FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	People detection using IR camera on a drone for more effective rescue operations
Student:	Matej Glejtek
Supervisor:	Ing. Lukáš Brchl
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2021/22

Instructions

The thesis aims to design and implement an application that will detect people on images captured by an IR camera placed on a drone for more effective rescue operations. Implementation of the application should be done in Python language.

- Research existing solutions for people detection from thermograms.
- Study appropriate data collection methods (drone parameters, camera parameters, etc.).
- Design and describe the architecture of the application.
- Implement the application with the help of computer vision algorithms.
- Document the code.
- Test the application on real data.
- Evaluate the resulting application and suggest its future extension.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 18, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

People detection using IR camera on a drone for more effective rescue operations

Matej Glejtek

Department of Software Engineering
Supervisor: Ing. Lukáš Brchl

June 4, 2020

Acknowledgements

In the first place, I would like to thank my thesis supervisor Lukáš Brchl for his overseeing of the process and his support. Next, I want to appreciate the approach of the company Workswell and it's employees, they provided the cameras and drones and offered guidance for working with thermograms. I also want to thank all the volunteers that took part in dataset acquisition.

Declaration

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

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

In Prague on June 4, 2020 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Glejtek, Matej. *People detection using IR camera on a drone for more effective rescue operations.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Bakalárska práca sa zaobera spojením disciplíny nazývanej temografia so softwarovými systémami na dekekciu objektov. Cieľom je pomocou analýzy a testovania nájsť vhodnú metódu, ktorá dokáže zautomatizovať analýzu dát z termokamier na dronoch. Využitie tejto práce spočíva napríklad v zefektívnení záchranných operácií. Pre dosiahnutie daných cieľov bolo potrebné implementovat aplikáciu v jazyku Python, ktorá realizuje detekciu pomocou dostupných systémov, ako je Darknet. Pomocou tejto aplikácie som experimentálne preukázal, že detekcia pomocou neurónových sietí predstavuje najlepšiu možnost a pomocou systému Darknet je možné detekovať objekty dostatočne rýchlo a presne.

Klíčová slova termografia, UAV, analýza obrazu, neuronové siete, termokamery

Abstract

This bachelor's thesis investigates the usage of object detection algorithms on images captured by an infrared camera placed on a drone. The solution will help to automate the analysis of captured data, targeting to increase the effectiveness of rescue operations. During the completion of the task, I developed a Python desktop application, that realizes chosen detection methods. The methods selection was based on an analysis of current approaches and take advantage of the existing detection systems. The application was used to measure the accuracy and performance of these approaches on the dataset created as a part of the thesis. In the end, the conclusion evaluates the possibility to use image detection on a thermogram, in a real-world application. The single-stage Region Proposal Convolutional Network showed the best result and was chosen for future development.

Keywords thermography, UAV, radiometry, thermal cameras, object detection, neural networks, search and rescue

Contents

1 Introduction **1**

2 Theoretical Background **3**
2.1 Thermography 3
2.1.1 History of thermography 4
2.2 Thermal cameras 4
2.3 Physics of Thermography 5
2.3.1 Equation of Thermography 6
2.3.2 Measurement with thermal cameras 7
2.3.3 Visualization 8
2.4 Human skin temperature 9
2.5 Current thermal analysis methods 9
2.6 Unmanned Autonomous Vehicle (UAV) and thermography . . . 10
2.7 Analysis of object detection methods 11
2.7.1 Introduction to image processing and object detection . 11
2.7.2 Object detection approaches 11
2.7.3 Convolutional Neural Networks (ConvNet or Convolutional Neural Network (CNN)) 12
2.7.4 Region Based Convolutional Neural Network 15
2.7.5 Fast Region Based Convolutional Neural Network . . . 16
2.7.6 Faster Region-Based Convolutional Neural Network . . 16
2.7.7 You Only Look Once 17
2.7.8 Machine Learning Methods 19
2.8 Related work 19
2.8.1 Doherty & Rudol paper - Human body Detection [40] . 20
2.8.2 Corcoran paper - detecting koalas 20
2.8.3 Chretien paper - wildlife sensing [17] 20

3 Design **21**

3.1 Specification of requirements . . . 21
3.2 Use cases . . . 22
3.3 Chosen technologies . . . 22
3.3.1 Qt Framework and PySide 2 . . . 22
3.3.2 Used libraries . . . 24
3.4 Software architecture . . . 24
3.5 Choosing the suitable option object detection method . . . 26

4 Realisation **27**
4.1 Data Acquisition . . . 27
4.2 Application Overview . . . 30
4.2.1 Project Structure . . . 30
4.3 Creating Graphical User Interface . . . 30
4.3.1 Application Window design . . . 31
4.3.2 Custom Graphic Components . . . 32
4.4 Application Implementation . . . 33
4.4.1 Graphical User Interface (GUI) Controllers . . . 34
4.4.2 Image handling . . . 35
4.4.3 Visualizations . . . 36
4.4.4 Multithreading . . . 36
4.5 Detection Implementation . . . 37
4.5.1 Blob detection algorithm . . . 38
4.5.2 You Only Look Once (YOLO) Detector . . . 43
4.5.3 Faster Region Based Convolutional Neural Network (Faster R-CNN) Detector . . . 44
4.5.4 Presentation of results . . . 44

5 Testing and detection evaluation **47**
5.1 Detection testing . . . 47
5.1.1 Testing conditions . . . 48
5.1.2 Results . . . 48
5.1.3 Performance . . . 49

6 Conclusion **51**

Bibliography **53**

A Contents of enclosed CD **63**

List of Figures

2.1 Infrared part of the Electromagnetic Spectrum [10]. 6
2.2 Measurement of the surface temperature scheme, modification of work by J.Sova [8] 8
2.3 Different visualizations of the same image. 9
2.4 Simple CNN scheme [23]. 12
2.5 Convolution: 1 - green matrix represents one chanel 5x5 input image for simplicity, 2 - feature detector, 3 - first step of matrix multiplication, 4 - last step of matrix multiplication [23]. 13
2.6 Max Pooling [23]. 14
2.7 Fully connected layer [23]. 15
2.8 R-CNN scheme [23]. 16
2.9 Faster R-CNN scheme [29]. 17
2.10 YOLO Model [29]. 18

3.1 Signals and slots connection [42] 23
3.2 MVC example: displaying loaded files in a table 25
3.3 Application package diagram. 26

4.1 Camera and Drone setup used for the data acquisition [47]. 28
4.2 Example of captured data, both visible and thermal image. 28
4.3 Example of captured data from session 2. 29
4.4 Setup used during session 2. 29
4.5 Application GUI overview with distinction of the most important parts. 31
4.6 Application GUI with native look, Linux on the left and MacOS on the right. 32
4.7 Part of the class model showing the situation with the detection management. Dashed line notes signal-slot connection. 38
4.8 Examples of blobs or candidate areas. 40
4.9 Examples of blobs that are false positives. 40

4.10 Foreground Extraction. 41
4.11 Field of view of the camera 50 meters above ground[61]. 42
4.12 Blobs with masked contours. Left one is a front part of a car, another 3 are people. 43
4.13 Application window displaying results of a detection. 45

List of Tables

5.1 Detection results comparison. 48
5.2 Detection speed comparison. 49

Chapter 1

Introduction

The main goal of this thesis is to investigate and evaluate the possibility to use image detection methods on images captured by thermal cameras and thus connect the fields of thermography with image detection. Thermal cameras on drones are helpful for scanning the environments from above and can survey large areas for temperature disturbances. This thesis aims to take advantage of the unique functionality of thermal cameras to propose a solution that would make rescue operations more effective by detecting possible points of interest automatically. I will use the thermal imprint of mammals to predict their presence and position. Development is supported by the company Workswell, which will provide cameras and contribute to the realization, with the possibility to further utilize the solution into their products.

In the process, I will create a desktop application written in Python programming language that implements the detection of objects on images captured by thermal cameras, more specifically cameras on unmanned autonomous vehicles. The application will serve as a demonstration of the concepts presented in this thesis and will be developed concerning future development, particularly real-time detection. The desktop version works with captured thermal images and should provide basic image browsing features and realization of chosen object detection algorithms, with emphasis on the detection of people or animals. I would like to use the created application later to compare the accuracy and performance of chosen detection methods.

Before realization, I will explain the basic concepts of thermography and temperature measurement with infrared cameras and point out possible difficulties that may arise when dealing with the task. Next, I will provide an analysis of image processing and object detection methods and detailed research on relevant algorithms and related work. It is crucial to choose the detection method that best suits the application needs and can work with the data the most effectively and efficiently. That is why I will attempt to utilize more methods to be able to compare them.

After laying out the required theoretical background, realization with sev-

eral stages needs to be completed to achieve the overall goal. Realization starts with a creating dataset of thermal images that will be used for training and testing. It is necessary to prepare distinct scenarios and environments to cover situations that may occur during rescue operations. After completing and annotating the dataset, I will compare captured data from the first stage with the capabilities of researched algorithms to select the possible best options to use in the application. Another dataset collection may be required in the later stages. After the evaluation of the data, I am going to select the technology for the application. I will build the application using Python programming language in conjunction with several libraries for processing images, a graphical interface framework and most importantly, the implementation will make use of available systems that implement state-of-the-art object detection algorithms. The software analysis stage consists of modeling the use cases, describing the application domain, and proposing the architecture of the solution. After determining the software architecture and dependencies I will move to implementation. During development, I will apply the object-oriented approach and focus on creating a clean structure and apply software development practices and patterns.

After implementation, I will test the solution using automated evaluation of detection by comparing application output with the original annotations and compare results of different methods. Testing will include creating a metrics for determining accuracy and performance. I will analyze and exploit obtained results and make final considerations about the usage of object detection in thermography, highlighting the most important remarks or difficulties. Lastly, I will outline possible future development and extensions.

Chapter 2

Theoretical Background

This chapter contains basic theoretical background that is required for the completion of the thesis. I will explain what is a thermography, what tools it uses and list a few applications. Then I will briefly describe the physics behind and discuss possible difficulties in measurement. The second part focuses on the introduction to image processing and object detection, together with the analysis of available detection algorithms. In the end, I will summarize related work, pointing out important findings.

2.1 Thermography

The Thermography, or Thermovision, is a technical science discipline. The goal of a thermography is to measure temperatures without physical interaction with measured objects. This is achieved by observing electromagnetic radiation, especially in the infrared part of the spectrum. The radiation is collected and processed using devices called thermal cameras. There are many various applications of thermography nowadays basically in every industry. To name a few, thermal cameras are good quality assurance tool, could detect fire even before the first flame appears, detect gas leakage, and so on. According to Jan Sova, thermal cameras are today used to determine surface temperatures in virtually every industrial sector and scientific research, ranging from heavy metal processing industry to medicine and microbiological research[1]. Medical usage of thermography was made apparent during the Coronavirus pandemic of 2020, as cameras were adopted as reliable tools for body temperature screening. Nowadays, thermography based-solutions are installed in public places to help fight the pandemics by detection of people with fever[2].

The application on which I want to focus on is the help with search and rescue operations. Infrared cameras can see people and animals that would be otherwise hidden from human sight, for example in a forest and other greenery. Human, being a homeotherm, is capable of maintaining a constant tempera-

ture of the body, which may be different from surrounding temperature [3]. I plan to utilize this characteristic into the solution that uses temperatures to detect living objects on thermograms.

2.1.1 History of thermography

Origins of thermography, date back to the 19th century when thermal radiation was discovered. It was discovered by William Hershel when he was experimenting with passing the light through a glass prism. He unwillingly placed a thermometer next to the part where the red part of the light shined, and he discovered that this part is the hottest, even though he could not see any visible light hitting the thermometer. This way, he demonstrated the presence of invisible radiation whose energy could be detected by its heating effect[4]. Hershel discovered a new spectrum of invisible light which we now know as infrared, meaning "below the red"[5]. Bolometers, which are devices for measuring thermal radiation, made their appearance in 1880[4]. "By the 1920s, scientists were using photography to record the infrared spectrum. The 30s, 40s, and 50s saw remarkable improvements in imaging with special infrared sensors, thanks in large part to World War II and the Korean conflict, which used infrared for a variety of military applications, such as troop movement detection. Once these infrared technologies were declassified post-war, scientists immediately turned to research of their other applications"[5]. The first company that could bring the thermography to market is the American company called Forward-looking Infrared (FLIR). They introduced cameras for commercial use during the 1970s. It is perceived as a pioneer in this field and also the current global leader in production, even though there are more manufactures now, one of them being the Czech company Workswell. The term "Thermovision", which is another name for thermography originates from the name of a company Thermovision, manufacturer of the first thermal cameras, which was later renamed to FLIR.[6].

2.2 Thermal cameras

Thermal camera is a device that captures and measures infrared radiation and processes the radiation into output in form of temperature for each pixel. A thermal camera consists of several parts, namely the optics, core, computing unit and housing. The most important component is the camera core with bolometers. These are small devices, one per each camera pixel, that capture radiation and output values of resistance. The stronger the radiation, the higher the resistance. The active element in a bolometer is typically a resistor with a large temperature coefficient of the resistance[7], which means that resistance increases rapidly with the change in the temperature. As a consequence, a significant change in resistance occurs when the detector is heated by incident infrared radiation. Camera computing unit then applies several

computations to work out the final temperature from the measured values of resistance. This computations involve working with physical parameters which influence the measurement. These parameters needs to be added by the user to correct for the external impacts. Another indispensable part of the camera is the lens, which is usually made of precious germanium glass, because such lens enables passing of the light just in the infrared spectrum. Output of a camera is a two dimensional map of pixels, similar to classic digital image, but values of each pixels represent thermal radiation, not brightness. Such images are called thermograms. Notably, the cameras split into two categories - radiometric and non-radiometric. Radiometric thermal camera is able to measure the actual temperature per each pixel in absolute scale and using the SI units. On the other hand, non-radiometric cameras only compare the apparent temperature of objects on scene and visualize the temperature differences in relation to objects on a scene. One easily imaginable representative of non-radiometric thermal camera is the night vision device. For the purpose of this thesis, I will focus on the radiometric thermal cameras and refer to them as just thermal cameras from now on.

Thermal cameras could be very different from each other, from low end handheld devices with accuracy worse than 1 degree to a scientific cameras with cooling. The most important parameters, when speaking about accuracy of measuring temperature, are resolution and uncertainty of measurement[8]. I am going to use cameras that are adjusted to usege with drones, with lightweight construction and firmware compatible with drone operating systems.

2.3 Physics of Thermography

All materials continuously emit and absorb electromagnetic waves, or photons, by lowering and raising molecular energy levels. The strength and wavelengths of emission depend on the temperature of the emitting material[9]. We perceive the thermal radiation as heat. Infrared is the part of the electromagnetic spectrum with the wavelength longer than the visible light and shorter than microwaves. Thermal cameras benefit from the fact that the intensity of infrared radiation emitted by each object rises with the rising temperature of the object. This is very useful for determining the temperature, but on the other hand, we could not forget that also material parameters influence the amount of radiation and we need to correct for them.

