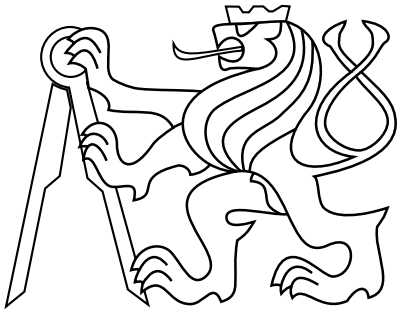
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Cybernetics
Multi-robot Systems



# Drone Detection and Relative Localization Using an RGB and Thermal Camera

**Bachelor's Thesis**

**Aimira Baitieva**

Prague, May 2022

Study programme: Open Informatics
Branch of study: Artificial Intelligence and Computer Science

**Supervisor: Ing. Matouš Vrba**

## Prohlášení autorky práce

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ..... .....
podpis autorky práce

## Author statement for undergraduate thesis

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

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

# BACHELOR‘S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Baitieva Aimira** Personal ID number: **492313**

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

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

## II. Bachelor’s thesis details

Bachelor’s thesis title in English:

**Drone Detection and Relative Localization Using an RGB and Thermal Camera**

Bachelor’s thesis title in Czech:

**Detekce dronů a jejich relativní lokalizace pomocí RGB a termální kamery**

Guidelines:

Develop an algorithm for detection and relative localization of drones using convolutional neural networks and using a combined RGB+thermal camera placed onboard a flying UAV. Evaluate the precision, detection range, robustness and general performance of the resulting algorithm and perform an ablation study. Implement the approach to run under ROS with the MRS UAV system and test it in a real-world experiment. The motivation to this assignment is autonomous drone hunting which requires a reliable detector capable of working under diverse conditions
(see http://mrs.felk.cvut.cz/projects/eagle-one).

Bibliography / sources:

[1] F. Svanström, C. Englund and F. Alonso-Fernandez, "Real-Time Drone Detection and Tracking With Visible, Thermal and Acoustic Sensors," ICPR 2020, pp. 7265-7272.
[2] B. Khalid, A. M. Khan, M. U. Akram and S. Batool, "Person Detection by Fusion of Visible and Thermal Images Using Convolutional Neural Network," C-CODE, 2019, pp. 143-148.
[3] M. Vrba and M. Saska, "Marker-Less Micro Aerial Vehicle Detection and Localization Using Convolutional Neural Networks," RA-L 5(2), 2020, pp. 2459-2466.
[4] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: towards real-time object detection with region proposal networks," TPAMI 39(6), 2020, pp. 1137-1149.

Name and workplace of bachelor’s thesis supervisor:

**Ing. Matouš Vrba Department of Cybernetics FEE**

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

Date of bachelor’s thesis assignment: **06.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

| Ing. Matouš Vrba | prof. Ing. Tomáš Svoboda, Ph.D. | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

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

. _____
Date of assignment receipt

_____
Student’s signature

## Acknowledgments

I would like to express my deepest gratitude to my supervisor Ing. Matouš Vrba for his support and valuable advice. I am also thankful to all members of the Multi-Robot Systems group for building the system I used in this thesis and for their priceless help with experiments. And of course, many thanks to my friends and family for keeping my motivation high during the whole study process.

## Abstract

This thesis covers the design, implementation, and testing of a program for the detection and localization of unmanned flying vehicles (UAVs) with the use of RGB and thermal photos and fusion algorithms. Detection and relative localization is based on the Faster R-CNN network with FPN as a feature map generator and ResNeXt or ResNet as a feature extractor. A comparison of detecting and localizing UAVs with the use of RGB photos only, thermal photos only, and different fusion architectures is provided, and the advantages of the different approaches are discussed.



**Keywords** unmanned aerial vehicles, thermal camera, convolutional neural network, relative localization, computer vision



## Abstrakt

Tato práce se zabývá návrhem, implementací a testováním programu pro detekci a lokalizaci bezpilotních létajících prostředků (UAV) s využitím RGB a termálních fotografií a fúzních algoritmů. Detekce a relativní lokalizace je založena na Faster R-CNN síti s FPN pro generování příznakových map a ResNeXt nebo ResNet pro extrakci příznaků. Na datech z praktických experimentů jsou porovnány detekce a lokalizace UAV s použitím pouze RGB fotografií, pouze termálních fotografií a různých fúzních architektur a výhody různých přístupů jsou rozebírány.



**Klíčová slova** bezpilotní prostředky, termální kamera, konvoluční neuronová síť, relativní lokalizace, počítačové vidění

## Abbreviations

**RTK** Real-time Kinematic positioning
**LiDAR** Light Detection and Ranging
**MRS** Multi-robot Systems Group
**ROS** Robot Operating System
**UAV** Unmanned Aerial Vehicle
**RGB** Red, Green, Blue
**CNN** Convolutional Neural Network
**R-CNN** Region Based Convolutional Neural Network
**FPN** Feature Pyramid Network
**ResNet** Residual Neural Network
**YOLO** You Only Look Once
**SSD** Single Shot Detector
**MAV** Micro Aerial Vehicle
**KAIST** Korea Advanced Institute of Science and Technology
**TPR** True Positive Rate
**FPR** False Positive Rate
**ReLU** Rectified Linear Unit
**SGD** Stochastic Gradient Descent
**AP** Average Precision
**AR** Average Recall
**IoU** Intersection over Union
**FLIR** Forward-looking Infrared
**GUI** Graphical User Interface
**CV** Computer Vision
**API** Application Programming Interface
**MAE** Mean Absolute Error
**RoI** Region of Interest
**FP** False Positive
**FN** False Negative
**TP** True Positive
**FPS** Frames Per Second

# Contents

# Chapter 1

# Introduction

This thesis is focused on the detection and relative localization of drones. A drone is an unmanned aircraft guided remotely by an operator or autonomously by some program. Unmanned aerial vehicle (UAV) means the same, these terms are interchangeable, and both will be used in this thesis.

