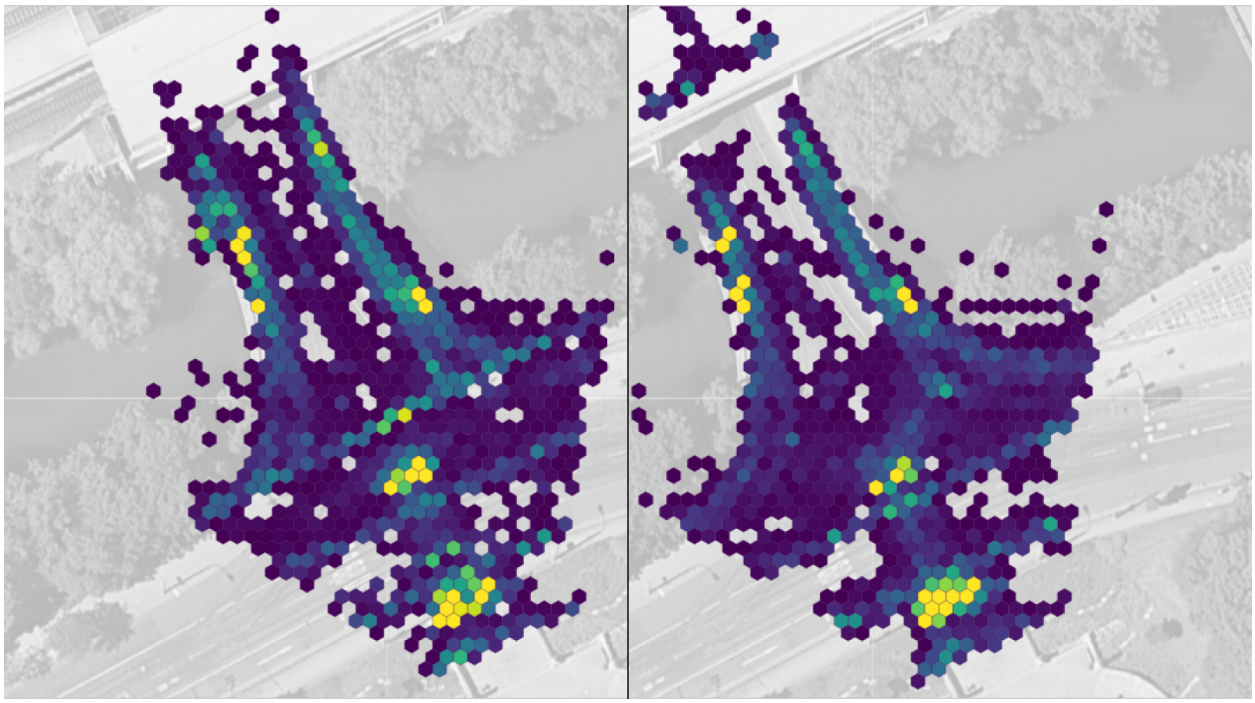


**Figure 6.6:** Visualization output of the Low-Fi network (left) compared to the Mid-Fi network (right) on the same data. [Image data: ©Google 2018, GeoBasis-DE/BKG ©2009]

regions detections of the network should not be treated as FP or FN. This is not done for this evaluation, thus the reported precisions are slightly lower than a more complex evaluation.

To get a better understanding of the networks performance in edge cases, **difficulty ratings** of bounding boxes are useful. The dataset lacks this and thus our evaluation too.

When calculating tiles for images, the tiling algorithm needs to deal with a multitude of different resolutions for different smartphone cameras. For this evaluation the tiling algorithm uses fixed sizes without padding which results in a **cropped image**. If an image has a size of 4200 pixels and 1000-pixel-wide tiles are used, 100 pixels are cut at each image border. This image data is not processed by the detector and all objects are marked as false negatives during evaluation. This results in a lower precision for higher tile sizes as these increase the ignored border space. This could be solved by adding a padding strategy (computationally expensive) or by increasing or decreasing tile sizes for a perfect fit (this increases noise when comparing tile sizes).

# 7 DISCUSSION

To recapitulate the initial goals for our automated survey tool: We did want to offer a pedestrian tracking and counting tool based on low-cost hardware. It should be easy to use without specialized technical knowledge, suitable for streets and public places.

Our weatherproof enclosure performs reliably and is sufficiently waterproof for any reasonable usecase. It is easy to build and versatile. The only downside is that we did need to add a non-standard part to the bill of materials and the seal requires manual cutting. Furthermore, enclosures must be specifically printed for every single phone and not a one-size-fits-all, although this is a reasonable design decision to avoid additional complexity and should not pose a problem.