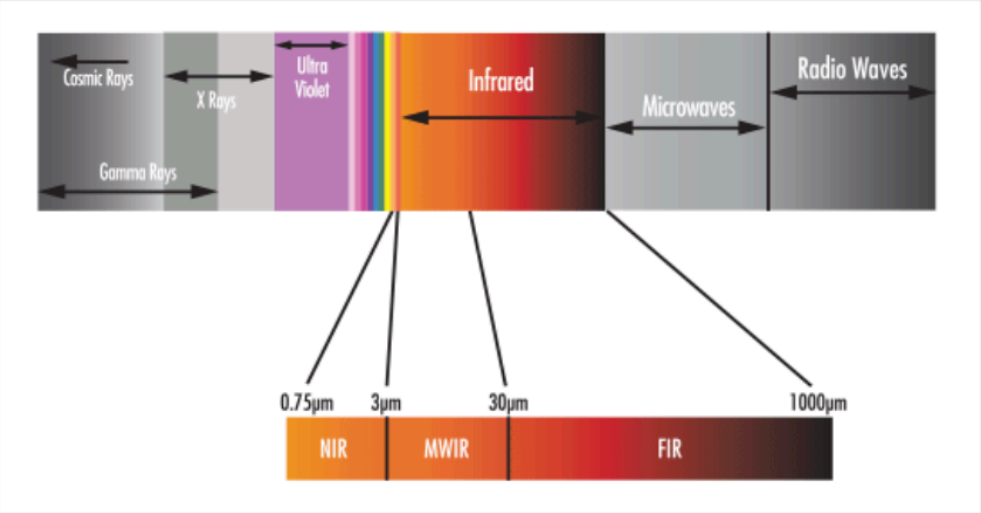

Figure 2.1: Infrared part of the Electromagnetic Spectrum [10].

To be able to model thermal radiation, scientists come up with an idea of the blackbody. According to thermal radiation theory, a blackbody is considered as a hypothetical object that absorbs all incident radiation and radiates a continuous spectrum according to Planck's law[9]. Blackbody represents and the ideal case for thermography because it emits spectrum that is determined by the temperature alone. However, objects in the real world emit the radiation differently and exact measurement requires knowledge about the material we want to measure.

The fact, that the intensity of thermal radiation from the surface of objects increases with the surface temperatures, could be utilized in measuring the surface temperature. And it actually is, but the situation is a bit complicated in practice and it is necessary to know and take care of parameters of measurement and work with so-called "Equation of thermography", which takes into consideration the influence of the atmosphere and other nearby objects. [1].

2.3.1 Equation of Thermography

The equation of thermography describes the radiation which enters the camera and its relation with detected temperature[8]. All the factors playing part in a measurement are represented by this equation [1]. They usually need to be filled in by the user of the cameras, therefore it is important to understand their role. Details will be presented in the next section. Visual explanation of the equation is displayed on Figure 2.2

2.3.2 Measurement with thermal cameras

It is important to keep in mind that the are many variables playing part in the measurement in thermal cameras. Thermal cameras do not measure the temperature directly, but it determines the temperature based on thermal emission and given parameters [8]. The most important is the emissivity. It is a property of a surface that can be described as an effectiveness of emitting heat. Its value is a fraction of the emissivity of a surface blackbody that emits thermal radiation at the highest possible rate. The emissivity also influences how much other variables affect the measurement. The emissivity of human skin is reported to be almost constant and its value is 0.98 [3]. Most of the cameras, as well as the camera used for this project, offer emissivity correction directly in the camera. Another important variable is the reflected temperature. This variable stands for the thermal radiation that originates from other objects and reflects off the object we want to measure. Its impact is lowered with lower values of emissivity [8]. Other parameters include the description of an atmosphere (humidity) and parameters of the optics used, but they are not so influential and can be omitted in most cases. All the mentioned parameters play their part in an equation of thermography, which is used to compute the final temperature. To conclude, measurement using thermal cameras can be very useful, but it is not totally exact and may be dependent on external factors. Also, it is necessary to perceive the resulting temperature as the temperature of a surface, not a whole object.

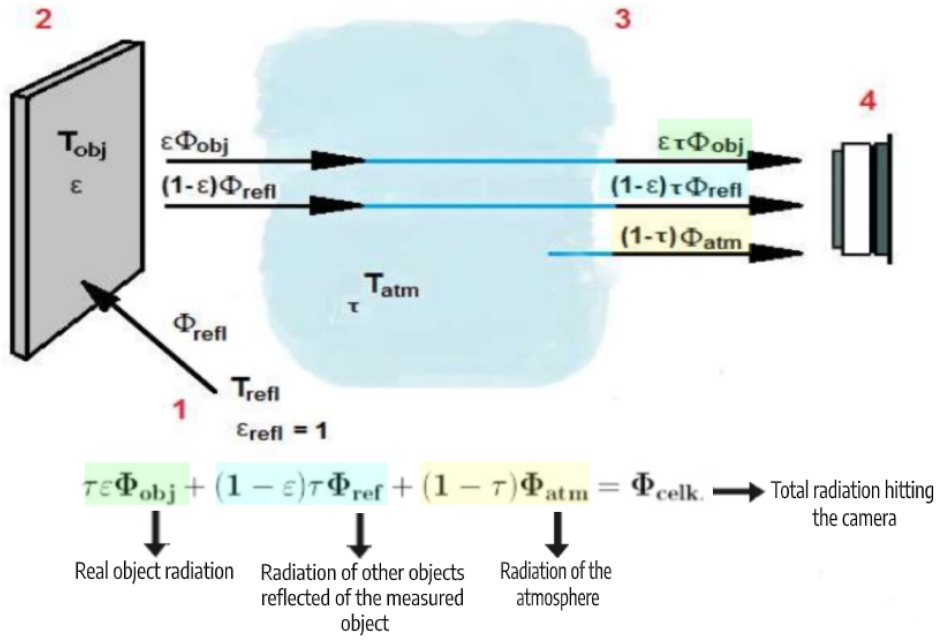

Figure 2.2: Measurement of the surface temperature scheme, modification of work by J.Sova [8]

1. Heat radiation of the surroundings, which reflects on surface of the measured object,

2. Surface of the measured object,

3. Atmosphere, which attenuates the radiation from other objects as well as emits own.

4. Camera Lens, resulting radiation is the sum of the previous points.

2.3.3 Visualization

During the measurement, only the invisible infrared light is captured. All the colors we see on thermograms are a result of visualization and are artificial. Visualization is done using color palettes, which define a range of usually 256 colors and one color represents a specific temperature range. For each pixel, the color for that pixel is calculated from its temperature. This aritificial coloring gives me big manuevering space, as I can modify visualizations by highlight specific color ranges or using custom palettes.

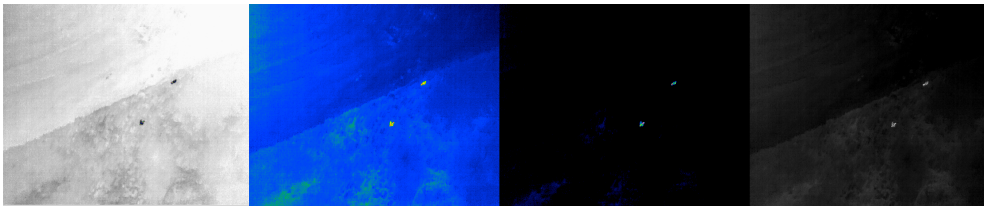

Figure 2.3: Different visualizations of the same image.

2.4 Human skin temperature

In order to be able to correctly identify the humans on thermograms, it is important to know for what temperatures to look for. Normal human skin temperature on the trunk of the body varies between 33.5 and 36.9 °C, though the skin's temperature is lower over protruding parts, like the nose, and higher over muscles and active organs [11]. The temperature of human skin varies with the environment. Researchers from the School of Mechanical Engineering in the USA and Tianjin University in China measured temperatures of human skin in various conditions with temperatures ranging from -0.1 to 32 °C. The study is summed up in a paper[12], which also presents results. The study used 24 people as test subjects. They were exposed to various temperatures in a climatic chamber and researchers measured their surface temperature on various body parts. They found out that in a cold environment, human facial skin can have just 18 degrees Celsius. Generally speaking, the highest temperature is on the head and abdomen and lowest on hands and nose.

To conclude, the average human body temperature varies between 30 and 34 degrees Celsius, with a minimum of 24 degrees. However, only certain parts of a body are exposed normally, so I need to concentrate mainly on the temperature of the head, arms, and legs, as most of a human body is covered with clothes. I will be testing the temperature of humans during the dataset collection.

2.5 Current thermal analysis methods

Companies that are manufacturing thermal cameras usually provide software for the analysis of thermograms and thermal sequences. Their desktop software is usually proprietary and it is accessible mainly to camera owners. From the functionality point of view, they offer basic temperature visualization, elaborating certain regions of interest on the image, changing the parameters discussed in the previous section, and finally, creating reports. However, none of the tools I came across offer task automation or any form of object detection or another service for rapid evaluation and search of many sources for

objects. The alarm function, which highlights a certain part of the image with a temperature above or under a given threshold, may be useful but could not be used reliably because the temperature of an environment varies and also influences the temperature of target objects. Examples of thermal analysis software may be CorePlayer by Workswell[13] or FLIR Tools+ by FLIR [14]. Image data collected from aerial surveys are often interpreted manually, but this can be tedious, time-consuming, and is still subject to interpreter bias with a tendency towards the low probability of detection and high rates of false negative error [15] [16]. Automated detection in remotely sensed imagery can reduce bias and increase the accuracy and precision of surveys, but few methods have been developed and tested in the field [16]. For mammals, the automated detection methods are shown to be most accurate thus far have been applied to thermal imagery, as the large temperature gradient between mammals and their background environment allows computer vision to easily detect and count their thermal signatures. [17]

2.6 UAV and thermography

Currently, UAVs, otherwise known as drones, are finding their way into many aspects of our lives. One of the most obvious applications is equipping the vehicle with a camera and flying above unreachable areas. Such systems are today widely used by filmmakers and photographers to capture wonderful shots, but the UAV has the even bigger potential as a tool helping in search and rescue operations. Operation of the drone must obey laws which differ between the countries. The authorities in these countries may require the operator to hold a license and several areas are restricted, such as densely populated areas or plane corridors and airports. In a Czech republic, the regulator is the Czech Republic Civil Aviation Authority. According to the guidelines[18], published by this institution, the drone operation is legal with certain rules. The commercial usage is permitted only with a license. The operations is are generally permitted in the airspace in from the ground up to 300 meters above the ground. A visual line of sight must be maintained with the UAV at all times by the drone pilot [19]. The Czech company Workswell is a significant player in manufacturing thermal cameras specially for drones. Their camera WIRIS is fully compatible with Dà-Jiāng Innovations (DJI) Drone and it is easy for the operator to control the drone and camera simultaneously. The signal is transmitted directly to remote control with a tablet connected to the drone, so the operator can see thermal as well as digital pictures in real time. The functions of the camera can be accessed directly from the same remote control. For the dataset collection, I am going to use camera WIRIS Pro, which is the third iteration [20], together with DJI Matrice drone. The equipment was provided by Workswell.

2.7 Analysis of object detection methods

In this section I will introduce the topic of object detection on images, listing various concepts used. Subsequently, I will explain selected methods, based on research by a number of computer scientists and describe available systems that implement these methods. Most of the algorithms proposed for these tasks have open-source implementations, that are ready to be used.

2.7.1 Introduction to image processing and object detection

Digital image processing is a discipline of computer science. In simple words, it is the usage of a computer to manipulate the images. The field of image processing itself covers different topics, from image enhancement to compression, restoration, and analysis. In my thesis, I will concentrate on the analysis part, which also includes object detection. The image processing gains more publicity and recognition, as it is used for face detection, vision text processing, and many other uses, many of which we have in our smartphones. Object detection is a technique of image processing for locating and classifying the objects captured on images or videos.

2.7.2 Object detection approaches

There are various approaches to object detection. They can be roughly divided into categories according to the paradigm used. There are methods that use traditional programming and also that use artificial intelligence and machine learning or deep learning approach. Standard algorithms, that are not based on machine learning, and working with data, excel in solving the tasks, which can be easily defined. For example, a task to process an online shop order can be easily defined in a straightforward code. It consists of creating n database entry, filling in invoices, managing payment, and others.

Considering the case I am attempting to solve, it is not perfectly clear whether the solution can be described procedurally. Straightforward implementation could focus on the analysis of the temperature data and looking for a specific temperature range and filtering candidate ideas by shapes. On the other hand, this solution would be very error-prone, as it would not be able to adapt well to various new situations.

For task that does not have a simple workflow, scientist came up with an idea to imitate human reasoning and simulate the neural network in software. The network is similar to that in a human brain, but simpler, obviously. This network has a learning ability and is capable to train itself using large amounts of data. After training, the network can do its job with a high probability of success. It is important to keep in mind that neural networks are suitable just for certain types of tasks and the result is always in a form of probability. The downside of them is the requirement of a lot of training data and

computing power.[21]. Basically, object detection with some form of Artificial Inteligence (AI) is based on two main approaches: Either deep learning or machine learning approaches. The difference is whether the algorithm has some prior knowledge about the image features before the detection. In machine learning-based approaches, it detects the features using Haar, Scale-Invariant Feature Transform (SIFT) and Histograms of Oriented Gradients (HOG) then for classification it can use Support Vector Machines (SVM) [22], whereas in deep learning approaches it generally uses the CNN for the object detection without requiring any knowledge about the features.

2.7.3 Convolutional Neural Networks (ConvNet or CNN)

CNN is a very popular algorithm for image classification. It is a type of deep learning network, based the mathematical operation called convolution. Convolutional networks perceive images in three dimensions, rather than flat canvases to be measured only by width and height. Digital color images have a Red Green Blue (RGB) encoding, mixing those three colors to produce the color spectrum humans perceive. A convolutional network ingests such images as three separate strata of color stacked one on top of the other. Those depth layers are referred to as channels. Convolutional neural networks ingest and process images as tensors, and tensors are matrices of numbers with additional dimensions. As images move through a convolutional network, they are being discribed in terms of input and output volumes, expressing them mathematically as matrices of multiple dimensions.

There are four main operations in CNN: Convolution, Non-Linearity(Rectified Linear Unit (ReLU)), Pooling, Classification(Fully connected layer).

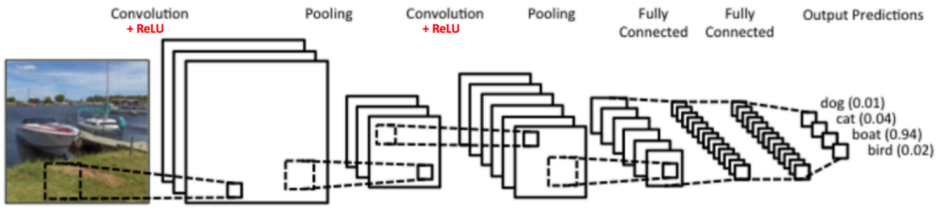

Figure 2.4: Simple CNN scheme [23].