Although there exist many constructions of UAVs, this thesis mainly considers quadcopters, a type of helicopter with four rotors, as in Figure 1.1. They have a relatively simple control system, can fly vertically, and are widely used due to these reasons.

**Figure 1.1:** A flying quadcopter DJI F450 used for the experiments in this thesis[1].

Detection and localization of drones are important due to several reasons. It may be used for a practical realization of cooperative exploration tasks, collision avoidance [7], or even for the interception of non-cooperative drones [1], which can be dangerous for critical infrastructure objects, like power plants or airports.

This problem can be tackled using different methods. The most obvious is with visual information from a camera, processing it by a convolutional neural network (CNN). This method leads to stable detections in good weather and lighting, but in bad weather conditions or without sufficient illumination, the task becomes more problematic.

For this reason, many researchers use additional sources of information. For example, some works employ acoustic sensors [14] or thermal cameras [2]. In [5], the authors process inputs from the thermal camera with a CNN in the same way as inputs from a standard RGB camera to obtain reliable information in more difficult lighting conditions (e.g., during the night, in a fog, etc.). Another approach is to combine the thermal and RGB images as in [9].

[1] `http://mrs.felk.cvut.cz/research/micro-aerial-vehicles`

A crucial element of this approach to drone detection is choosing a suitable network architecture. The current state-of-the-art object detectors are Faster R-CNN [22], YOLO [20], SSD [19], etc. This project uses Faster R-CNN due to its outperforming quality of detection [15], especially on small objects [3]. The feature-pyramid network (FPN) was used for feature extraction as part of the Faster R-CNN because it improves the detection of objects at different scales while maintaining good computational efficiency [16].

Using known parameters of the camera and dimensions of the target, the 3D position of the detected objects may be calculated from its bounding box predicted by the CNN. A 3D position estimation method that takes advantage of this approach is proposed in this thesis.

## 1.1 Related works

There are no popular benchmarks for multispectral drone detection, so in this work, the results of detection using thermal photos will be compared with the results of detection with RGB input only.

Vrba et al. [4] proved that drones can be successfully detected with an RGB camera without any special markers placed on the drone. They used YOLO for detection to minimize processing delay and make it run online onboard another UAV. Relative localization was done with the calculations using the real size of the drone (which is either known or guessed by a neural network).

Combining information from both thermal and RGB photos in order to acquire more data may potentially improve the robustness and precision of the detection. The fusion may be tackled in many ways, from putting thermal photos as a fourth input channel of the CNN in addition to the R, G, and B channels, to more sophisticated structures, including concatenating features or generating proposals and detections by the two sub-networks separately and merging detection scores at the end (cascade design).

| Parameters | Accuracy | Miss Rate | TPR | FPR | Specificity |
|---|---|---|---|---|---|
| Visible | 97 | 3.4 | 97.05 | 3.22 | 96.6 |
| Thermal | 97.6 | 2.8 | 97.65 | 2.67 | 96.8 |
| Feature Fusion | 99.2 | 0.7 | 99.25 | 0.59 | 98.76 |
| Proposed | 99.8 | 0.01 | 99.85 | 0.02 | 99.86 |

**Figure 1.2:** Comparison of different fusion techniques for thermal and RGB detection [8]. TPR is the true positive rate, and FPR is the false positive rate.

Khalid et al. [8] experimented with fusion using handcrafted features and fusion using CNN on the KAIST dataset[2]. They showed that processing visible and thermal images separately and fusing features at the end in ResNet-152 allows achieving better results than using only one type of input, as seen in Figure 1.2.

Li et al. [9] performed an in-depth comparison of different ways of merging thermal and RGB photos in their work on the detection of pedestrians on the KAIST dataset. They used Faster R-CNN as a base network and derived six different fusion architectures from it, which are shown in Figure 1.3.