When it comes to hardware, we did achieve compatibility with most Android smartphones. While not each and every old phone at hand will be compatible, nearly all modern phones will work and reasonably well-performing phones are available for less than 100 Euro. The main reason for incompatibility on ultra-low-end phones is low memory and insufficient software updates by the vendors, however, some of these phones can be upgraded with alternative versions of Android such as Lineage OS[1]. As stated in the software chapter, with a range of about 64 percent compatibility of all current Android devices (even without updates) and by offering networks with low performance requirements, we are confident that we have been able to satisfy this requirement.

We show that it is already feasible to run this kind of object detection locally on inexpensive mobile devices and this will perform even better in the very near future. Our detector reaches 0.54 to 0.80 in mean average precision, depending on network type and camera resolution. While this shows that at least our high-fi network is already well performing on our challenging evaluation set, we are confident to reach reliably average precisions values above 0.9 for scenes with low difficulty in the future. While we did plan to contribute a trained network for this specialized purpose, this was not possible owing to time constraints. A different prioritization and less time for the enclosure could have made this possible, but this is now a part of the future work.

Another important requirement is privacy. By running processing completely local on the device and only exporting averaged images we did not compromise on this. However, it needs to be stated that while we do not save and retain image data, we can not guarantee or communicate this to people passing by and seeing our devices. Nobody can know whats really happening on the device, it is after all – literally – a black box.

The last requirement: the usability of our app requires improvements before publishing it. While we did invest time and effort to offer a low barrier when starting to use the app, right now a short introduction is still required for new users. One of the main problems of our process of capturing is the lack of verifiability. When running object detection directly on image data one can always inspect single images as samples. We, however, retain no image data and the user can not check if certain parts of a scene did produce an overwhelmingly large number of false positives or false negatives. The only output is an overlay of all detected objects on the averaged image on the phone and the map in the visualization tool.

1 https://lineageos.org/

When discussing the actual usefulness of the data we gather, we have to emphasize its limitations. While our initial plan was to present a system that is already able to perform reliable counting too, we did not achieve this within the time of this thesis. We do gather only positions of objects, such as pedestrians, bicycles, cars or busses. The pedestrians are neither tracked across the scene nor counted. This gives an averaged map of positions of objects across the whole scene for a time window. Based on this data one can answer questions like which parts of streets and places are busy and which not. This can be done based on time, too. A place may be twice as busy during rush hour as during noon, and this can be shown with the gathered data. In addition to that, it is possible to answer where people are crossing the street or a space or how obstacles influence traffic. The usefulness of our datasets will increase further as soon as the action detection and counting modes have been added to our app (see the future work in the next chapter).

To conclude we will have a look at four examples of recorded datasets and discuss this results. Every dataset can be inspected with our visualization tool, running at a provided web address. All of this data is part of the digital addendum to this thesis, too.