Convolution

The purpose of the convolution step is to extract features from the image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data [23].

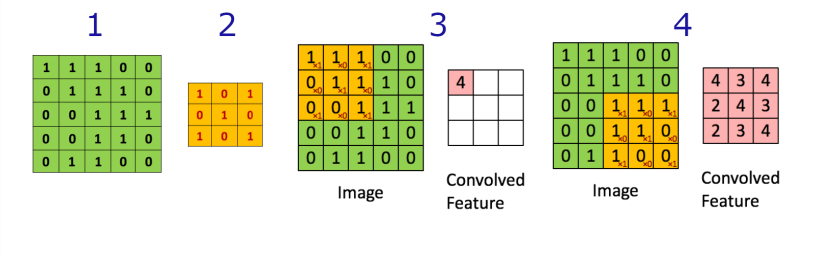

Figure 2.5: Convolution: 1 - green matrix represents one chanel 5x5 input image for simplicity, 2 - feature detector, 3 - first step of matrix multiplication, 4 - last step of matrix multiplication [23].

The convolution step has two matrices as input, one of them being the input image and second matrix is called a filter. In the process, filter is placed over the image and for every position, element wise multiplication is applied, performing dot product.The result matrix is called a feature map. "Using different filters will produce different feature maps. In practice, a CNN learns the values of these filters on its own during the training process. The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images" [23].

Non Linearity - ReLU

After each convolution step, activation function is applied. The activation function models the biological processes and is an abstraction for the rate of action potential firing in the cell. In its simplest form, this function is binary—that is, either the neuron is firing or not. In CNN, negative values in feature maps are replaced by zeros by function textttmax(0, val). Browniee[24] describes this step in the following words: "The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero".

The Pooling Step

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum, etc.

In case of Max Pooling, we define a spatial neighbourhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

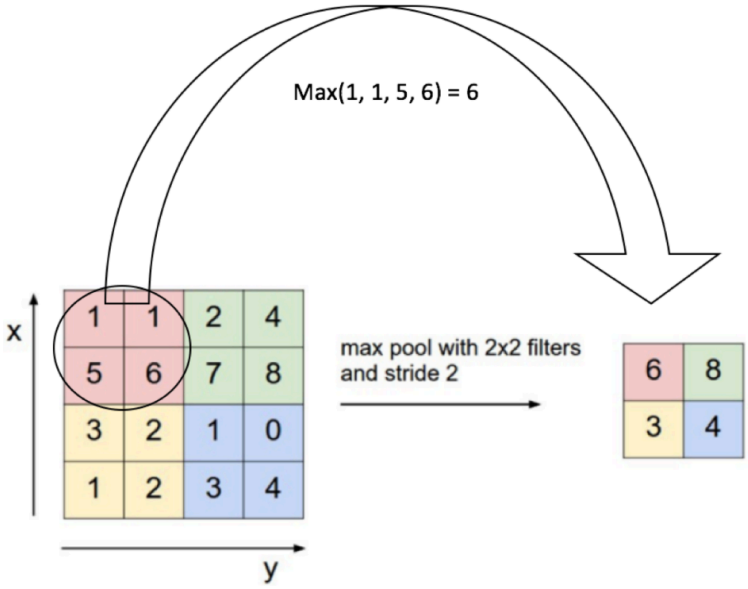

Figure 2.6: Max Pooling [23].

Fully Connected Layer

The purpose of this layer is to make use of the features extracted in previous steps. It uses softmax activation. "The output of convolution/pooling is flattened into a single vector of values, each representing a probability that a certain feature belongs to a label. For example, if the image is of a cat, features representing things like whiskers or fur should have high probabilities for the label cat" [25]. "Fully Connected layers in a neural networks are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular machine learning models, the last few layers are full connected layers which compiles the data extracted by previous layers to form the final output" [26].

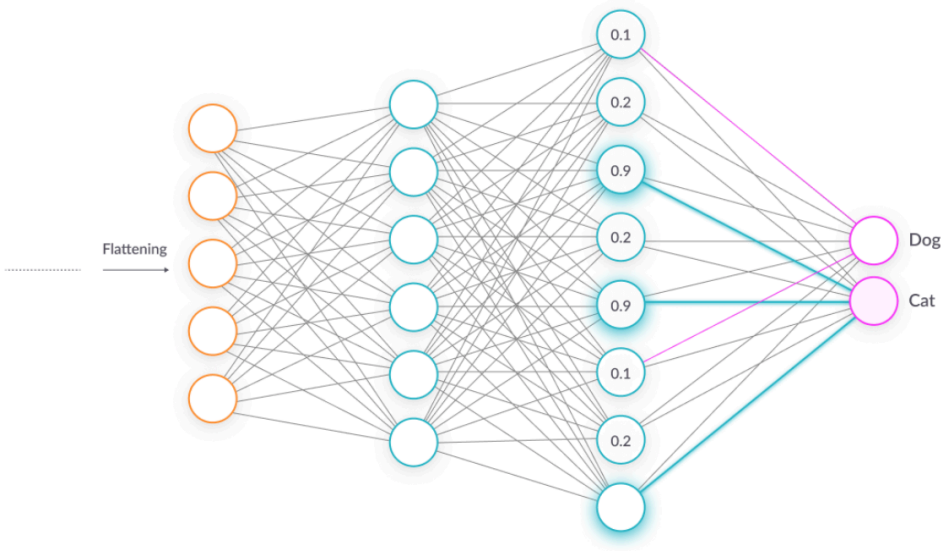

Figure 2.7: Fully connected layer [23].

As I stated before, CNN is a tool for image classification. To successfully detect humans on thermograms, I need not only to classify the objects, but also work out their location. For the object detection, there are four approaches:

- Region Based Convolutional Neural Network (R-CNN),
- Fast Region Based Convolutional Neural Network (Fast R-CNN) [27],
- Faster R-CNN [28],
- YOLO [29] [30].

Listed approaches can be divided according to their architecture, which can be one stage or two stage. Two stage frameworks address this issue by adding an object proposal method and then classifies these proposals using CNN. These are the first three from the list. The last one uses one stage architecture.

2.7.4 Region Based Convolutional Neural Network

R-CNN generates 2000 proposal in a first step and CNN is then applied to each one of these region proposals for the classification. Region proposal uses selective search algorithm [31], that consists of initial generation and combining similar regions to larger ones. "The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image

and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal" [32]. This method takes long time to both train and use, and definetely could not be used in real time, as it takes about 40 seconds to run the method on one image. Also, first stage is fixed and does not use any learning.

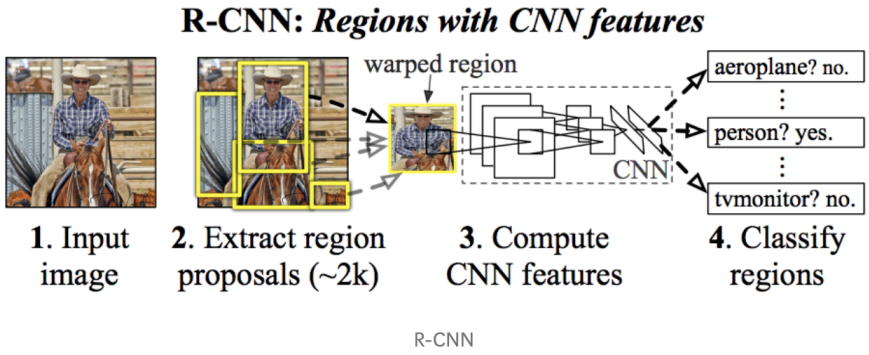

Figure 2.8: R-CNN scheme [23].

2.7.5 Fast Region Based Convolutional Neural Network

The difference between R-CNN and Fast R-CNN is that the second one applies the CNN first, before region prosal. It uses CNN just once, making a significant increase in speed. "Instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map"[32]. From the convolutional feature map, the algorithm identifies the region of proposals and warp them into squares. By using an Region of Interest (RoI) pooling layer these squares are reshaped into a fixed size so that it can be fed into a fully connected layer. From theRoI feature vector, a softmax layer is applied instead of SVM to predict the class of the proposed region and also the offset values for the bounding box.

2.7.6 Faster Region-Based Convolutional Neural Network

Faster R-CNN starts in the same way as the Fast R-CNN creating the feature map. Instead of using a selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. It replaces selective search with a region proposal network [33]. The predicted region proposals are then reshaped using an RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes[32].

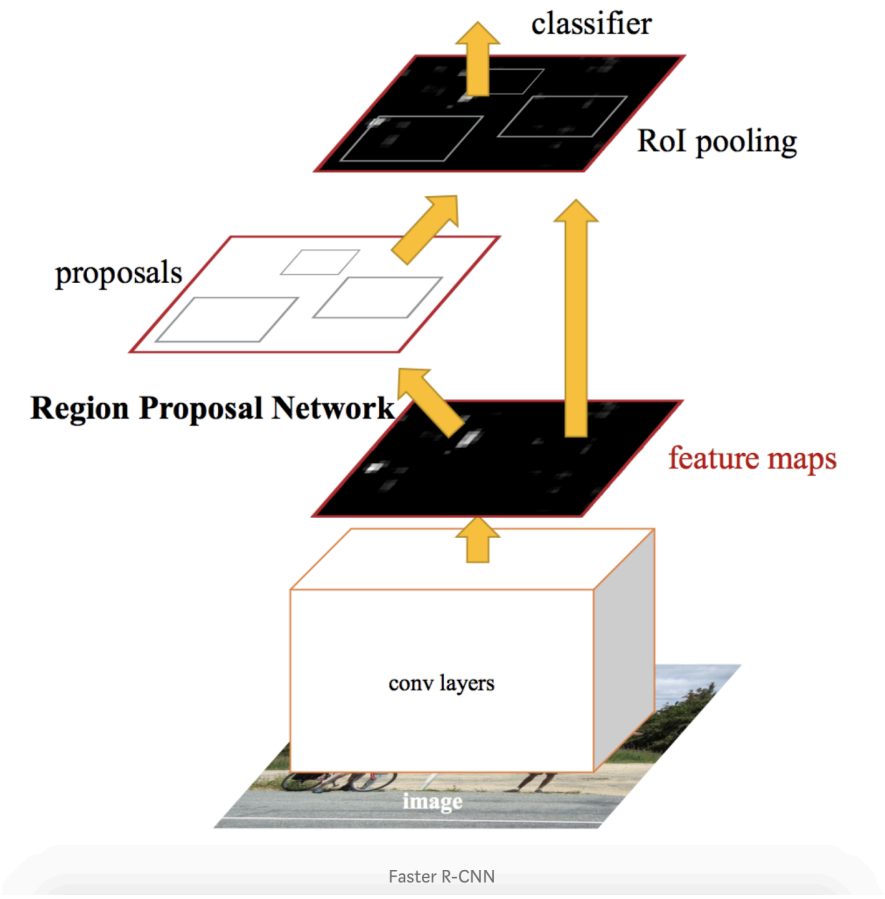

Figure 2.9: Faster R-CNN scheme [29].

2.7.7 You Only Look Once

The original YOLO paper [29] describes the object detection model that uses a single convolutional network to simultaneously predict multiple object bounding boxes in full images as well as class probabilities for those boxes.The network architecture of this model has 24 convolutional layers and two fully connected layers. The convolutional layers perform feature extraction while the fully connected layers predict the bounding box locations and their probabilities. The system first divides the input image into a square grid. Two bounding boxes and corresponding class confidences are associated with each grid cell, so at most two objects can be detected within a cell, and if an object occupies more than one cell, the centre cell is selected to be the holder of prediction for that object [34]. For each of the bounding boxes, the network outputs a class probability and offset values for the bounding box. The bound-

ing boxes having the class probability above a threshold value is selected and used to locate the object within the image [32].

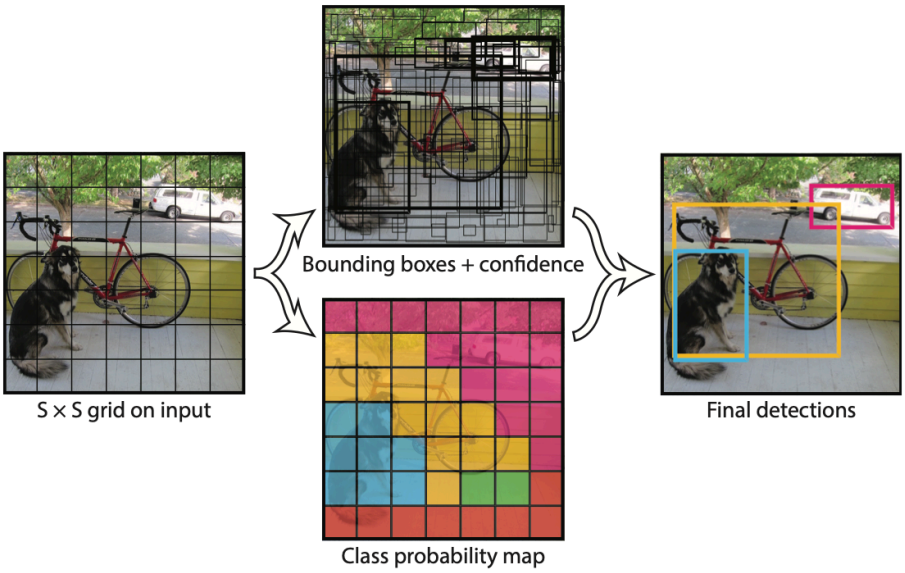

Figure 2.10: YOLO Model [29].

It is quite easy to start developing with YOLO, as pre-trained models and image datasets can e found on the internet. One of them is called Common Objects In Context (COCO) [35].YOLO is one of the most successful algorithms today and there have been certain attempts to use it for thermal images too. However, because of its nature, the original YOLO is not suitable for detecting small objects. According to paper on this topic [34], the features that YOLO has learned on a large COCO dataset of RGB images will still provide a reasonable baseline for thermal images. Unfortunately, due to the difference between visual and thermal images, the original YOLO model (bYOLO) has achieved average precision of only 7% for person detection in the thermal images. That result is significantly worse than the results YOLO achieves on the images of the visible spectrum where the results depending on the scenario range around 90 %. To make the precision much higher, additional training is compulsory. To fully train the network, thousands of images are required, Luckily, there is a method called transfer learning, which takes a pre-trained network and applies additional training on new data. This way, the network can be modified using just hundreds of images.

2.7.8 Machine Learning Methods

Machine learning aims to create rules from observations, guided by the knowledge representation required for a specific system. There are various machine learning techniques used in the development of image interpretation systems [36]. Learning can be supervised or unsupervised. They differ in a fact that with supervised learning the criteria for labeling data are known explicitly. Given some training data describes in terms of a set of features and their labels, the goal is to find a partitioning of a feature space that allows correct classification of all training data, as well as unseen, similar data [36]. Most of the implementations follow popular pipeline with three main steps: region proposal generation to generate a set of candidate boxes that may cover objects, proposal feature extraction to extract features from these proposals, and proposed classification to classify each proposal as an object class, or background [37]. I would like to use a similar pipeline, with the exploitation of thermal data in the region proposal step. To extract features, several feature descriptors are used. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information [38]. HOG and SVM are examples of feature descriptors. In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. [38]. The HOG can be calculated using OpenCV, as shown in the article [38]. Support vector machines are a core machine learning technology.