[2] https://soonminhwang.github.io/rgbt-ped-detection/

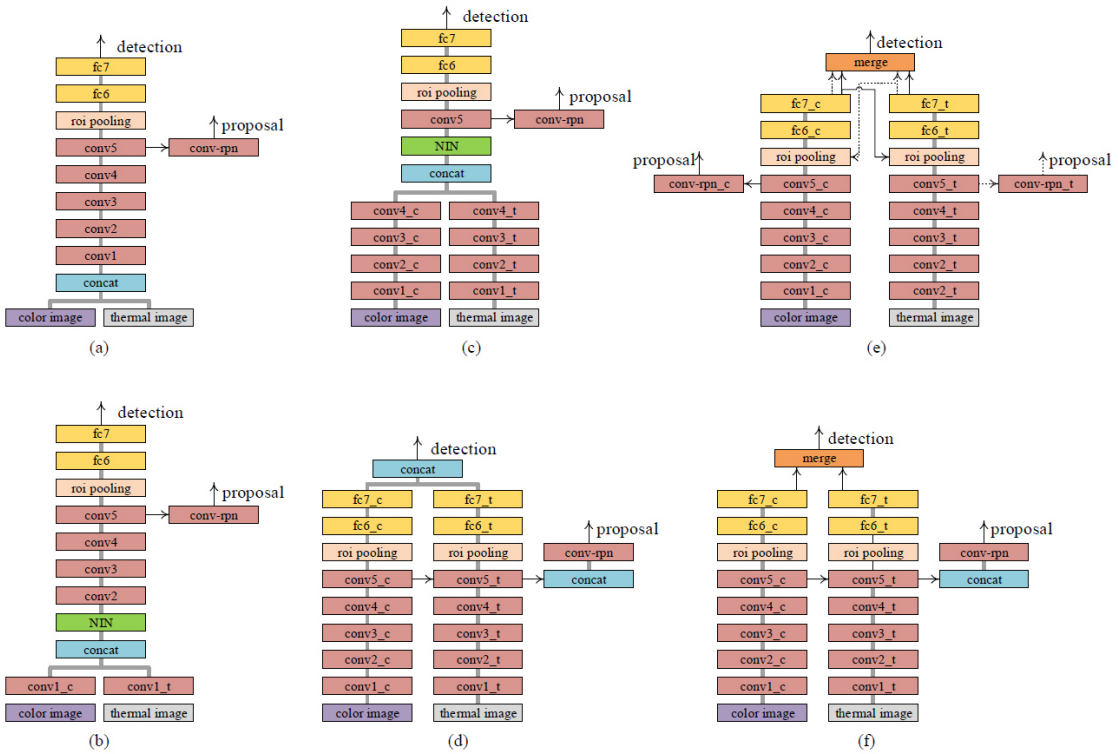



**Figure 1.3:** Different architectures for thermal and RGB fusion, (a) Input Fusion, (b) Early Fusion, (c) Halfway Fusion, (d) Late Fusion, (e) Score Fusion I, (f) Score Fusion II [9].

Halfway Fusion and Score Fusion I outperformed the other architectures. The authors attribute it to benefiting from the balance between semantic information and low-level features in the case of Halfway Fusion and cascade design in the case of Score Fusion I.

This thesis also explores the possibility of onboard localization, which adds stricter requirements to the complexity of calculations because of SWaP (Size, Weight, and Power) constraints of today's micro-scale UAVs. In that case, using many input sources can slow down the speed of localization up to the point of being unusable in applications where a low delay and high update rate are crucial, such as multi-robot cooperation, collision avoidance, or autonomous aerial interception. This also is true for artificially increasing the resolution of the image, in [10] authors showed that it can improve recall up to 32.4%, but such a promising technique can hardly be used for onboard application.

Svanström et al. [2] claim that thermal cameras are successful in UAV detection even without any additional sensors. Drones often have metal chassis that is heated either by the sun or by batteries (which can heat up to 60°C by themselves) and usually have noticeably higher temperatures than their surroundings (as apparent in Figure 1.4).

In [11] a high-resolution RGB input is used for detection of a small UAV-like objects. Later, the thermal image was used to distinguish drones from airplanes or birds, which helped to achieve larger than 80% accuracy.

Another possibility for localizing drones is to use a depth map (which can be obtained from a LiDAR or stereo camera). Adrian Carrio et al. [12] trained their detector to use depth data only and achieved an average precision of 98.7% on a distance up to 9.5 meters.

**Figure 1.4:** Thermal image of a drone [2].

## 1.2 Contributions

The goal of this thesis is to establish that thermal photos can be used as a reliable source of information about drones and their position, especially in bad weather and lighting conditions, and to compare it with the quality of detection using RGB photos only and different types of fusion between thermal and RGB photos. The most promising results (both from the quality and computational efficiency sides) will be tested in real-life experiments with real drones.

# Chapter 2

# Neural Network

In this project, slightly different network architectures are used for different tasks. The best quality is achieved by using a network based on the Faster R-CNN with FPN as a backbone, which uses ResNeXt-101 for bottom-up feature extraction. The faster network for real-time experiments uses a smaller feature extractor, the ResNet-50.

The network takes thermal or RGB images or both and outputs predictions of bounding boxes of drones in the input together with their respective confidence scores.

## 2.1 Architecture

A neural network consists of layers of individual learning units - neurons, see Figure 2.1. Each of these is a function that processes input and returns output. This function works by multiplying inputs by their weights, summing them, and applying the activation function. Weights are parameters that are changed while learning. For convolutional layers, the output of the neuron can be interpreted as a measure of how the specific feature represented by the neuron is present in the input (for example, a vertical line in the image).