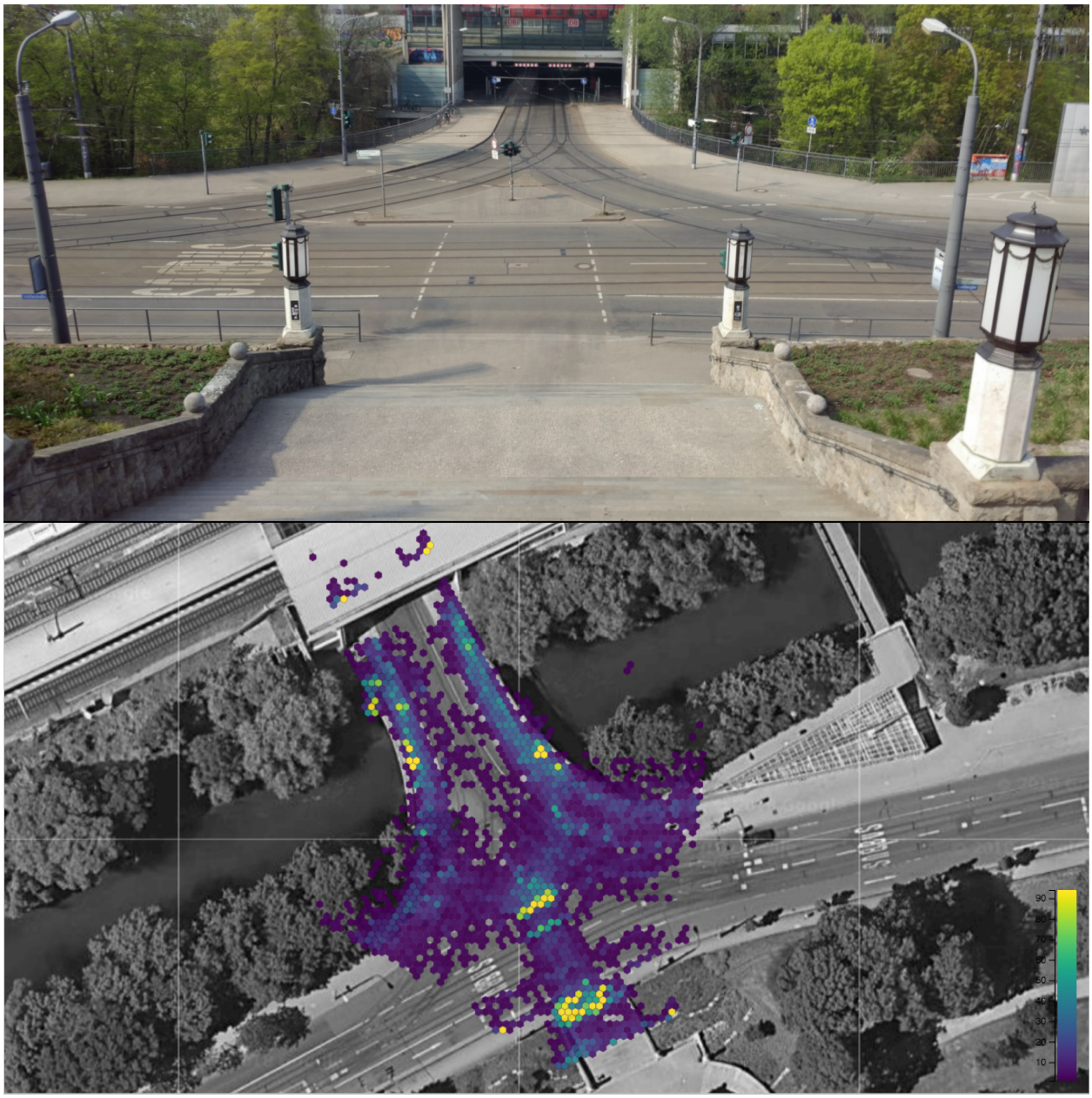



**Figure 7.1:** Erfurt [Image data: ©Google 2018, GeoBasis-DE/BKG ©2009]

This dataset from the rear entry to the main station of Erfurt, Germany is a good example of the amount of space a single camera can survey when placed well. It is clearly visible which routes people prefer to cross the street and if the visualization tool is used, one can even see the arrival time of trams and trains.