2.8 Related work

In this section, I will go through the relevant related work, published on a science papers on the internet archives, such as Arvix or IEEE. The papers usually go through peer reviews and community acceptance, thus are regarded reliable. I will state the challenges, solutions and important notes from the each case. There have been a many initiatives regarding object detection on thermal images as the infrared detectors have big potential in autonomous vehicles. For this, purpose, the free dataset was created. According to FLIR, the ability to sense thermal infrared radiation, or heat, provides both complementary and distinct advantages to existing sensor technologies [39]. On the other hand, there are not many examples to use object detection for thermal cameras on drones. One such example is the paper by Doherty and Rudol [40]. When considering using thermal cameras to detect animals, I was able to find works on koalas [41] and wildlife [17]. In the sources I found, authors used several different detection methods. I will take a closer look on these three papers in the following subsections.

2.8.1 Doherty & Rudol paper - Human body Detection [40]

In their paper, Doherty and Rudol present an interesting technique for people detection using thermal imaginary on UAV. Their setup involves thermal camera, digital camera and hardware unit on board the mini helicopter. They employ both cameras to work complementary, as they decided to separate the detection into two phases. As they stated, "The technique presented detects humans at a rate up to 25Hz (sporadically lower for scenes with high numbers of potential bodies) by first analysing an infrared image to find human-temperature silhouettes and then using the corresponding colour image regions to classify human bodies. A thermal image is analysed first to find human body sized silhouettes. Corresponding regions in a colour image are subjected to a human body classifier which is configured to allow weak classifications".In their conclusion, they stated that the detection had a significant number of false positive, as a result of weak classifications. The approach using both cameras holds potential for my case, however, much of their work focuses on calculating the right transformations between the images ad the thermal and visible cameras have different resolutions and field of view.

2.8.2 Corcoran paper - detecting koalas

Another interesting paper is by Australian researchers. They aim to monitor the presence and distribution of endangered animal species. In their words: "This study introduces a new automated method for detection using published object detection algorithms to detect their heat signatures in UAV-derived thermal imaging" [41]. The detection method used combines R-CNN and YOLO, creating a pipeline. Images are reviewed by both detectors and then the results are combined. The results show that the automated method had an overall probability of detection at 85 percent.

2.8.3 Chretien paper - wildlife sensing [17]

The subject of this paper is animal detection in nature using thermal imagery and drones. The author aims to research the possibility of automation in the monitoring of wildlife. Even though the paper focuses on animals, it is quite relevant as the author tackled the problems of height and low resolution. The proposed method consists of creating Multicriteria Object-Based Image Analysis (MOBIA). The paper results "showed that all bison and elks were detected without errors, while for deer and wolves, 0–2 individuals per flight line were mistaken with ground elements". As the author states, "This method also demonstrated its potential to perform the census of a single targeted species using its specific threshold values. However, more research is needed to improve the detection rate of each species".

Chapter 3

Design

The purpose of this chapter is to propose and describe solution that will be implemented during the realisation. First, I will specify requirements from the point of view of a user, formulate use cases and functional requirements. Then I will move to the non functional requirements, elaborate the software architecture and specify principles that will be use to create user interface.

3.1 Specification of requirements

From the functional point of view, the application has a clear operation domain. Images are given as input, and the product is the set of bounding boxes with detected objects.The exact ways in which the application could be operated follows in the subsequent section. However, the application must obey certain non-functional requirements. The desktop version will serve as a prove of concept and test for the future real time version which will run in real-time. Therefore the application should have clear division between the backend detection implementation and user interface. In the future development, the graphical interface could be swapped for Application Programming Interface (API) deployed on server, exposing the functionality using web services.

- Run on desktop computers with all the operations system, thus be platform independent
- Event-driven, thus the flow is not sequential but controlled by user
- Following object oriented programming principles to maintain scalability, ease of maintenance
- Design with regard to future development
- Layered application with a clearly distinguishable graphical interface and backend parts

3.2 Use cases

Even though the main usage is the testing of detection methods, application will also serve as image viewer for thermograms.

- Browse captured thermal images from the thermal camera with the opportunity to examine measured temperatures
- Survey the metadata saved to thermal images, e.g. GPS tags or camera information
- Browse digital images attached to thermal images
- Classify objects on the images and find their location
- Store the detected objects

3.3 Chosen technologies

To implement the desired application, I decided to use Python programming language, as it is the most used language for the image processing with big community and wide range of libraries and frameworks. All the papers and studies regarding the topic of image processing consider a Python their go-to language. Many of the algorithms I came across were implemented in this language, and many useful methods are already implemented in a libraries, such as *OpenCV*. I will take advantage of the python support of object oriented programming to create application structure and distribute responsibilities across classes grouped to logical application components.

3.3.1 Qt Framework and PySide 2

Another important choice is the choice of the frontend framework. I need Python framework that enables rapid development of GUI with native look on every platform. *PySide* is the ideal candidate, not only it meets the requirements but also I have an experience with continuous development using *Qt Framework*, in which *PySide2* belongs to. Even through I used the *Qt Framework* just for C++ language, using in it Python is similar and uses the same graphical components and development principles. The application created by *PySide2* framework is event-driven, and the most significant feature is the introduction of **Signal-Slot system**. Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks [42]. Signal is a message, that instance emits, and slot is a method that is connected to this signal. This enables communication between the instances together, the system contributes to program

flow control. It is an alternative to callbacks. Callbacks are a pointers to functions that ongoing function calls. Signals and slots are part of the *Qt Meta-Object System* and may be predefined in the Qt classes, but new ones can be created by subclassing. To enable Qt features, created classes must be derived from `QtObject` [43]. Another important part of the framework is the **Event System**. Citing from the Qt Documentation: "Events are objects, derived from the abstract texttttQEvent class, that represent things that have happened either within an application or as a result of outside activity that the application needs to know about. Events can be received and handled by any instance of a texttttQObject subclass, but they are especially relevant to widgets.

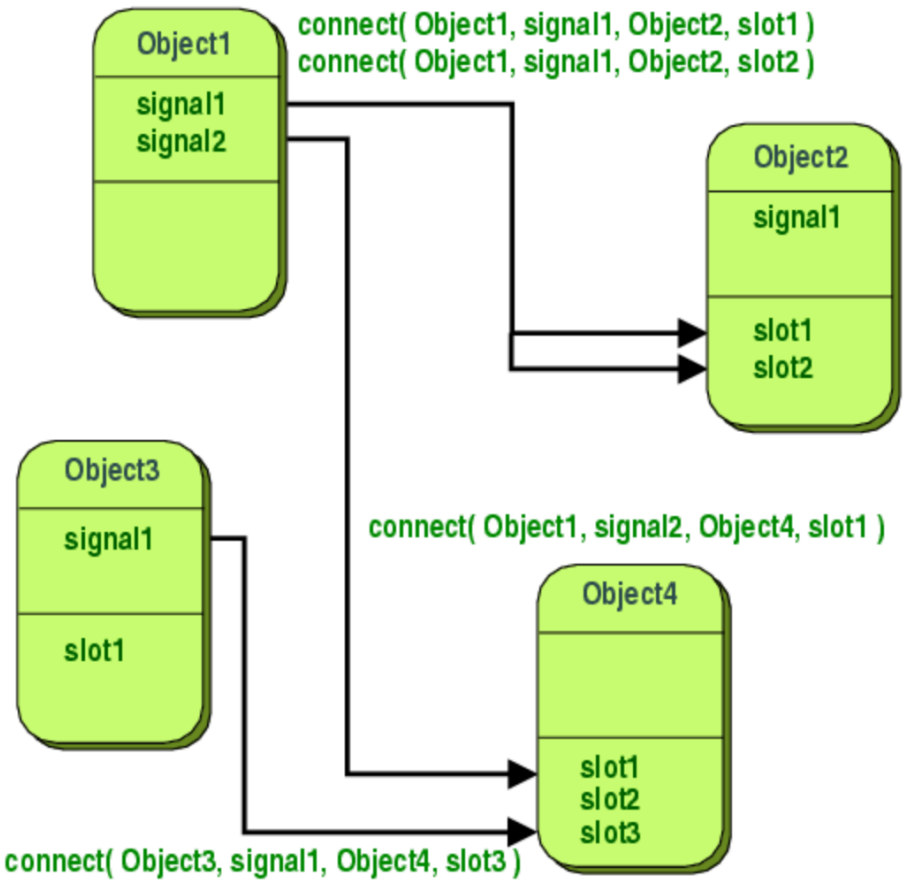

Figure 3.1: Signals and slots connection [42]

3.3.2 Used libraries

In the backend, *PIL Image library* is used to load image, and then in will be processed as a *NumPy* array, which offers convenient way to process the data numericaly. I also need a library for reading exif tags and sending network request for finding out elevation. I selected *exifread* and *urllib*, respectively.

3.4 Software architecture

The software architecture of the application is partially determined by technologies used. Basically, the application can be split into two layers - backend and frontend layers. Frontend layer implements user interface, presents data and handles user input. In the backend layer, we have the application logic, communication with operating system, accessing the file system and most importantly, the detection algorithm. It is important to clearly distinguish the layers and do not mix them, to maintain low coupling, extensibility and ease of the code maintenance.

The part of the application user directly interacts with, is called GUI. I will create an interface that in intuitive, simple and easy to use. Then, after presentation layer is outlined, I will need *controller* classes that will handle user inputs, present data from backend into graphical interface and handle overall communication between frontend and backend. To manage the relationship between data and the presentation to the user, I will use the traditional Model-View-Controller (MVC) scheme. Model is the data provided by backend, view is the graphical interface component and controller, that handles user inputs from view and updates model.

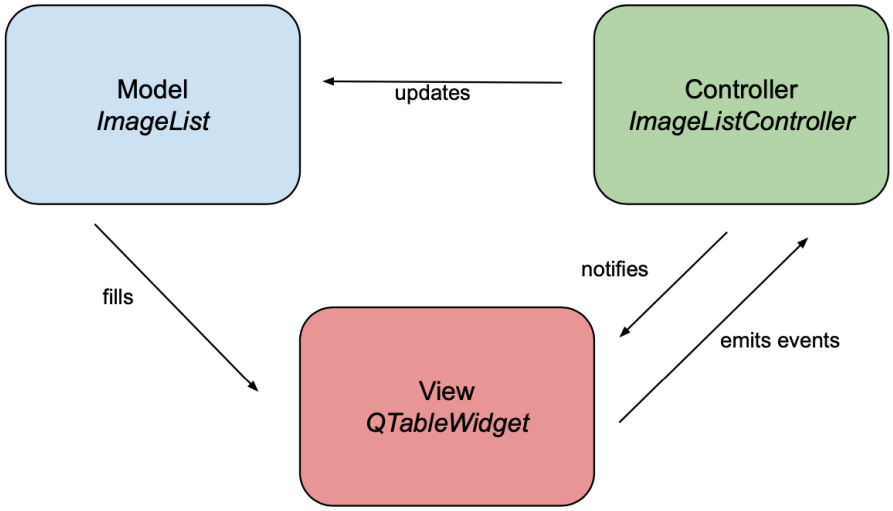

Figure 3.2: MVC example: displaying loaded files in a table

The backend mainly manages application logic and implements the chosen detection algorithm, as well as stores data. Classes communicate with the operation system using known APIs. The application require access to the file system and the internet. The detection should be independent of user interface implementation, as well as specific libraries or frameworks using just core Python tools and established libraries.

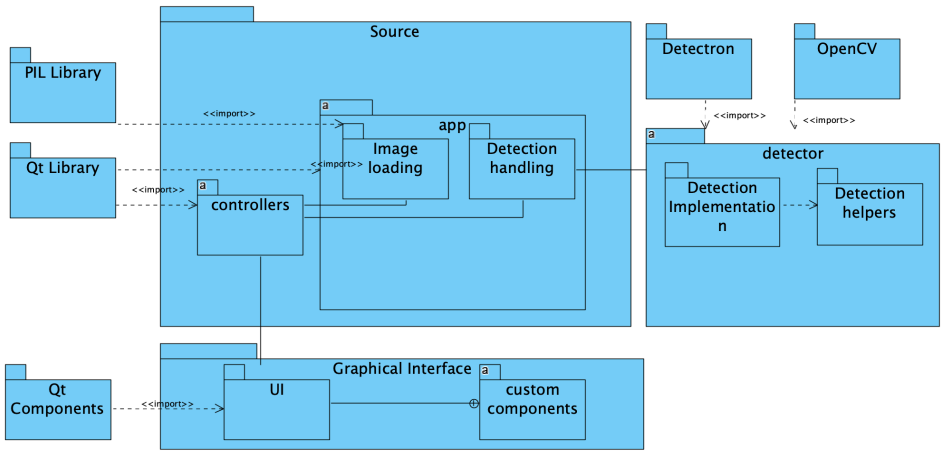

Figure 3.3: Application package diagram.

Figure 3.3 shows package model of the application packages with dependencies.

3.5 Choosing the suitable option object detection method

Concerning deep learning methods, it is apparent from the theoretical background section, that R-CNN would be too slow for the task. YOLO seems much more promising, but because some sources state that is not good with small objects [44]. I will also implement the Faster R-CNN to be able to compare them. I will also implement straightforward method without machine learning, using blob detection and thresholding, to see whether such solution could be reliable, especially when dealing with different environments.

Chapter 4

Realisation

Following chapter reports about the process of implementing the requirements specified in the previous chapter. The chapter starts with a description of the dataset collection, followed by a summary of the implementation stage. I will describe a whole implementation procedure, highlighting the interesting and important parts, and explaining used concepts. The realization of the application consists of creating GUI, input, and output handling and detection module, that will implement the detection algorithms.

4.1 Data Acquisition

Despite the great progress in general computer vision algorithms, such as detection and tracking, these algorithms are not usually optimal for dealing with sequences of images captured by drones, due to various challenges such as viewpoint changes and scales. However, studies toward this goal are seriously limited by the lack of publicly available large-scale benchmarks or datasets [45] [46]. Because of this, I have decided to create my own dataset for training and evaluation. I could not find a compatible dataset on the internet.

Thermal camera and drone for the data acquisition were provided by the company Workswell and company operator Miroslav Kleinbaurer handled the operation of the drone. We used the camera *Workswell Wiris Pro*, which is the thermal imaging system for UAV. It is a lightweight all-in-one system equipped with a thermal imaging camera and a visible spectrum camera [20]. The drone used was the *Matrice M600* by DJI.

Figure 4.1: Camera and Drone setup used for the data acquisition [47].