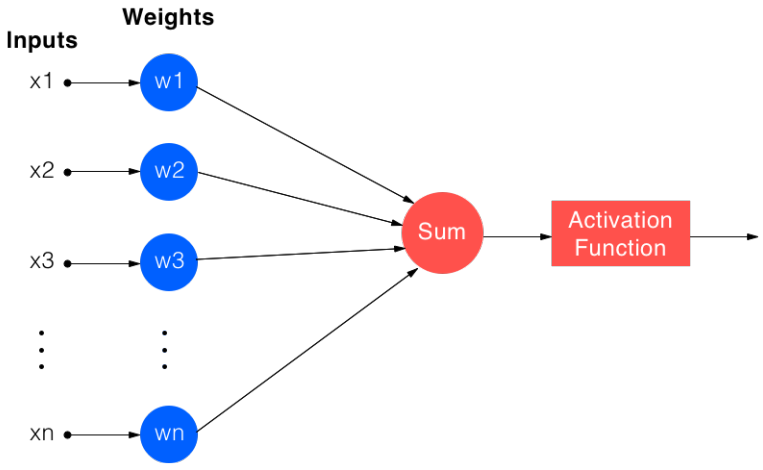



**Figure 2.1:** A model of a single neuron in a neural network[1].

The activation function defines the output. The simplest activation function is the binary step function, which returns one if the feature is detected (input passes some threshold) or zero if not. In practice, non-linear functions are used for this because it is important to see

[1] https://hackernoon.com/a-hands-on-introduction-to-neural-networks-6a03afb468b1

not only if a feature was detected but how strong it is, so results are usually a number in the range from zero to one. Non-linearity helps to solve non-trivial problems with various data. Typical non-linear activation functions are Sigmoid, Tanh, the most popular is ReLU (Rectified Linear Unit) or its modifications. Sigmoid and ReLU are visualized in Figure 2.2.

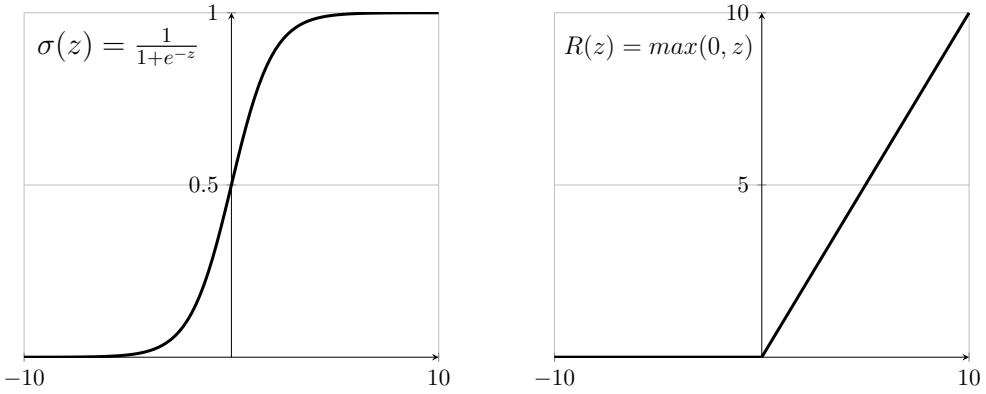



**Figure 2.2:** Example activation functions typically employed in neural networks, Sigmoid (left) and ReLU (right).

A stack of neurons is called a layer. Each neuron in a layer takes input from all neurons in the previous layer and processes it as described before. Due to this, the neural network can recognize the smallest features first (parts of lines or edges in the image) and use them to recognize more and more complicated concepts (circles or angles, and later even separate details like eyes or wheels in the image), which is shown in Figure 2.3.

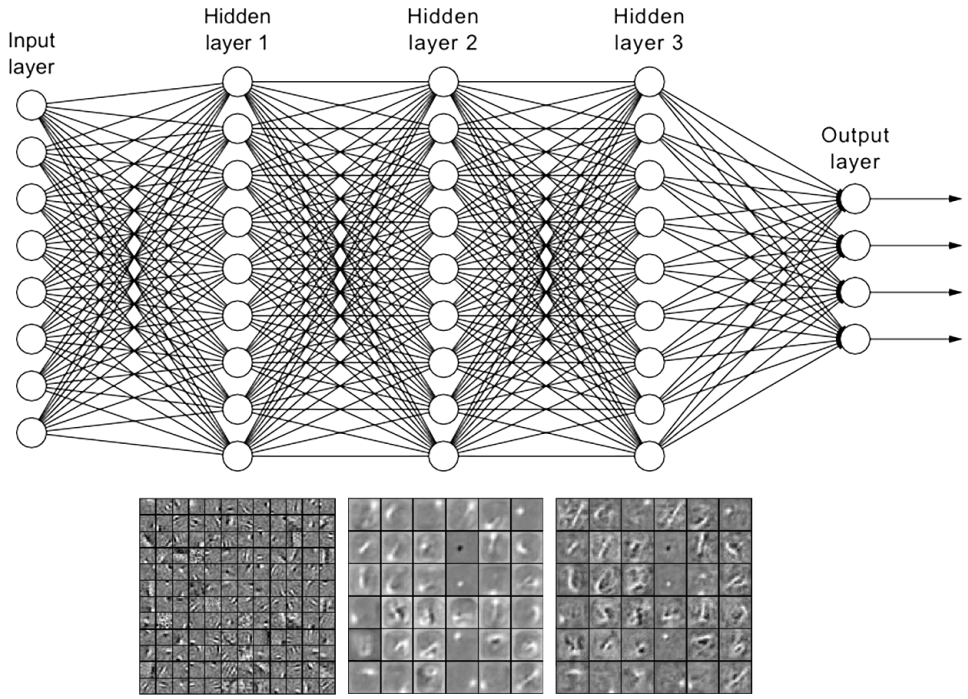



**Figure 2.3:** Illustration of a fully connected neural network and features recognized by each layer [26].

### 2.1.1 Convolutional Neural Network (CNN)

CNN is the most widely used way to classify images since the success of Alexnet [25]. As shown in Figure 2.4, a typical CNN architecture is a linear stack of different layers.