Visualization tool: `http://grinzold.de/map/#dataset_erfurt`

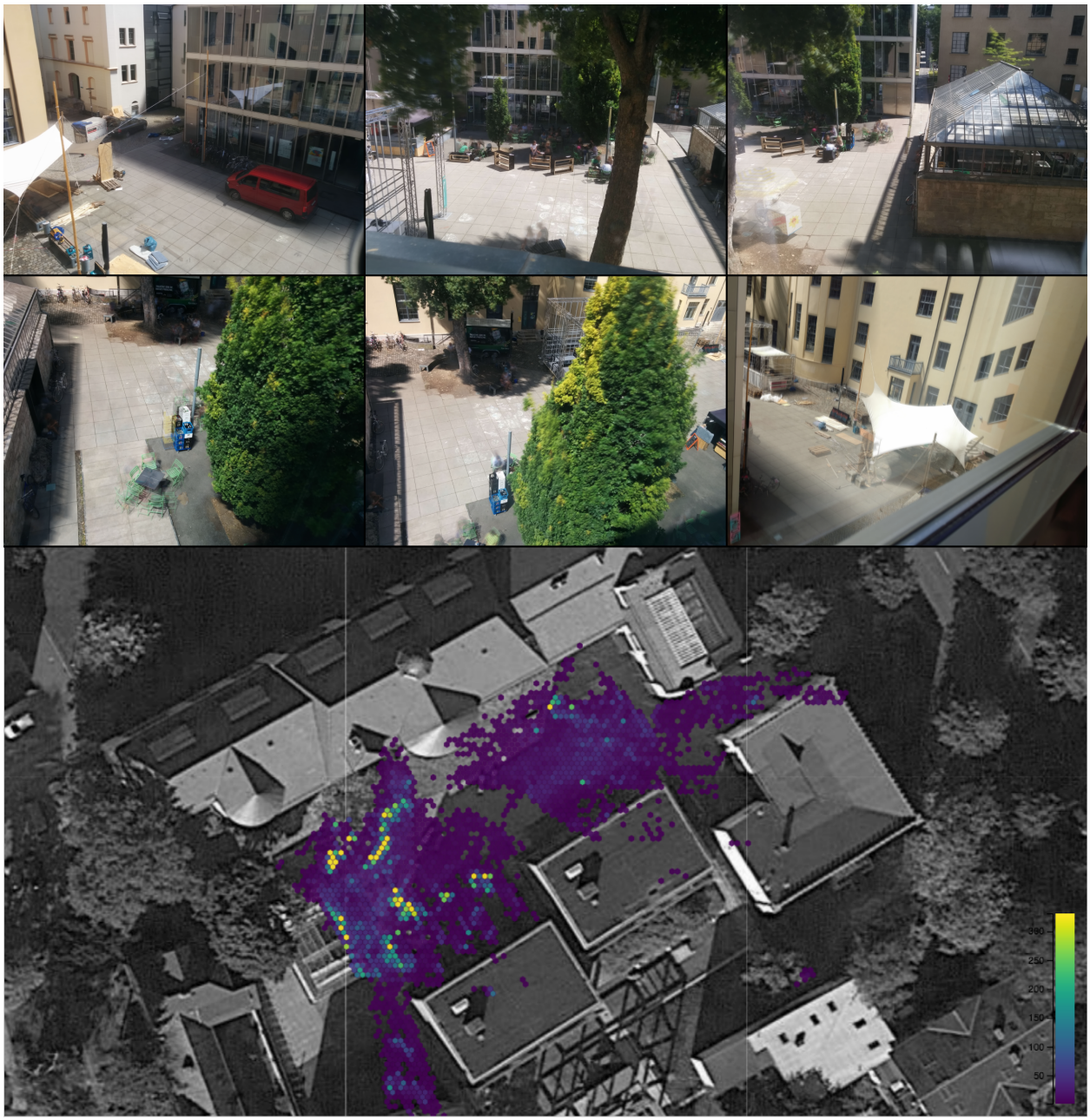



**Figure 7.2:** Summaery [Image data: ©Google 2018, GeoBasis-DE/BKG ©2009]

When using multiple phones in parallel a more complex geometry of a public space (or simply larger spaces) can be surveyed. However, the visualization tool is not yet capable of merging the overlapping areas, pedestrians that have been picked up by several cameras at once are simply added up. To cope with this problems, the visualization tool allows to show and hide individual capture sessions.