The first session was held on 25 January 2020 near Prague. We captured about 200 thermal and digital images with a drone. Scenes were set in forest, field, or rocks environments and covered one to five persons. Persons were positioned into varied postures and body positions, such as running, laying on the ground, or crouching. The drone flew in height from 30 to 50 meters above the ground. The resulting data are in the form of three images captured at the same time: thermal Joint Photographic Experts Group (JPEG), thermal Tagged Image File Format (TIFF), and visible JPEG. All three will be used.

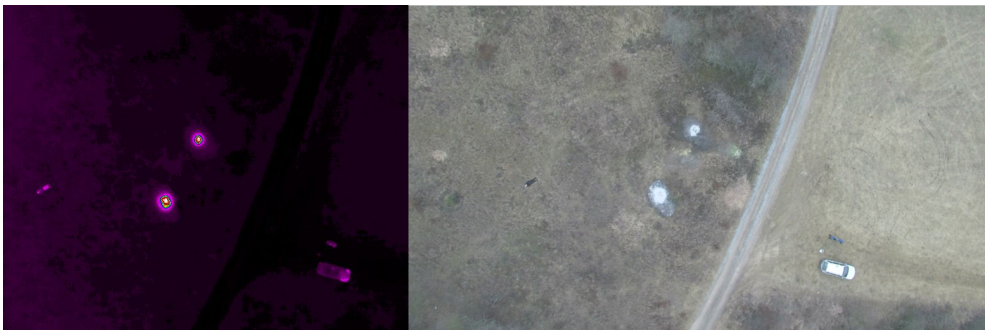

Figure 4.2: Example of captured data, both visible and thermal image.

The second session took place also in Prague in May of 2020. This time we focused mainly on taking pictures of people in greenery and fields. As the first session was done during the winter, the second session was necessary to be able to test the adoption of algorithms to changing temperatures of the environment and seasonal changes in the country. Captured images proved that it was easier to hide from the camera's sight in vegetation. Silhouettes

were also sometimes irregularly shaped because they were covered by dense trees or bushes.

Figure 4.3: Example of captured data from session 2.

Figure 4.4: Setup used during session 2.

After the image acquisition, data need to be sorted and interpreted. After duplicate or irrelevant pictures were deleted, I marked the positions of objects manually, creating annotations. To create simple bounding box annotations,

I used a tool called LabelImg for Python, which is open source and accessible using public repository [48]. Annotations are saved in the XML file format or YOLO specific format, which is a text file, one file per each image. These annotations will be used later for the training or evaluation in a ratio of approximately 80 to 20. When the data were captured and ready to be used, I advanced to the application.

4.2 Application Overview

To implement the project I decided to use *Qt Creator Integrated Development Environment (IDE)*. I used its project wizard do create *PySide2* Application with *Qt Widgets*. IDE provides tools for running, debugging, and deploying *PySide2* projects. Files in the project are divided into several directories. Python source files are stored in the directory *source*, *.ui* files in directory *ui*. Directory *detection_helper_files* contains required resources for detectors, such as trained YOLO weights. At the root of the project, there is a configuration file with a list of all the files and one Python source that holds the main application procedure within the `ThermalDetector` class. This class wraps the whole application and contains the main routine. The application is launched by the run method, which calls utility methods to initiate the GUI, backend, and signal-slot connections. I also included test dataset in the project. The directory *test-dataset* contains sources from both sessions. These images were not used for training.

4.2.1 Project Structure

Python source files are further organized into subdirectories. Directory *app* contains backend classes that use the *Qt Meta-Object System* and are an integral part of the Qt application. `RadiometricImage` encapsulates all the data about one image, `ImageList` holds these image instances. There are also handler classes implementing tasks such as a file or palette loading. Classes in the directory *controllers* are used to control graphical components by handling user inputs and presenting data from backend classes. *Detector* directory contains detection implementations, which all implement the `DetectorInterface`, together with helper classes for image processing. These are implemented without Qt libraries and can be used with Python. More information about all the modules will be provided in the following sections.

4.3 Creating Graphical User Interface

The *PySide2* framework offers very convenient way to create user interfaces. I used the *Qt Designer* to simply draw the required components without writing code. These components are called *Widgets*. Example of such components

are: `label`, `PushButton`, `ComboBox`. The interface consist of several separate .ui files. Each of these files can be modified in the Designer application, which then converts the component and layout to code. Programmers can compose and customise windows or dialogs in a what-you-see-is-what-you-get manner, and test them using different styles and resolutions. Widgets and forms created with *Qt Designer* integrate seamlessly with programmed code, using Qt's signals and slots mechanism, so that programmers can easily assign behaviour to graphical elements. [49]

4.3.1 Application Window design

My interface is inspired by known and well established layout of almost all the image viewing applications, for example *MacOs Preview* [50]. It will consist of one window. Creation of the window content starts with outlining the main layout. Center part of the interface is the view of a current image with a toolbar located above the image. On the right side I have a panel with tab view. There are three tabs, from the left: list of loaded files, file information, detection. Lastly, application offers menu with file loading options, which are also accessible using keyboard shortcuts. In my application, I have two *ui* files. *MainWindow* file holds the application window. The window is filled with *CentralWidget*.*CentralWidget* file contains whole layout as described.

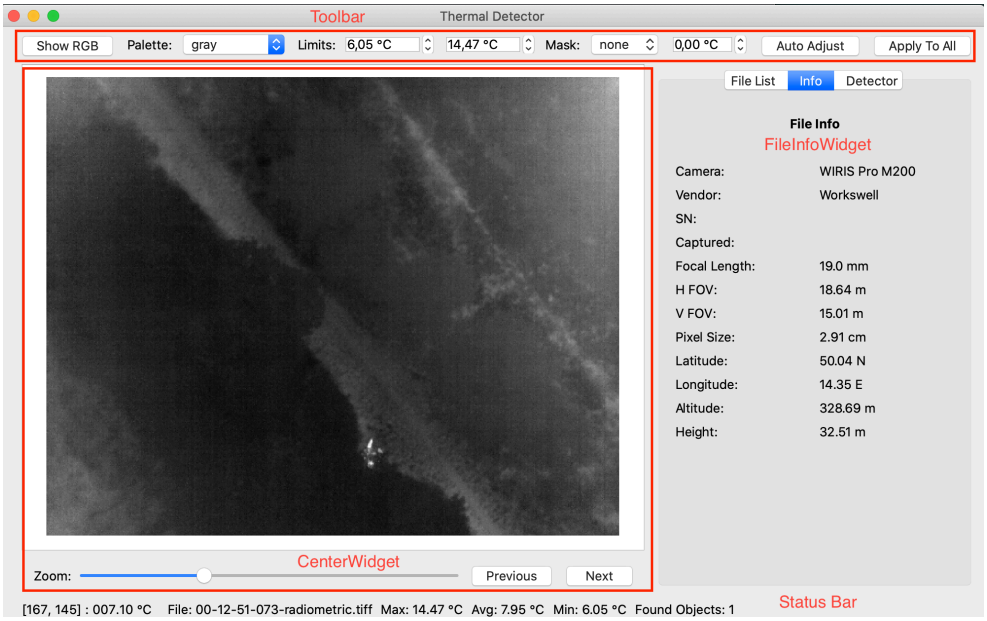

Figure 4.5: Application GUI overview with distinction of the most important parts.

Components persist a native look on a platform used.

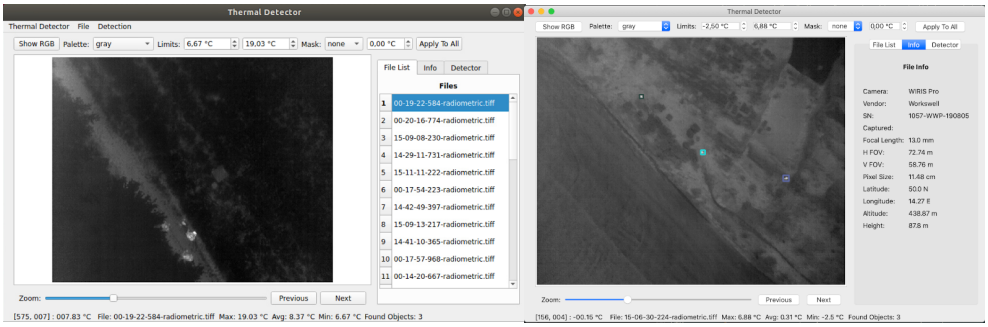

Figure 4.6: Application GUI with native look, Linux on the left and MacOS on the right.

4.3.2 Custom Graphic Components

Customisation and modification of behaviour of graphic components in Qt is enabled by subclassing widgets from the *QtWidgets* library. This is useful e.g. for implementation of mouse or other input events. In my application I will subclass `QGraphicsView` and `QGraphicsScene` classes and override methods to make it suitable for displaying both thermal and visible images, which have different resolution and aspect ratio. I will override `mouseMoved` method to track mouse position on image and show the temperature of pixel under the cursor. Another subclassed component is the `ImageLabel`, which is the simple way to display image without zoom or mouse tracking. `Show-Pixmap` method is overriden. Because the application uses multiple threads, I need to implement handlers for safely terminating the running thread. For this purpose I subclassed `MainWindow`, reimplementing `closeEvent`, in which threads are notified about window being closed.

```
# image scene, reimplemented to enable mouse tracking and zoom
# draws the image given as numpy array with RGB data
class ImageScene(QGraphicsScene):
    mousePosChanged = Signal(int, int)

    def __init__(self, width, height):
        QGraphicsScene.__init__(self)
        self.__mapped_width = width
        self.__mapped_height = height
        self.setSceneRect(0,0, width, height)

    def draw(self, imarray):
        self.clear()
        self.__drawImage(imarray)

    def __drawImage(self, imarray):
        image = QImage(imarray, imarray.shape[1], imarray.shape[0],
                    imarray.shape[1] * 3, QImage.Format_RGB888)
        pixmap = QPixmap(image)
        self.addPixmap(pixmap.scaled(self.__mapped_width,
            self.__mapped_height));

    def mouseMoveEvent(self, event):
        self.mousePosChanged.emit(event.scenePos().x(),
            event.scenePos().y())
```

Listing 4.1: Subclassing GUI Component.

4.4 Application Implementation

The implementation consists of classes with given responsibilities. These classes make up logical sets, according to the service they implement. I divided the services into controllers, handling interaction with the GUI, classes for image manipulation, and data storage and detection itself. The code follows Python guidelines and conventions, for example, maximum line length, indentation, blank lines. I used guidelines from the document [51], which gives coding conventions for the Python code comprising the standard library in the main Python distribution. Classes are named with capitalized *CamelCase*, methods, and instance variables with non-capitalized *underscore notation*. Because Python does not contain access modifiers, I decided to use mangling to mark private methods. Mangling is noted with two underscores before the name, and it prevents accidental overrides.The wrapper class for the whole application is the `ThermalDetector` class. On the startup, command-line arguments are processed and `QApplication`, which is the Qt application class, is launched. `BeanPool` is the container for all the instances that are created during application runtime. It is a *singleton* class, so only one instance exists at all the time. This way, the instances inside the pool can be accessed using the singleton from anywhere in the application without the need to hold

references. `BeanPool` also manages connections between the instances.

4.4.1 GUI Controllers

After creating the Graphical Interface, the next step is to create controller classes that will handle the user interaction with the interface and contribute to the MVC architecture for presenting data to GUI. I have decided to create one controller class per logical component of the interface. These controller classes implement handler methods, that are called when certain events occur, for example, click on a detect button will trigger respective handler, which will notice the `DetectorHandler` instance to fetch the data and start the algorithm. Furthermore, when the detection finishes, the detector class will notify the corresponding controller to present the result to the frontend. The controller classes serve as a connection between backend and graphical interface components. Object communication using *Qt Signals and Slots system* [42].

```
# define signal
image_selected = Signal(object)
# connect to graphical component signal
self.__image_table.cellClicked.connect(self.__handle_table_click)
# emits signal image_selected when user clicks the row in a table
@Slot(object, object)
    def __handle_table_click(self, row, column):
        self.image_selected.emit(row)
...
# connection to select_image method of image list
 self.__image_list_controller.image_selected.connect(
           self.__image_list.select_image)
```

Listing 4.2: Signals and Slots demonstration.

There are 6 controller classes:

- `MainWindowController` - handling actions from main window menu, like loading and saving files,
- `CentralWidgetController` - drawing image, mouse tracking, image zoom,
- `ToolbarWidgetController` - image visualization corrections from the upper toolbar,
- `DetectorWidgetController` - triggering detection and setting parameters, presenting results,
- `ImageListWidgetController` - loaded files list, current image selection,
- `FileInfoController` - displaying selected file metadata.

Each of these handles action from the respective graphic components and propagates the actions towards backend classes. On the program startup, *.ui* files are loaded using class `QUiLoader`. The `QUiLoader` class provides a collection of functions allowing programmers to create widgets based on the information stored in UI files [52]. As a result, parent widget of the file is accessible as a Python class and its components can be connected to signals and slots of backend classes. To approach specific components, such as buttons, `findChild` method of parent component can be used.

```
# loading components from ui file created by Qt Designer
ui_loader = QtUiTools.QUiLoader()
form = ui_loader.load(QFile(filename))
# find button
self.__run_button = centralWidget.findChild(
    QPushButton, "runButton")
```

Listing 4.3: Loading *ui* files into Python code.

4.4.2 Image handling

To carry out access to the file system and loading files I used standard Python libraries. The responsibilities are delegated to several classes. `FileHandler` class contains methods for loading images using the *PIL* library, as well as saving them. An instance of this class exposes public methods for loading image or directory, as well as saving them. *PIL* library provides extensive file format support and is designed for fast access to data stored in a few basic pixel formats [53]. I use it to convert TIFF images to *NumPy* arrays directly. *NumPy* is the fundamental package for scientific computing with Python [53]. *NumPy* arrays provide a very convenient way for data storing and manipulation. Data are stored in a two-dimensional array, each pixel is represented by one 16 bit number. This array is loaded directly from the image. A pixel value is convertible to degree celsius using simple formula:

$$temperature = pixel\_value/25 - 100 \tag{4.1}$$

For the internal representation of the image, I constructed class `RadiometricImage`, with pixel data in *NumPy* arrays and image metadata. Metadata are obtained from the Exchangeable Image File Format (EXIF) tag of the JPEG image. *PieExif* is another library I used, this time to read the EXIF tags. It includes the tools necessary for extracting and manipulating and writing EXIF data to both JPEG and TIFF files [54].`FileHandler` is in fact a factory for `RadiometricImage` instances. `RadiometricImage` instances are stored in `ImageList` throughout the run of the application. `ImageList` manages loaded sources and holds the reference to the currently selected source. After the detection, the result can be saved in a JavaScript Object Notation (JSON) file. This notation can be later used to transport data using web API.

4.4.3 Visualizations

Settings for visualizations of the measured temperatures can be altered using the toolbar. I have included several palettes commonly used in thermography software [13]. There are also options to change minimum and maximum temperature used, or mask pixels above or bellow certain threshold. Calculation of the colors for each pixel is implemented by `RadiometricImage`.