The core building block is a convolutional layer. Its kernels slide over the input data, computing the dot product between itself and the input, returning a feature map for each kernel. Feature maps capture features, preserving semantic meaning, which depends on pixel position and its neighborhood.

Another important part of a CNN is the pooling layer. It reduces the spatial size of feature maps. The most common pooling method is max pooling, which returns the highest value from an area of the feature map.

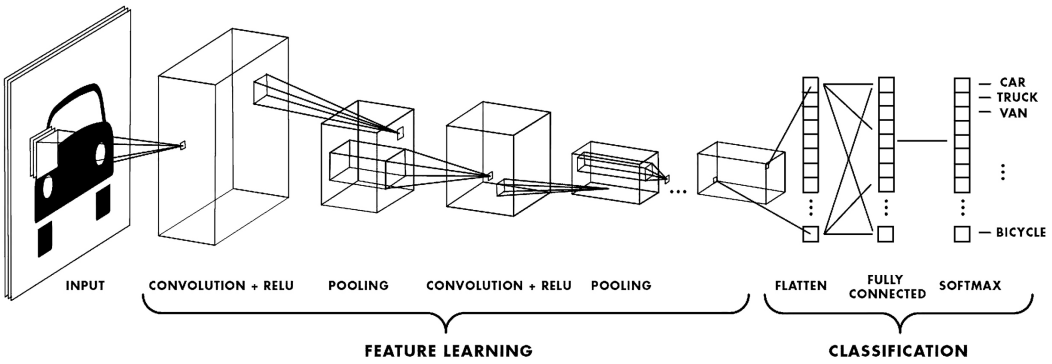



**Figure 2.4:** Illustration of a typical structure of a Convolutional Neural Network[2].

A fully connected layer takes features extracted before and uses them to classify the initial image. Its input is flattened to make it a vector instead of a 3-dimensional matrix.

Softmax is a type of multi-input activation function which converts its input to a vector of probabilities of the output classes (makes them sum up to 1). For each $x_i, x_j$ in the input, it can be calculated using the following formula:

$$\sigma(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}. \tag{2.1}$$

### 2.1.2 ResNet

ResNet-50 (residual neural network with 50 layers) is a CNN backbone that is optimized for computation-efficient feature extraction. In this work, it was used as the backbone for a faster variant of the detection CNN. The ResNet-50 was pre-trained on the ImageNet dataset. ResNet was created to address the accuracy saturation problem and the vanishing gradient in deeper networks.

The accuracy saturation problem causes deeper networks to have a higher training error rate than smaller networks. In theory, the error rate should be decreasing or at least not changing with an increase in the network's depth.

The vanishing gradient problem may be encountered when using gradient backpropagation to train a neural network. During the backward propagation, the gradient can become

[2] https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306

vanishingly small, which prevents the network from updating its weights and stopping the training process.

Instead of simply stacking layers, ResNet-50 skips some of the layers in between, which is shown in Figure 2.5. This method is named shortcut connections, and it successfully helps mitigate the problems described above.

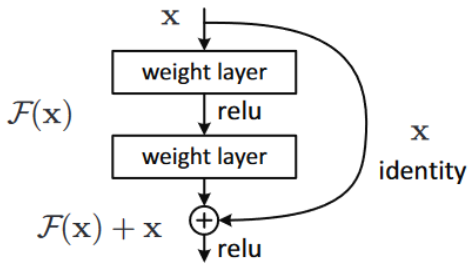



**Figure 2.5:** Illustration of the skip connection, which is a building block of the ResNet-50 CNN architecture [18].

### 2.1.3 ResNeXt

ResNeXt-101 [17] is a state-of-the-art network for image classification. In this work, it was used as the feature-extraction backbone for a larger variant of the detection CNN. It was pre-trained on the ImageNet dataset. It is a state-of-the-art network for image classification. In our case, it is used as a feature detector. It follows the VGG strategy of repeated layers [24] and uses shortcuts between blocks as ResNet while having comparable complexity.

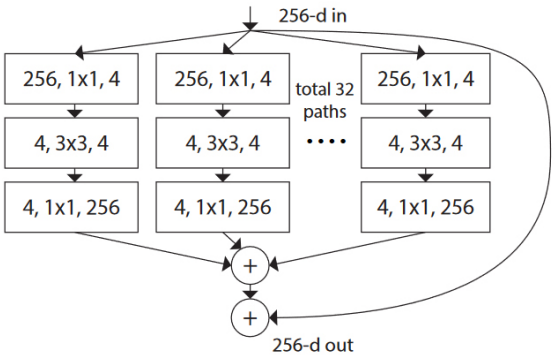



**Figure 2.6:** A single block of ResNeXt, C = 32 [17].

The main difference from ResNet is an additional dimension called Cardinality (the parameter C in Figure 2.6). A ResNeXt block uses a strategy similar to the Inception neural network [13]: splitting the input into several branches with lower-dimension data, applying convolutions, and merging them back through concatenation. Cardinality shows how many uniform branches ResNeXt has in each block, which can be seen in Figure 2.6. Increasing cardinality improves the network's performance better than increasing its width or depth.