Visualization tool: `http://grinzold.de/map/#dataset_summaery`

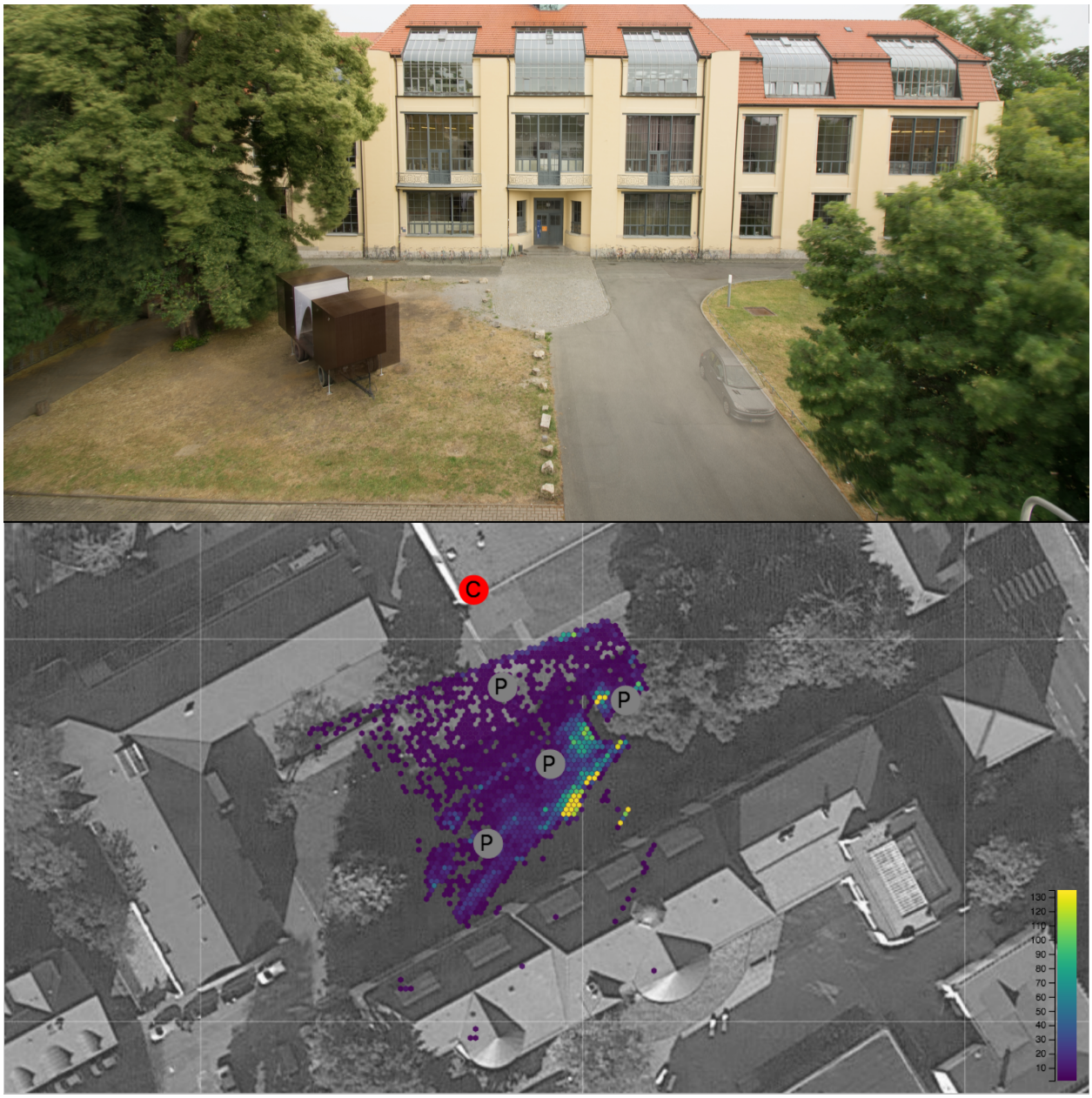



**Figure 7.3:** Summaery [Image data: ©Google 2018, GeoBasis-DE/BKG ©2009]

This dataset gathered at the main building of the Bauhaus-University Weimar is an excellent example of a erroneus homography transformation due to badly selected point correspondences. When comparing with the averaged image, it is clearly visible that the line of pedestrians is skewed.