Using right visualization can have a significant impact on the accuracy of the detection. I have created `auto_adjust` method to automatically adjust colors and prepare for the detection. This is done by applying analysis of the temperatures, suppressing pixels bellow the mode value and above certain threshold, given by maximum skin temperature. User can calibrate the settings on one image and then apply it on all the images, for example when analyzing images collected during one session.

4.4.4 Multithreading

The application uses threads to implement demanding tasks which could take longer time to complete. Qt framework offers multithreading support with its `QThread` class. "Instantiating a `QThread` provides a parallel event loop, allowing `QObject` slots to be invoked in a secondary thread"[55]. This means that during the completion of a heavy task, primary thread remains active, so user can be informed about a progress or cancel a task. I created 3 different `QThread` subclasses, one for loading multiple images, one for running a detection on more images and finally, one for batch processing loaded images. Data are loaded to thread instance by constructor. textttrun_flag is used for control the flow, and thread can be stopped using textttstop method. Calling this method will place it in the parallel event loop and will be planned to execute accordingly[56]. This way, there is no need for synchroziation primitives, like texttQMutex.

```
# worker for setting min/max, mask to more images
class ToolbarWorker(QtCore.QThread):
    updateProgress = QtCore.Signal(int)
    workDone = QtCore.Signal()

    def stop(self):
        self.__run_flag = False

    def __init__(self, list, pal, min, max, mask_type, mask):
        QtCore.QThread.__init__(self)
        self.__list = list
        self.__pal = pal
        self.__max = max
        self.__min = min
        self.__mask_type = mask_type
        self.__mask = mask
        self.__run_flag = True

    def run(self):
        i = 0
        for image in self.__list:
            if not self.__run_flag:
                self.workDone.emit()
                return
            self.updateProgress.emit(i)
            i += 1
            image.set_range(
                self.__min, self.__max, self.__pal,
                self.__mask_type, self.__mask)
        self.workDone.emit()
```

Listing 4.4: Thread-derived worker class.

4.5 Detection Implementation

Detecting humans and animals in thermal imagery poses additional challenges such as lower resolution, halos around hot or cold objects, and smudging artifacts in case of camera movement [40]. To overcome these challenges, I decided to combine various approaches and do comparisons of their accuracy and performance.

I defined `DetectorInterface`, which is the base class for the detector implementations. This way, it has well defined basic functionality and the particular implementations can be used interchangeably. Base class contains methods for loading the input data and setting parameters. Another method is for for triggering the detection. In it, input images are iterated over and for each one, an abstract method `_detectImage`, is called. Each type of detector will be realise this method differently. Because Python itself does not contain strong inheritance mechanisms, I decided to use the library abc to mark method abstract and prevent instances of `DetectorInterface` [57].

The `DetectorInterface` and all the derived classes are implemented in such a way that enables their usage outside Qt. There are no Qt dependencies, so these core classes can be used in future projects independently.

In an application class model, class `DetectorHandler` manages the detection and holds collection of available detectors. When a user triggers the detection, selected `RadiometricImage` instances are loaded to the desired detector instance and the algorithm starts. `DetectorInterface` controls the algorithm flow and calls support procedures, that are divided across support classes. The separation was done according to single responsibility and low coupling principle, so individual tasks can be modified or replaced.

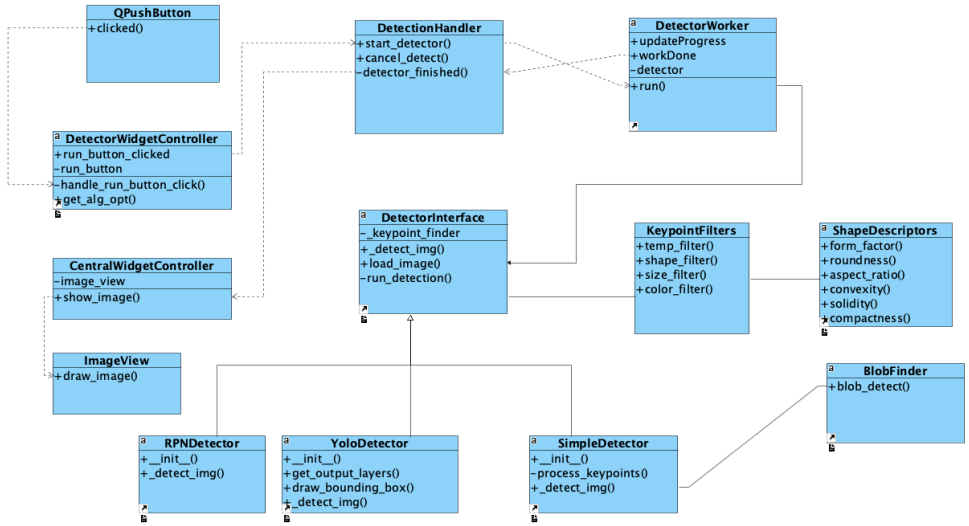

Figure 4.7: Part of the class model showing the situation with the detection management. Dashed line notes signal-slot connection.

First, I start with a procedural routine for blob detection, that analysis the data and computes results based on given temperature thresholds, shape descriptors and other features. Then I will take a step towards machine and deep learning principles, take advantage of pre-trained networks, and open source systems.

4.5.1 Blob detection algorithm

For a procedural analysis, an algorithm with three stages is proposed. In the first stage, I will analyze the captured data numerically, then I will apply image segmentation to the foreground and background and finally, classify the candidate objects.

The environment temperature is important for the task, as it affects the target object's temperature, as shown in chapter 2. It is important to gain as much information about the scene as possible before running the detection itself. Some of the information can be extracted directly from EXIF saved to JPEG source, such as Global Positioning System (GPS) location. As the drone is equipped with a GPS locator, the altitude of the drone can be used to calculate height above the ground. I used *Google Maps API* [58] to find elevation by calling their web service with coordinates saved in TIFF It is free to use with a limited number of requests per day, which is sufficient for me. By subtracting the elevation from altitude saved by the drone locator, we get height above the ground. All these parameters can be also changed by a user.

When all required information had been gathered, the second part of the detection can launch. It is a task to split the scene into the background and foreground with candidate objects. Such objects are called blobs. A Blob is a group of connected pixels in an image that share some common property [59]. For his purpose, I used a blob detection algorithm by *OpenCV*, implemented by class `KeypointFinder`. *OpenCV* provides a convenient way to detect blobs and filter them based on different characteristics. [59].

The class implements a simple algorithm for extracting blobs from an image [60]:

1. Convert the source image to binary images by applying thresholding with several thresholds from `minThreshold` (inclusive) to `maxThreshold` (exclusive) with distance `thresholdStep` between neighboring thresholds.

2. Extract connected components from every binary image by `findContours` and calculate their centers.

3. Group centers from several binary images by their coordinates. Close centers form one group that corresponds to one blob, which is controlled by the `minDistBetweenBlobs` parameter.

4. From the groups, estimate final centers of blobs and their radiuses and return as locations and sizes of keypoints.

After the blobs were detected, I will work out the temperatures of them and filter them accordingly. Blobs with a temperature too high or too low are removed instantly. After this stage, several candidate areas will remain that will be processed in the next part.

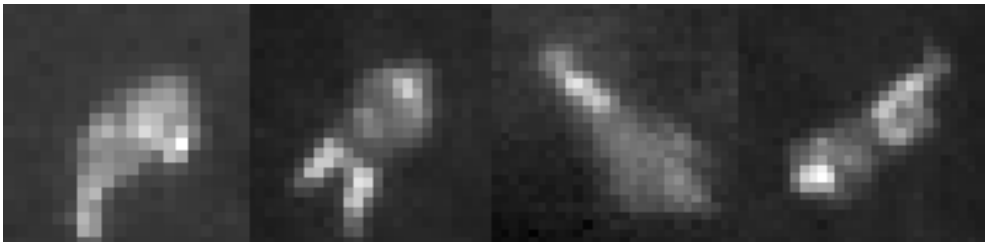

Figure 4.8: Examples of blobs or candidate areas.

Figure 4.5 shows human silhouettes on thermograms captured in height around 50 meters. No details of the subjects are visible, although the shape resembles a human body with one ax significantly shorter. The shape of a silhouette varies because of the angle of sight, from directly above, the shape is more circular. Even with this combination, the shape is distinguishable from another object that may appear on the scene, as shown in figure 4.6.

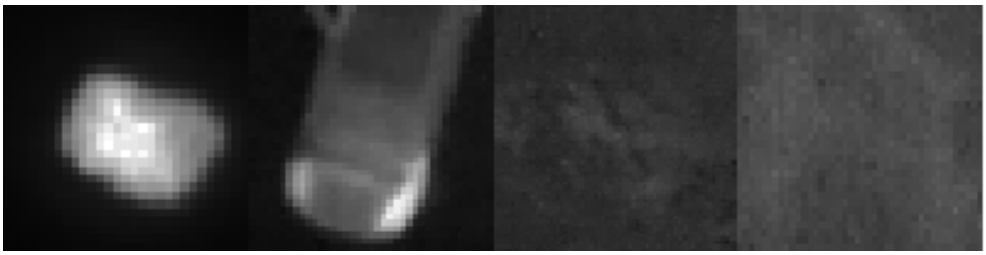

Figure 4.9: Examples of blobs that are false positives.

Figure 4.6. shows examples of areas, that may be part of the results of blob detection. These include a car, a fireplace, and a terrain or other features. Most of these can be filtered out easily, but others may posses a great challenge.

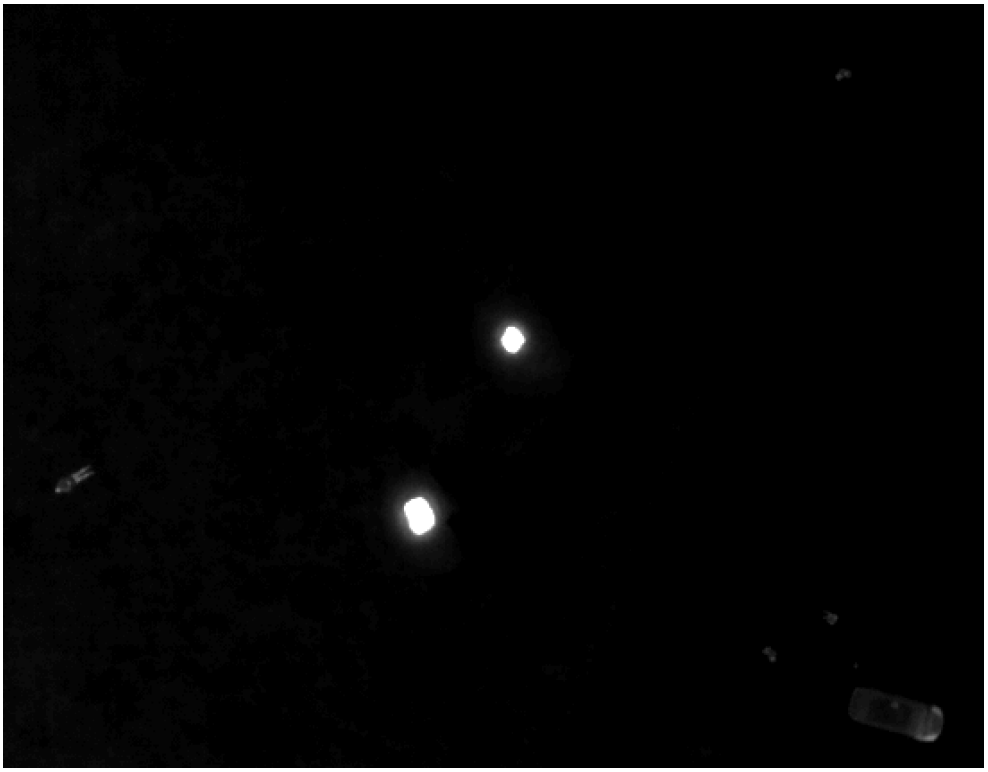

Figure 4.10: Foreground Extraction.

To verify whether the candidates from the previous step belong to classes we look for, I implement various filters based on size, shape, and color. Filtering by temperatures works by determining upper and lower temperature bounds. Because the temperature of a surface of a body varies with an environment, it is important to base these calculations on the current situation captured on a scene. By statical analysis, I define environment temperature, which will be used as a lower bound. Upper bound will be set by a user. Another type of filter is the size filter. The apparent size of an object depends on distance and type of lens. The thermal camera can have lenses with various focal lengths equipped. The full list of lenses was given by Workswell. Focal length are saved in an EXIF file, so from this information, I can work out pixel size and thus minimum and maximum area in pixels.

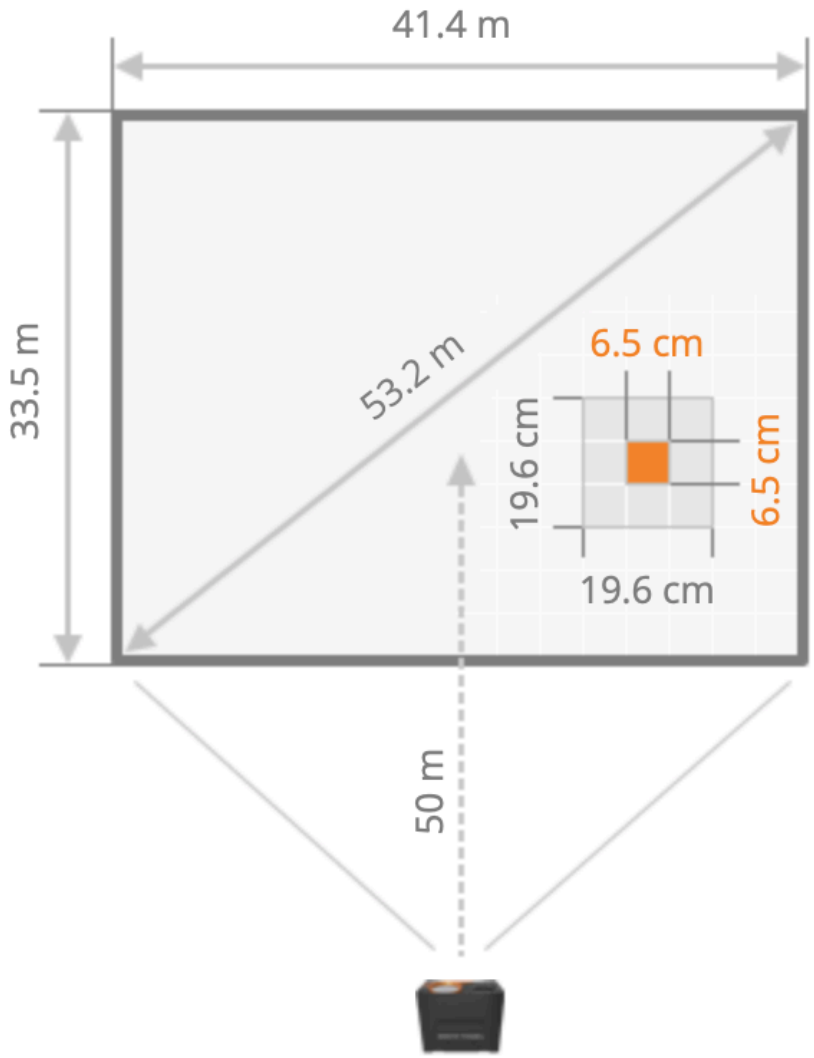

Figure 4.11: Field of view of the camera 50 meters above ground[61].