### 2.1.4 Feature Pyramid Network (FPN)

FPN [16] is a modern and efficient solution to multi-scale detection. It uses an old concept of processing differently scaled images at once, which is time and memory-consuming compared to processing the image only once. However, instead of working with multi-scale images, it works with multi-scale feature maps, which helps it be reasonably fast while having better quality detections than pyramids of images. It is compared with the similar architectures in Figure 2.7.

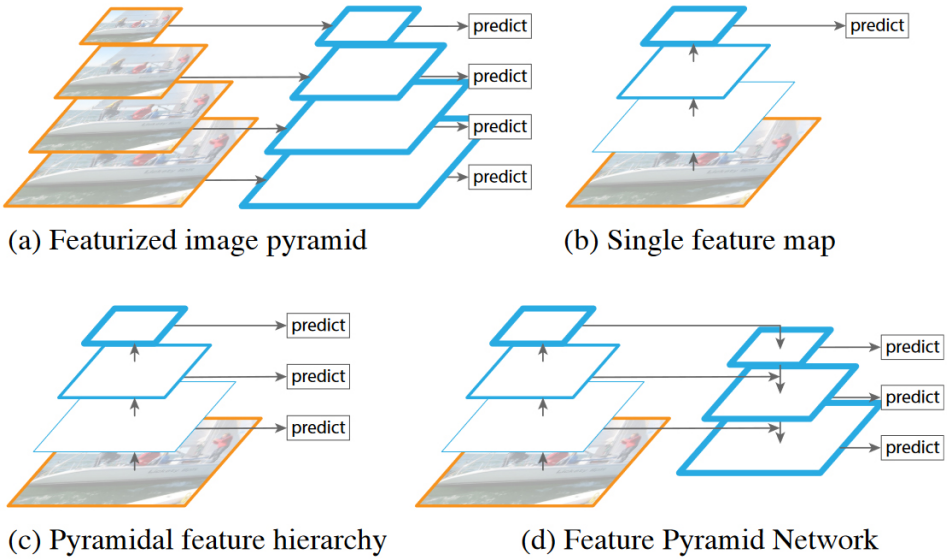



**Figure 2.7:** Comparison of FPN and similar architectures [16].

FPN has a bottom-up and top-down pathway. The bottom-up pathway is used to extract features, decreasing spatial resolution with each layer and increasing the semantic value. The top-down pathway utilizes both high resolution of the bottom layers and rich features from the upper layers, creating a high-resolution feature map that allows detecting even small objects.

### 2.1.5 Faster R-CNN

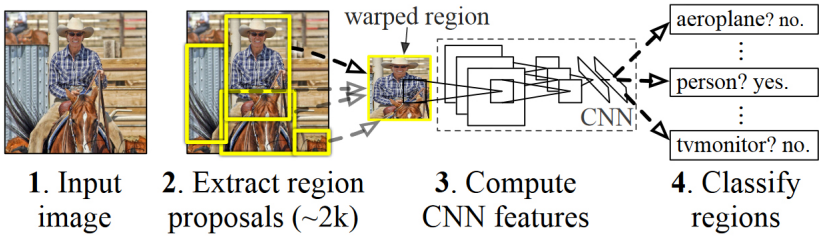



**Figure 2.8:** R-CNN: Regions with CNN features [23].

The need to detect not only the presence of the object but also its boundaries within the image (bounding box) differentiates the object detection task from the image classification task. Furthermore, images can contain multiple objects of the same class, and all of them need

to be detected. To tackle this task, neural networks for object detection may select multiple areas of interest, which are later processed by a CNN to decide if there is an object inside. Such a naive approach would produce a massive amount of these areas of different sizes and shapes, and processing all of them would take an unreasonable amount of time.

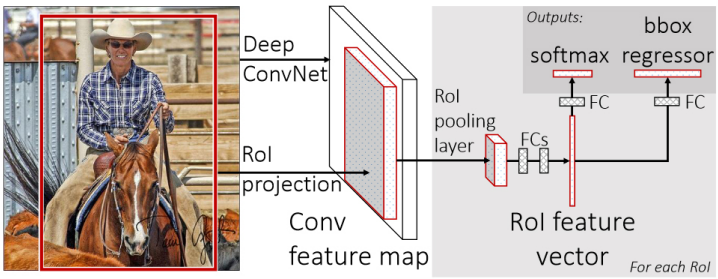



**Figure 2.9:** Fast R-CNN [21].

To solve this problem, different algorithms were created, including the Region-CNN (R-CNN) [23]. The main feature of this network is generating around 2000 category-independent region proposals (much less than the naive version), using a greedy algorithm to merge similar regions. Figure 2.8 shows the main principles of its work.

The next step to improve the detection algorithms was Fast R-CNN [21]. It generates a feature map from the whole image and uses it to process all proposals instead of generating it separately for each proposal as R-CNN does, which is shown in Figure 2.9. The training time decreased 10 times, and inference time decreased more than 20 times compared to R-CNN.