Visualization tool: `http://grinzold.de/map/#dataset_campusoffice`

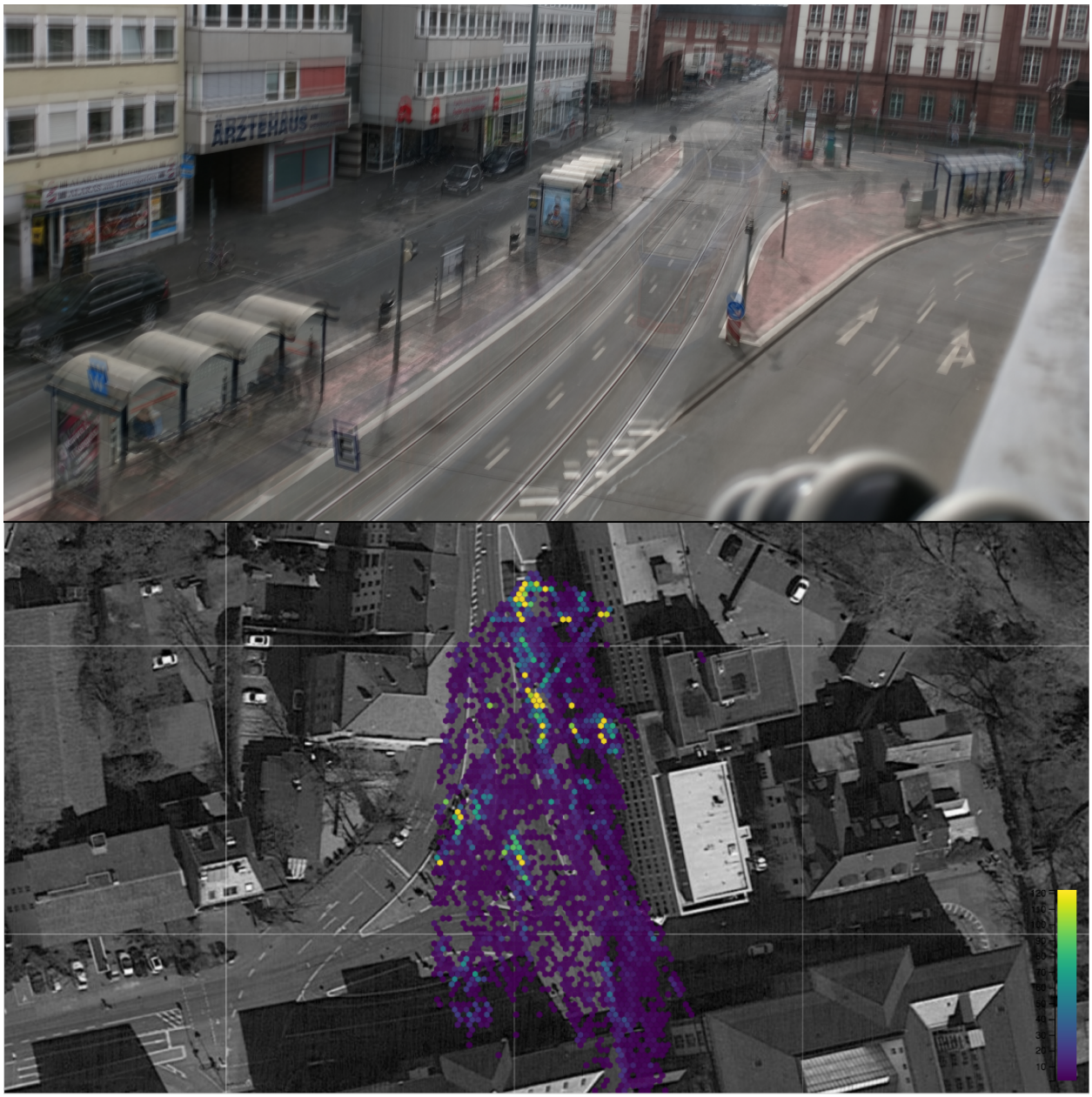
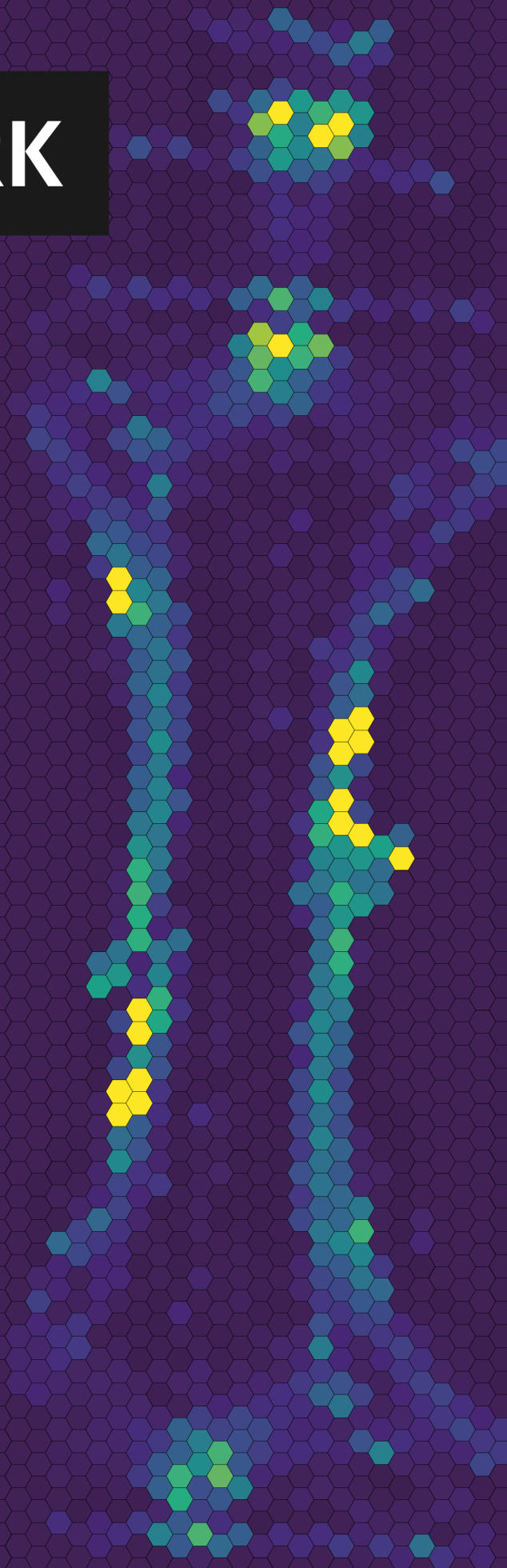



**Figure 7.4:** Darmstadt [Image data: ©Google 2018, GeoBasis-DE/BKG ©2009]

Another common problem during recording are shaky mounts. When the phone moves during capturing the shifted camera is clearly visible in the averaged scene image. This results in noisy data of low quality.