The final filter assesses the shapes of objects. To be able to examine the shapes, exact contours need to be found. For this task, I will again use the *OpenCV* library. Contours are a curves joining all the continuous points along the boundary. The full set of characteristics commonly used to describe shapes was formulated by John C and F. Brent [62]. Here I list the ones that I will use:

- Formfactor,
- Roundness,

- Aspect Ratio,
- Convexity,
- Solidity,
- Compactness,
- Extent.

Analysis by shape is required to thermal traces, because generally speaking, the human body has a round silhouette. However, there may be certain cases in which shape may be irrelevant, for example when part of the trace is covered by plants. Also, traces of moving bodies have irregular shapes. I will use shape descriptors mainly to filter out obvious false detections.

Resulting blob detection procedure is just partially successful, as it detects many false positives and can hardly cope with the changing environment. Filters helped to suppress false positives, however, the overall accuracy is not high enough. Shape Filters can be altered by setting parameters in file `global_defs`.

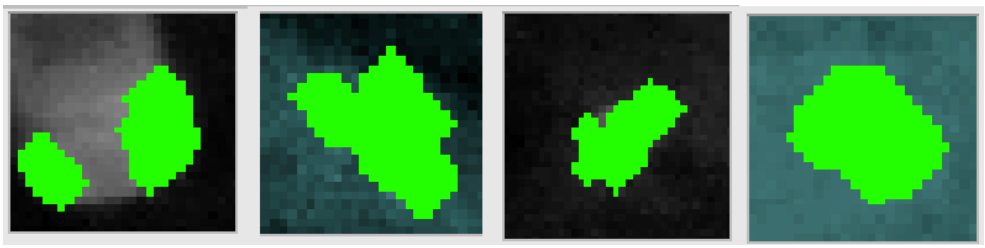

Figure 4.12: Blobs with masked contours. Left one is a front part of a car, another 3 are people.

4.5.2 YOLO Detector

As stated in a theoretical background chapter, YOLO is one of the most popular detection methods with very good results. I will use You Only Look Once version3 (YOLOv3) implementation, which has improved accuracy. To try it on thermograms, I created class `YoloDetector`, implementing the `_detect_image()` method. All the support files are located in a folder *app_root/yolo3*. These are config files, weights, and class labels [63]. Config file holds the parameters used for the particular model. I used pre-trained COCO at first, but as the thermograms look significantly different from RGB images, it was clear that the model needs to be re-trained. For this purpose, I used captured data, with images split to training and control parts. For the training, I used the *Darknet framework*. I have chosen YOLOv3 implementation. Description

of the algorithm and full workflow is summed up by Manivannan Murugavel [64]. To sum up, this particular implementation uses 53 convolutional layers with filter size 3x3 and 1x1 The network is fully described in a paper by it's creators [65]. The paper also includes comparison of speed and accuracy of various networks, e.g. RetinaNet. YOLOv3 is very fast and still maintains good precision.

The training was done in ImproLab with *NVidia RTX 1080* graphics card used the technique called transfer learning, which means using pre-trained weights and apply another training session with custom data. I prepared a set of 400 thermal images from drones with annotations. Then I split the data into training and test sets. I used the default YOLOv3 config file which determines the number of layers used. Usage of the trained weights is implemented by `YOLODetector`. Three files are required for detection, these are weights file, configuration file, and object category list. `YOLODetector` uses *OpenCV* to read trained model using function `cv2.dnn.readNet`. Then input blob is created from the image using `cv2.dnn.blobFromImage` and set as input for net created in a previous step. Then detector runs inference through the network and gather predictions from output layers using `net.forward(self.get_output_layers(net))`. For each detection from each output layer, algorithm outputs get the confidence and bounding box position. Detections with a confidence lower than 0.5 are considered weak detection and are ignored. Detected objects are then saved to image instance and drawn on the image.

4.5.3 Faster R-CNN Detector

To test how Faster R-CNN would perform with thermograms, I decided to use implementation which is part of *Detectron2* [66]. *Detectron2* is Facebook AI Research's software system that implements many state-of-the-art object detection algorithms. *Detectron2* is powered by the *PyTorch* framework. *PyTorch* is an open-source machine learning library that enables Graphics Processing Unit (GPU)-accelerated deep neural network programming[67].To start the training on thermal images, I first need to prepare a dataset for this particular implementation by changing annotation files and creating the required structure. For this purpose, I will create a Python script that prepares the data and runs the training. Annotations for training are given in absolute coordinates, not by ratios as in the case of YOLO. In the process, I will use the detectors `DefaultTrainer` class and then interpret the result using the `DefaultPredictor` class. Full process of the training and inference is described in a tutorial by *Detectron2* creators [68].

4.5.4 Presentation of results

After the detection was complete by one of the detectors, results are shown in a table. Each table entry contains position of the bounding box, and in-

formation about temperatures. Entries can be also selected for displaying the RGB data of that box to small image view bellow the table.

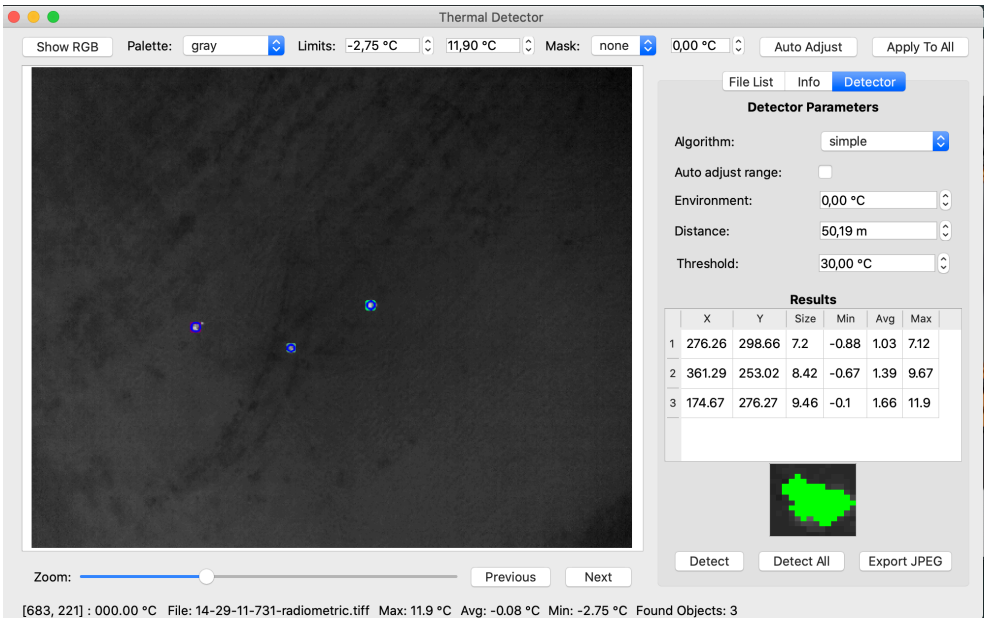

Figure 4.13: Application window displaying results of a detection.

Chapter 5

Testing and detection evaluation

In this chapter, I will describe how the application will be tested, with the emphasis on testing the accuracy of the detection. I will compare the results from the application with the control part of the dataset.

5.1 Detection testing

To test the success rate of the detection I wrote a simple Python script. This script takes a directory path as an input parameter. For each tiff image entry, the script tries to load the original annotation, JSON file with objects detected by an application, and compares them, calculating how individual entries overlap and outputs the total detection score. Despite that this script is not a part of the application itself and will not be available to users, I decided to develop it within the application project. Qt Creator enables running each Python project script with the concept of *Run configurations*. My project will have the main configuration called *thermal detector*, which runs the application and second configuration called *thermal detector-test*, which runs the test script. For each image entry found in a given directory, two files are read. One of them is a solo annotation created in LabelImg and the other is a JSON with serialized data from the application. Yolo annotation contains one line per object and coordinates are fractions of the image size, so values range from 0 to 1. These values need to be calculated for real image coordinates. Detection results are serialized and saved to JSON file by the application. Saved annotation file contains a list of objects representing bounding boxes definitions. Parameters of these objects are top-left point coordinates, width, and height. The format is the same for all the detectors. To compare annotations with the detection result, I calculate how much area these bounding boxes have in common. Firstly I work out their intersection and then two ratios which are

an area of the first box divided by area of an intersection and are of the second one divided by intersection area. For each original annotation, the test procedure finds detected objects with the highest overlap and connects these boxes. If overlap exceeds a certain threshold, which is variable, detection is counted as successful. Then false positives and not detected objects are noted. After examining each file, overall statistics are printed to standard output.

5.1.1 Testing conditions

For testing, I used Central Processing Unit (CPU) inference on MacBook Pro 2015 with IntelCore i7, CPU inference on Linux computer, and GPU inference using NVidia RTX 2080 Graphic card on the same Linux machine in ImproLab. Before the training and implementation, I did randomly choose images for testing, that were not used for training. These images make up the primary test sample. I will also run the detection of all the images gathered.

5.1.2 Results

Test Run	Algorithm	Total Objects	Detections	False Positives	Accuracy
1	YOLOv3	807	658	59	81 %
1	Faster R-CNN	807	645	81	80 %
2	YOLOv3	60	54	5	90 %
2	Faster R-CNN	60	53	7	89 %
3	Blob Detection	60	36	16	60 %
3	YOLOv3	60	55	0	91,5 %
3	Faster R-CNN	60	51	7	85 %
3	Blob Detection	60	42	16	70 %
4	YOLOv3	23	22	2	95 %
4	Faster R-CNN	23	16	7	69 %
4	Blob Detection	23	12	10	52 %
5	YOLOv3	37	34	2	91 %
5	Faster R-CNN	37	37	3	100 %
5	Blob Detection	37	23	0	62 %
6	YOLOv3	807	682	26	84 %
6	Faster R-CNN	807	674	47	83 %
6	Blob Detection	807	554	71	68 %

Table 5.1: Detection results comparison.

I did several test runs with different data:

1. all the data with default visualizations,
2. test collection with default visualizations,

3. test collection, visualization correction by textttauto_adjust,
4. test collection, visualization correction by textttauto_adjust, images just from session 1,
5. test collection, visualization correction by textttauto_adjust, images just from session 2,
6. final set with all the data, visualization correction by textttauto_adjust

Results show, that YOLOv3 is even more accurate than a Faster R-CNN. This is a surprise to me, as I expected that the two-stage region proposal network would be more successful. Both deep learning algorithms have slightly better results when the visualization is adjusted before the detection, but the difference is very small. On the other hand, the number of false positives decreased significantly. The blob detection algorithm has considerably worse results, as the hard-coded feature detection is not as effective as the deep learning approach.

For test runs 4 and 5 the test data were into images captured during winter and spring. Results show, that generally, detection is more accurate in the winter environment, as the temperature differences are higher. Temperatures similar to human skin temperature cause more false positives and decrease success rate, mainly in the case of blob detection. This shows that results depends highly on the environment, but some of these problems can be tackled by setting right parameters for the visualization.

5.1.3 Performance

Device	Method	Platform	Inference Time
CPU	YOLOv3	MacOS	1.3 - 1.5 sec
CPU	Faster R-CNN	MacOS	7.5 - 8.2 sec
CPU	Blob Detection	MacOS	0.7-1.0 sec
CPU	YOLOv3	Linux	0.8-0.9 sec
CPU	Faster R-CNN	Linux	2.2-2.4 sec
CPU	Blob Detection	Linux	0.3 sec
GPU	YOLOv3	Linux	0.08-0.11
GPU	Faster R-CNN	Linux	0.9 sec

Table 5.2: Detection speed comparison.

Performance tests prove that YOLO is the fastest, with the maximum performance it can handle images at 10 frames per second. If the visualization is being altered before the detection, it may slow down the process. For real-time detection, visualization would have to be adjusted on the camera, not in the detection module.

Chapter 6

Conclusion

In my thesis, I have shown that it is indeed possible to use object detection methods with thermograms. The developed application realizes a simple interface to the detection module, which was build based on the analysis of image detection methods. I took advantage of existing image detection implementations and systems, which I integrated into my solution. I introduced a disciple of Thermography, explaining the important concepts and obstacles behind.

I achieved significant accuracy using deep learning and transfer-learned neural networks. I have compared the performance and accuracy of neural networks with different architecture and also a blob detection algorithm based on color, shape, and temperature analysis.

The results show that the YOLO algorithm is not only the fastest, when running on GPU, but also the most accurate. I perceive it as a suitable option for future development. The network was developed and tested using only about 400 images and it may be prone to overfitting. Larger datasets that would cover more real-world situations and areas are required to train the network. Tests also proved that the environment conditions, like air temperature and humidity, as well as vegetation have a significant impact on the accuracy. To combat all these challenges, netwoir would have to be build on a larger dataset.

For me, this thesis represented a great challenge. It was demanding not only as an implementation task but as a whole, as all the parts from data acquisition to analysis had a big impact. As I did not have any prior experience with Python programming language or any image processing technology, the understanding presented concepts widen my horizons, especially the research on CNNs.

The real-world usage of the proposed solution may be limited due to the fact, that thermal cameras are still a high-profile product and the user base is not so high. On the other hand, I see big potential in the cooperation of the company Workswell, which already showed an interest in future development.

A complete solution would have to be integrated with the camera firmware, sending images to an application deployed to the server, which would run the YOLO detector on GPU. It is not possible to run the inference directly on the camera computing unit.

Bibliography

[1] Sova, J. Bezdotykové měření teplotních polí I. *Aldebaran Bulletin [online]*, 2017, [cit. 2020-05-08]. Available from: https://www.aldebaran.cz/bulletin/2017_18_ter.php

[2] Skalický, M. České nemocnice budou monitorovat termokamery. Jak fungují a nedají se zneužít? (Czech). *"Český rozhlas [online]*, 2020, [cit. 2020-05-31]. Available from: https://radiozurnal.rozhlas.cz/ceske-nemocnice-budou-monitorovat-termokamery-jak-funguji-a-nedaji-se-zneuzit-8197374

[3] Jones, B. A reappraisal of the use of infrared thermal image analysis in medicine. *IEEE Transactions on Medical Imaging [online]*, 1998, [cit. 2020-05-10]. Available from: https://ieeexplore.ieee.org/abstract/document/746635

[4] Gaussorgues, G. *Infrared Thermography*. Dordrecht: Springer Science+Business Media, third edition, 1994, ISBN 978-94-010-4306-9.