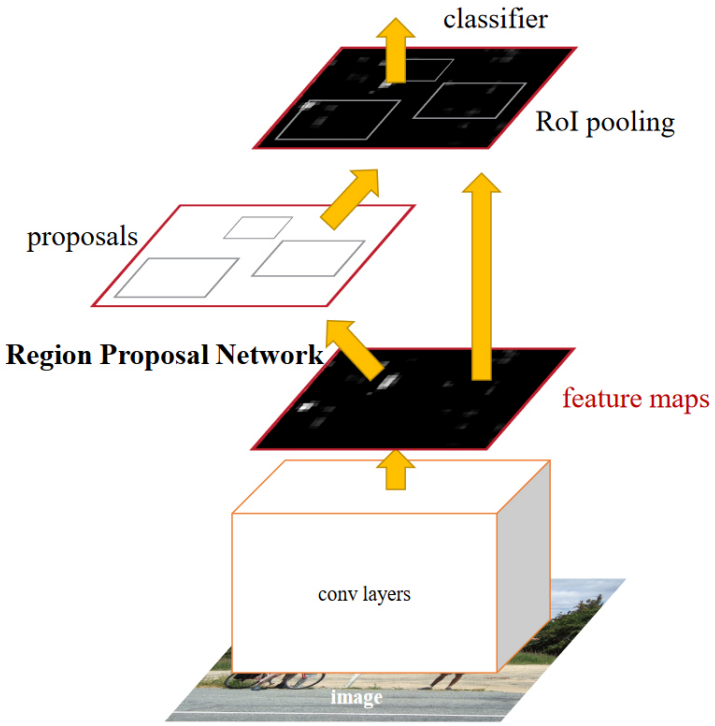



**Figure 2.10:** Faster R-CNN [22].

Faster R-CNN is a current state-of-the-art object detector. It succeeds Fast R-CNN, being faster (as subtly mentioned in its name). The main difference is the use of a specialized Region Proposal Network (RPN) instead of the slow selective search. The RPN is a small, fully convolutional subnetwork that applies a 3x3 sliding window over the feature maps to predict if there is an object in that box, which ensures an efficient way to compute proposals. It uses a pre-computed feature map (in our case, pre-computed by FPN using ResNeXt or ResNet) as the input. It returns proposals with their objectness score (probability of the object being there). Only boxes with high objectness scores will be processed further. RPN can be trained separately, but training it together with the whole Faster R-CNN showed up to be more effective.

After RPN, proposals are processed by region of interest (RoI) pooling. It takes feature maps and proposals as input, scales down proposals to the feature map level, performs max pooling, and returns standardized boxes of features to process them by fully-connected layers to classify them. Figure 2.10 summarizes all the descriptions above.

### 2.1.6 CNN architecture used in this thesis

This thesis utilizes several modifications of the standard Faster R-CNN. It uses FPN for extracting feature maps on different scales, which in its turn uses ResNeXt-101 or ResNet-50 for extracting features itself.

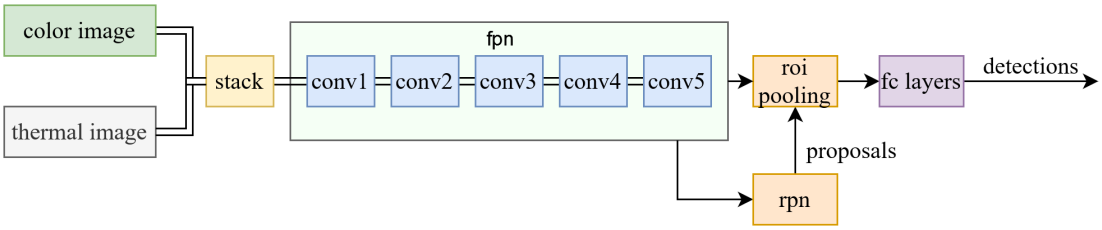



**Figure 2.11:** Input fusion in Faster R-CNN.

Firstly, several different architectures were tested to compare their effectivity and robustness. These were implemented using the Detectron2[3] library, which provides many features while being relatively simple. Two fusion architectures were created in addition to the existing configurations.

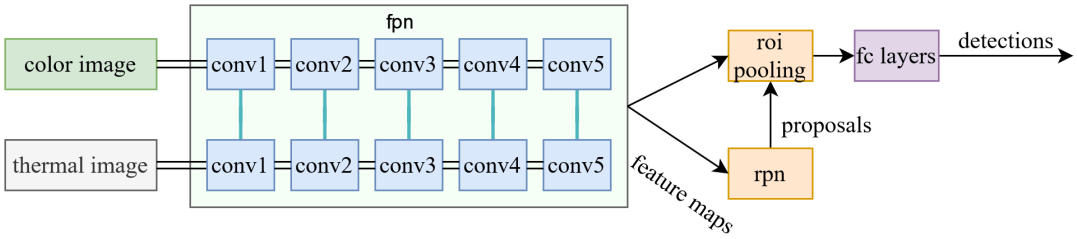



**Figure 2.12:** Halfway fusion in Faster R-CNN.

Input fusion stacks RGB and thermal images into one four-channel image that is passed into the network, as illustrated in Figure 2.11. For pre-trained weights of the newly added

[3]https://github.com/facebookresearch/detectron2

channel containing the thermal image, the mean value of the pre-trained weights of the RGB channels was used.