Visualization tool: `http://grinzold.de/map/#dataset_darmstadt`

# 8 FUTURE WORK



There are several paths we like to explore in the future:

## 8.1 Tracking & Counting

We want to introduce action labels for detected objects (static and moving) on the base of inter-image comparison. In addition to that, this is one of the fundaments for calculating paths which are need for reliable counting. Another benefit is that this makes it possible to calculate additional information such as movement speeds.

At the time of writing only general purpose detection of objects is possible. To be a viable alternative to manual urban surveys from people with clipboards the app needs to be able to acquire exact numbers of passing pedestrians. We will investigate if this can be reliably done for small regions of the image (e.g. zebra crossings, street corners, etc) with higher framerate data than the detection mode. In a second step other object classes than pedestrians can be counted, such as fast-moving bicycles or cars.

## 8.2 General detection accuracy & performance

The object detector network is an SSD FPN mobilenet (640 by 640 pixels input size) pretrainted on the Microsoft COCO dataset and its 80 object classes. Furthermore, there are changes to the detection network required to improve inference speed on the phone. Any reduction of processing time of the image batches directly impacts battery life, thermal stress and increases deployment time. Retraining the CNN with quantized weights and pruning will be tested to achieve this. A more mature mobile neural network framework such as tensorflow lite promises additional performance gains. While operations commonly used in CNNs for object detection are not supported at the time of writing, future releases may add these features. There are several measures for improving the detection accuracy and the inference speed of the network:

### Transfer learning with new pedestrian data

COCO contains mostly front views of persons. It can be assumed that the detection accuracy for objects in our scenes (viewing angle) improves after additional learning with our more specific data. Before this can be done, our evaluation dataset needs to be extended to be used as a training set.

### Different network architectures

Most datasets have an heavy bias towards objects that occupy a large (about 0.25-0.5) portion of the image. In our usecase nearly all objects with the exception of busses are considered small objects (in relation to the full image size). It may be benefical to train an SSD or FRCNN network with smaller box/anchor scale sizes.

### Quantization

Learnable weights in neural networks are traditionally floating point numbers. If after training these weights are converted to integers, precision will decrease, but can be reduced by some retraining on integers. This process is called Quantization and may be an acceptable tradeoff considering integer weights decrease the computational load for inference and the binary size of the network. The Tensorflow Lite framework for mobile devices supports this and we will evaluate if this is fits our problem when we migrate to the new framework.

### Postprocessing with geometric information

Object size and distance information from the scene can be used to improve detection precision by removing false positives. The camera position and angle can be calculated from the corresponding points and thus properties of the detected bounding boxes derived. In certain cases it might be viable to dismiss false positives based on calculated bounding box height (if an pedestrian bounding box has a height of 2.5 metres, its confidence can be decreased). We will investigate which improvement in precision we can achieve with these additional criterions for confidence.

### Background detection

Since our viewpoint is static, we can take advantage of background subtraction easily to infer additional information from our scene. Changes in background can be used to validate detections and reduce False Positives. In addition to that a basic action detection of objects is possible and it can be guessed if the object is static or moved between captures.

### Action detection

Another approach is to use an additional network for action detection on certain object classes. This may give insight into whether people are walking, standing or sitting, just indiscriminantly passing by or lingering at a place.

## 8.3 Usability

At the time of writing a short introduction talk must be given to new users of the app. This needs to be improved. We will add a short tour upon first opening of the app and small explanations and tooltips throughout the different screens of the workflow.

In order to make the additional information such as a static or moving label accessible for evaluation, the visualization tool will need to be expanded. In addition to that, we plan to package it as an standalone application for easier installation.

Another problem which is based on usability issues is the homography error. It is hard to achieve the highest degree of precision when marking the corresponding points between image and map on the touchscreen of the smartphone. This is caused in part by the Google Maps interface and US Law, as discussed in section 6.1.2.

One approach would be to do point mapping on a desktop computer with a high precision pointing device (a mouse) or by providing better points for mapping. The best viewing angle for the smartphone camera to capture data may not always contain four landmark points which are easy to precisely map to an orthophoto of questionable resolution. One solution to this problem may be to take a wide-angle photo of the surveyed space from a different location and transform from map to wide-angle photo to averaged image of phone. Combining two homography transformations will increase the error, but this may be a better approach on difficult spots.

## 8.4 Workshop

While not part of this work, we want to do an workshop with users of our app and get feedback on needs and problems for future development.

However, while this is an extensive list of future work, the very first step we will take is adding the counting mode and publishing our documentation for fabrication and usage.