[5] International Association of Medical Thermographers. History of Thermography [online]. [cit. 2020-05-31]. Available from: https://iamtonline.org/history-of-thermography/

[6] WikiSkript. *Termografie [online]*. 2018, [cit. 2020-05-08]. Available from: https://www.wikiskripta.eu/index.php?title=Termografie&oldid=407681

[7] Van Hoof, C.; Moor, D. P. *Handbook of Infra-red Detection Technologies*. Elsevier Science, 2002, ISBN 978-1-85617-388-9, [cit. 2020-05-31]. Available from: http://www.sciencedirect.com/science/article/pii/B9781856173889600012

[8] Sova, J. Screening horečnatých stavů pomocí termokamery [online]. 2020, [cit. 2020-05-11]. Available from: https://www.slideshare.net/

workswellEU/screening-horenatch-stav-pomoc-termokamery-koronavirus

[9] Modest, M. *Radiative Heat Transfer*. California: Academic Press, second edition, 2009, ISBN 9780123869906.

[10] Inc., E. O. The Correct Material for Infrared (IR) Applications [online]. 2018, [cit. 2020-05-15]. Available from: https://www.edmundoptics.com/knowledge-center/application-notes/optics/the-correct-material-for-infrared-applications/

[11] Bierman, W. The Temperature of the Skin Surface. *Journal of the American Medical Association [online]*, 1936, [cit. 2020-05-24]. Available from: 10.1001/jama.1936.02770140020007

[12] Lai, D.; Zhou, X.; et al. Measurements and predictions of the skin temperature of human subjects on outdoor environment. *Energy and Buildings [online]*, volume 151, 2017: pp. 476 – 486, ISSN 0378-7788, doi:https://doi.org/10.1016/j.enbuild.2017.07.009, [cit. 2020-05-24]. Available from: http://www.sciencedirect.com/science/article/pii/S0378778817305601

[13] Workswell. *Core Player [software]*. [cit. 2020-05-31]. Available from: https://workswell-thermal-camera.com/workswellcoreplayer/

[14] FLIR Systems Inc. *FLIR Tools+ [software]*. 2020, [cit. 2020-05-31]. Available from: https://www.flir.com/products/flir-tools-plus/

[15] LaRue, M. Satellite imagery can be used to detect variation in abundance of Weddell seals (Leptonychotes weddellii) in Erebus Bay, Antarctica. *Polar Biology [online]*, 2011, [cit. 2020-05-11]. Available from: https://ieeexplore.ieee.org/abstract/document/746635

[16] Hodgson, J. Drones count wildlife more accurately and precisely than humans. *Methods in Ecology and Evolutions [online]*, 2018, [cit. 2020-05-11]. Available from: https://ieeexplore.ieee.org/abstract/document/746635

[17] Chrétien, L. P.; Jérôme, T.; et al. Wildlife multispecies remote sensing using visible and thermal infrared imagery acquired from an unmanned aerial vehicle (UAV). *International Conference on Unmanned Aerial Vehicles in Geomatics [online]*, 2015, [cit. 2020-05-10]. Available from: https://www.researchgate.net/publication/281462647_Wildlife_multispecies_remote_sensing_using_visible_and_thermal_infrared_imagery_acquired_from_an_unmanned_aerial_vehicle_UAV

[18] Czech Republic Civil Aviation Authority. *Letecké předpisy[online].* [cit. 2020-05-12]. Available from: https://www.caa.cz/dokumenty/predpisy/letecke-predpisy/

[19] Coach, U. Drone Laws in Czech Republic [online]. 2015, [cit. 2020-05-12]. Available from: https://uavcoach.com/drone-laws-in-czech-republic/

[20] Workswell. *Workswell Wiris Pro User Manual [online].* 2019, [cit. 2020-05-11]. Available from: http://workswell-thermal-camera.com/docs/UAV_WWP_UM.pdf

[21] IT Network. *Úvod a motivace do programování neuronových sítí v Pythonu [online].* 2019, [cit. 2020-05-13]. Available from: https://www.itnetwork.cz/python/neuronove-site/uvod-a-motivace-do-programovani-neuronovych-siti-v-pythonu

[22] Sun, Z.; Bebis, G.; et al. Monocular precrash vehicle detection: Features and classifiers. *Proce. IEEE Trans. Image Processing [online]*, 2006, [cit. 2020-05-13].

[23] Ujjwalkarn. An Intuitive Explanation of Convolutional Neural Networks [online]. 2016, [cit. 2020-06-01]. Available from: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[24] Browniee, J. A Gentle Introduction to the Rectified Linear Unit (ReLU) [online]. 2019, [cit. 2020-06-04]. Available from: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[25] Fully Connected Layers in Convolutional Neural Networks: The Complete Guide [online]. [cit. 2020-06-01]. Available from: https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/

[26] Singh, S. P. Fully Connected Layer: The brute force layer of a Machine Learning model [online]. [cit. 2020-06-04]. Available from: https://iq.opengenus.org/fully-connected-layer/

[27] Girshick, R. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision [online]*, 2015: pp. 1–9, [cit. 2020-05-11].

[28] Ren, S.; He, K.; et al. Faster R- CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems [online]*, 2015, [cit. 2020-05-13].

[29] Redmon, J.; Divvala, S. You only look once: Unified, real-time object detection. *IEEE conference on computer vision and pattern recognition [online]*, 2016: p. 779–788, [cit. 2020-05-12].

[30] Mittal, U.; Srivastava, S.; et al. Object Detection and Classification from Thermal Images Using Region based Convolutional Neural Network. *Journal of Computer Science [online]*, 2019, [cit. 2020-05-13]. Available from: `https://thescipub.com/pdf/10.3844/jcssp.2019.961.971.pdf`

[31] Uijlings, J. Selective Search for Object Recognition [online]. 2012, [cit. 2020-06-01]. Available from: `https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf`

[32] Gandhi, R. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [online]. 2018, [cit. 2020-06-01]. Available from: `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`

[33] Zhang, A.; Lipton, Z. C.; et al. *Dive into Deep Learning*. 2020, `https://d2l.ai`.

[34] Pobar, M. Human Detection in Thermal Imaging Using YOLO. 2019, [cit. 2020-05-12]. Available from: `https://www.researchgate.net/publication/333360405_Human_Detection_in_Thermal_Imaging_Using_YOLO`

[35] Tsung-Yi L, M. M. B. S. J. Microsoft COCO: Common Objects in Context. *CoRR*, volume abs/1405.0312, 2014. Available from: `http://arxiv.org/abs/1405.0312`

[36] Caelli, T.; Bischof, W. T. *Machine Learning and Image Interpretation*. Springer Science & Business Media, third edition, 1997, ISBN 978-1-4899-1818-5.

[37] Tang, P.; Wang, X. Weakly Supervised Region Proposal Network and Object Detection. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[38] Mallick, S. Histogram of Oriented Gradients [online]. 2016, [cit. 2020-05-13]. Available from: `https://www.learnopencv.com/histogram-of-oriented-gradients/`

[39] FLIR Systems Inc. Free FLIR Thermal Dataset for Algorithm Training [online]. 2019, [cit. 2020-05-10]. Available from: `https://www.flir.com/oem/adas/adas-dataset-form/`

[40] Doherty, P.; Rudol, P. Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery. *2008 IEEE Aerospace Conference [online]*, 2008, [cit. 2020-05-10]. Available from: https://ieeexplore.ieee.org/document/4526559

[41] Corcoran, E.; Denman, S.; et al. Automated detection of koalas using low-level aerial surveillance and machine learning. *Sci Rep [online]*, 2019, [cit. 2020-05-10]. Available from: https://doi.org/10.1038/s41598-019-39917-5

[42] Nokia Corporation. *Qt Reference Documentation [online]*. [cit. 2020-05-10]. Available from: https://doc.qt.io/archives/qt-4.7/index.htmls

[43] Nokia Corporation. *Qt Reference Documentation [online]*. [cit. 2020-05-10]. Available from: https://doc.qt.io/archives/qt-4.7/index.htmls

[44] Du, Z.; Yin, J.; et al. Expanding Receptive Field YOLO for Small Object Detection. *Journal of Physics: Conference Series*, volume 1314, oct 2019: p. 012202, doi:10.1088/1742-6596/1314/1/012202. Available from: https://doi.org/10.1088%2F1742-6596%2F1314%2F1%2F012202

[45] Zhu, P.; Wen, L.; et al. Vision Meets Drones: A Challenge [online]. 2018, [cit. 2020-05-11]. Available from: https://arxiv.org/pdf/1804.07437.pdf

[46] Mueller, M.; Smith, N.; et al. A benchmark and simulator for UAV tracking. *Proceedings of European Conference on Computer Vision [online]*, 2016, [cit. 2020-05-11]. Available from: https://ieeexplore.ieee.org/abstract/document/746635

[47] Workswell. *Workswell Wiris Pro Quick Start Guide [online]*. 2019, [cit. 2020-05-11]. Available from: https://workswell-thermal-camera.com/docs/UAV_WWP_QSG.pdf

[48] Tzutalin. Label Img [online]. 2015, [cit. 2020-05-11]. Available from: https://github.com/tzutalin/labelImg

[49] Nokia Corporation. *Qt Designer Manual [online]*. [cit. 2020-05-10]. Available from: https://doc.qt.io/qt-5/qtdesigner-manual.html

[50] Apple, Inc. *Preview User Guide[online]*. [cit. 2020-06-02]. Available from: https://support.apple.com/guide/preview/welcome/mac

[51] Python Software Foundation. *PEP 8 – Style Guide for Python Code [online]*. 2013, [cit. 2020-05-15]. Available from: https://www.python.org/dev/peps/pep-0008/

[52] Nokia Corporation. *QUiLoader Class[online]*. [cit. 2020-05-16]. Available from: https://doc.qt.io/archives/qt-4.7/index.htmls

[53] Lundh, F. *Pillow Overview [online]*. Secret Labs AB, [cit. 2020-05-12]. Available from: https://pillow.readthedocs.io/en/stable/handbook/overview.html

[54] Matoba. *PieExif Documentation [online]*. 2018, [cit. 2020-05-12]. Available from: https://piexif.readthedocs.io/en/latest/index.html

[55] Nokia Corporation. *Multithreading Technologies in Qt [online]*. [cit. 2020-05-24]. Available from: https://doc.qt.io/qt-5/threads-technologies.html

[56] Nokia Corporation. *Synchronizing threads [online]*. [cit. 2020-06-04]. Available from: https://doc.qt.io/qt-5/threads-synchronizing.html

[57] Python Software Foundation. *abc - Abstract Base Classes [online]*. [cit. 2020-05-13]. Available from: https://docs.python.org/3/library/abc.html

[58] Google Inc. *Google Maps Elevation API*. 2020, [cit. 2020-05-08]. Available from: https://developers.google.com/maps/documentation/elevation/start

[59] Mallick, S. Blob Detection Using OpenCV [online]. 2015, [cit. 2020-05-10]. Available from: https://www.learnopencv.com/blob-detection-using-opencv-python-c/

[60] Open Source Computer Vision. *SimpleBlobDetector Class Reference [online]*. [cit. 2020-06-03]. Available from: https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html

[61] Workswell, s.r.o. Field of View Calculator [online]. 2020, [cit. 2020-05-11]. Available from: https://workswell-thermal-camera.com/field-of-view-calculator/s

[62] John, C.; Brent, F. *The image processing handbook*. Boca Raton: CRC Press, Taylor & Francis Group, 7th edition, 2016, ISBN 978-1-4987-4026-5.

[63] Ponnusamy, A. YOLO Object Detection with OpenCV and Python [online]. 2018, [cit. 2020-05-15]. Available from: https://www.arunponnusamy.com/yolo-object-detection-opencv-python.html

[64] Murugavel, M. How to train YOLOv3 to detect custom objects. 2018, [cit. 2020-06-04]. Available from: https://medium.com/@manivannan_data/how-to-train-yolov3-to-detect-custom-objects-ccbcafeb13d2

[65] Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv [online]*, 2018. Available from: https://pjreddie.com/media/files/papers/YOLOv3.pdf

[66] Wu, Y.; Kirillov, A.; et al. Detectron2. *InfoWorld [online]*, 2019, [cit. 2020-05-29]. Available from: https://github.com/facebookresearch/detectron2[online]

[67] Yegulalp, S. Facebook brings GPU-powered machine learning to Python [online]. 2017, [cit. 2020-05-29]. Available from: https://www.infoworld.com/article/3159120/artificial-intelligence/facebook-brings-gpu-powered-machine-learning-to-python.html

[68] Facebook Inc. *Detectron2 Beginner's Tutorial [online]*. 2020, [cit. 2020-05-15]. Available from: https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5

Acronyms

AI Artificial Inteligence. 12, 43

API Application Programming Interface. 21, 25, 35, 38

CNN Convolutional Neural Network. ix, xi, 12, 13, 15, 16, 49

COCO Common Objects In Context. 18, 42

CPU Central Processing Unit. 46, 47

DJI Dà-Jiāng Innovations. 10, 27

EXIF Exchangeable Image File Format. 34, 38

Fast R-CNN Fast Region Based Convolutional Neural Network. 15, 16

Faster R-CNN Faster Region Based Convolutional Neural Network. x, xi, 15–17, 26, 43, 46, 47

FLIR Forward-looking Infrared. 4, 10, 19

GPS Global Positioning System. 38

GPU Graphics Processing Unit. 43, 46, 47, 49, 50

GUI Graphical User Interface. x, 24, 29, 33

HOG Histograms of Oriented Gradients. 12, 19

IDE Integrated Development Environment. 29

JPEG Joint Photographic Experts Group. 28, 34, 38

JSON JavaScript Object Notation. 35, 45

MOBIA Multicriteria Object-Based Image Analysis. 20

MVC Model-View-Controller. 24, 33

R-CNN Region Based Convolutional Neural Network. 15, 16, 20, 26

ReLU Rectified Linear Unit. 13

RGB Red Green Blue. 12, 18, 42, 43

RoI Region of Interest. 16

SIFT Scale-Invariant Feature Transform. 12

SVM Support Vector Machines. 12, 16, 19

TIFF Tagged Image File Format. 28, 34, 38

UAV Unmanned Autonomous Vehicle. ix, 10, 20, 27

YOLO You Only Look Once. x, xi, 15, 17, 18, 20, 26, 29, 42, 43, 47, 49, 50

YOLOv3 You Only Look Once version3. 42, 43, 46, 47

Appendix A

Contents of enclosed CD

- `app` ..... the directory with the application source codes
  - `source` ..... Python sources
  - `ui` ..... Graphical Interface sources
  - `detector_helper_files` ..... Weights and configurations used by the application for detection
  - `test-dataset` ..... Scipt used for detection evaluation
  - `readme.md` ... the file with the application description and installation guide
  - `thermaldetector.pyproject` the file with the Qt project configuration
  - `thermaldetector.py` ..... Main application class
- `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `BP_Glejtek_Matej_2020.pdf` ..... the thesis text in PDF format
- `dataset` ..... directory with thermograms for testing purposes