# Bibliography


[1] E. Goubet, J. Katz, and F. Porikli, "Pedestrian tracking using thermal infrared imaging," p. 62062C, May 2006.

[2] Miovision, "Miovision scout." Website. Retrieved August 27th, 2018 from `http://miovision.com/data-link/scout/`.

[3] Placemeter, "Placemeter." Website. Retrieved August 27th, 2018 from `http://www.placemeter.com/`.

[4] Modcam, "Modcam." Website. Retrieved August 27th, 2018 from `https://www.modcam.com/`.

[5] B. Groß, M. Kreutzer, and R. Reimann, "opendatacam." Website. Retrieved August 27th, 2018 from `https://opendatacam.moovellab.com/`.

[6] N. Smith and S. van der Walt, "A better default colormap for matplotlib." Youtube. `https://www.youtube.com/watch?v=xAoljeRJ3lU`.

[7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005.

[8] P. Dollar, Z. Tu, P. Perona, and S. Belongie, "Integral Channel Features," pp. 91.1–91.11, British Machine Vision Association, 2009.

[9] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Visual object detection with deformable part models," *Communications of the ACM*, vol. 56, p. 97, Sept. 2013.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv:1506.01497 [cs]*, June 2015. arXiv: 1506.01497.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv:1311.2524 [cs]*, Nov. 2013. arXiv: 1311.2524.

[13] R. Girshick, "Fast R-CNN," *arXiv:1504.08083 [cs]*, Apr. 2015. arXiv: 1504.08083.

[14] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," *arXiv:1612.03144 [cs]*, Dec. 2016. arXiv: 1612.03144.

[15] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," *arXiv:1512.02325 [cs]*, vol. 9905, pp. 21–37, 2016. arXiv: 1512.02325.

[16] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[17] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[19] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," *arXiv:1405.0312 [cs]*, May 2014. arXiv: 1405.0312.

[20] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: A Benchmark," p. 8.

[21] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[22] N. Dalal, "Inria dataset." Website. Retrieved August 27th, 2018 from `http://pascal.inrialpes.fr/data/human/`.

[23] A. Ess, B. Leibe, K. Schindler, , and L. van Gool, "A mobile vision system for robust multi-person tracking," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, IEEE Press, June 2008.

[24] C. Wojek, S. Walk, and B. Schiele, "Multi-Cue Onboard Pedestrian Detection," p. 8.

[25] M. Enzweiler and D. Gavrila, "Monocular Pedestrian Detection: Survey and Experiments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 2179–2195, Dec. 2009.

[26] Y. Deng, P. Luo, C. C. Loy, and X. Tang, "Pedestrian Attribute Recognition At Far Distance," 2014.

[27] S. Zhang, R. Benenson, and B. Schiele, "CityPersons: A Diverse Dataset for Pedestrian Detection," pp. 4457–4465, IEEE, July 2017.

[28] T. T. Lin, "labelimg." `https://github.com/tzutalin/labelImg/`, 2018.

[29] S. Zhang, R. Benenson, M. Omran, J. Hosang, and B. Schiele, "How Far are We from Solving Pedestrian Detection?," *arXiv:1602.01237 [cs]*, Feb. 2016. arXiv: 1602.01237.

[30] Google, "tensorflow mobile." `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android/`, 2018.

[31] Google, "tensorflow object detection api." `https://github.com/tensorflow/models/tree/master/research/object_detection`, 2018.

[32] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861 [cs]*, Apr. 2017. arXiv: 1704.04861.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation," *arXiv:1801.04381 [cs]*, Jan. 2018. arXiv: 1801.04381.

[34] C. Eggert, S. Brehm, A. Winschel, D. Zecha, and R. Lienhart, "A closer look: Small object detection in faster R-CNN," pp. 421–426, IEEE, July 2017.

[35] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "DSSD : Deconvolutional Single Shot Detector," *arXiv:1701.06659 [cs]*, Jan. 2017. arXiv: 1701.06659.

[36] G. Cao, X. Xie, W. Yang, Q. Liao, G. Shi, and J. Wu, "Feature-Fused SSD: Fast Detection for Small Objects," *arXiv:1709.05054 [cs]*, Sept. 2017. arXiv: 1709.05054.

[37] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.

[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *arXiv:1409.0575 [cs]*, Sept. 2014. arXiv: 1409.0575.

# Declaration of Authorship

I hereby declare that I have written this thesis without the use of documents and aids other than those stated in the references, that I have mentioned all sources used and that I have cited them correctly according to established academic citation rules, and that the topic or parts of it are not already the object of any work or examination of another study programme.

Date Christopher Getschmann