Fusing images inside the FPN is slightly more complicated. Inputs of the FPN, ResNeXt (or ResNet) networks has to be modified. On each of the five levels of the FPN, the mean value of the extracted features is taken and feature maps with information from both images is returned, which can be seen in Figure 2.12.

During the preliminary tests, the direct input fusion (Figure 2.11) demonstrated worse performance than using only RGB images as the input, so this architecture was abandoned in favor of the other variants.

For the main test were chosen the RGB only, the thermal only, and the halfway fusion models. The first two are implemented using the MMDetection toolbox [6], which allows conversion to TensorRT[4] (faster inference with NVIDIA graphic cards). The halfway fusion is implemented with the same Detectron2 library as before due to a more complicated structure of the MMDetection toolbox.

## 2.2 Training

The neural network's output depends on its input, the architecture of the neural network (number of layers, activation function, size of the convolutional kernels, etc.), and the weights of the respective neurons. These weights are a parameter of the network that is typically not manually selected but rather trained using a supervised learning approach. The training is an optimization problem that iteratively minimizes the specified loss function.

### 2.2.1 Loss function

The error is calculated by the loss function (also sometimes called the cost function). A widely used loss function for image classification is the cross-entropy loss function, which was also used for pretrained weights in this thesis[5]. The formula below shows how to calculate it for the multi-class task, $M$ is a number of classes, $y_c$ is one the predicted class should be $c$ and zero otherwise, and $p_c$ is the predicted probability of class $c$:

$$L = \sum_{c=1}^{M} y_c \log(p_c). \tag{2.2}$$

The L1 loss function, which computes the mean absolute error (MAE), was used for the detection training. It is a loss function commonly used when training regression neural networks, which means it is used for the calculation of a continuous quantity instead of a discrete class label as with the classification functions. It can be calculated using the formula:

$$L = \frac{1}{N} \sum_{i=1}^{N} |x_i - y_i|, \tag{2.3}$$

where $x_i$ is the prediction and $y_i$ is the ground truth, $N$ is the sample size.

The learning process is minimization of the loss function, which is typically done with iterative algorithms, such as gradient descent.

[4] https://developer.nvidia.com/tensorrt

[5] https://github.com/pytorch/vision/blob/main/references/classification/

### 2.2.2 Gradient descent

Gradient descent is a commonly used optimization algorithm for training neural networks. In each iteration, it makes steps in the direction of a local minimum of the loss function, which is the opposite of a function's gradient:

$$a_{n+1} = a_n - \gamma \nabla f(a_n), \tag{2.4}$$

where $\gamma$ is learning rate, $\nabla f$ is gradient of the loss function, $a_n$ is a step of gradient descent.

The learning rate corresponds to the speed with which the network is trained. A larger learning rate corresponds to faster training, although it also makes missing the minimum more possible. A lower learning rate is considered to be safer, but it can be too slow or stuck in a local minimum, see Figure 2.13. There are different techniques to avoid these problems. The simplest example is learning rate decay, which decreases the learning rate after every n-th iteration, allowing achieving the minimum faster but not overshoot.

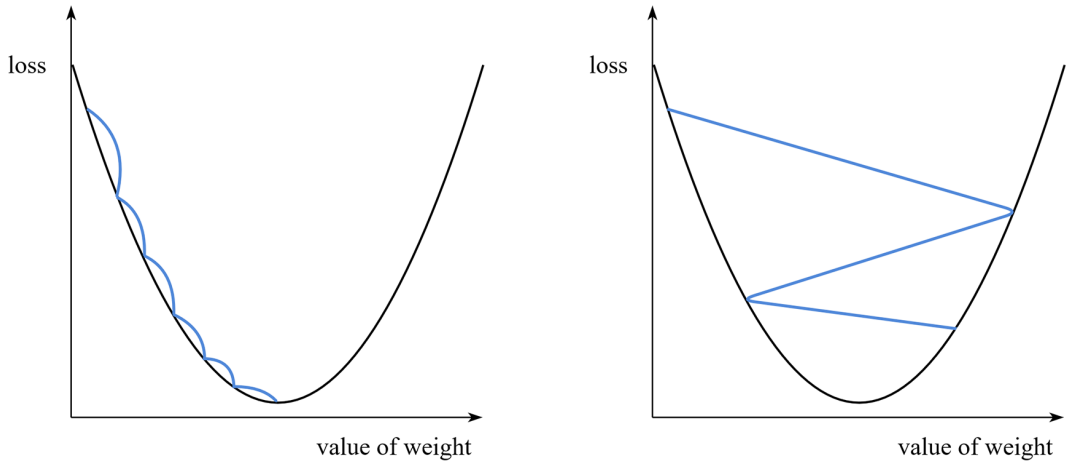



**Figure 2.13:** Illustration of gradient descent with low (left) and high (right) learning rates.

Backpropagation is a differentiation algorithm for computing gradient. It calculates the gradient of the loss function in the weights space using a chain rule[6].

Neural networks in this thesis were trained using the stochastic gradient descent (SGD), a typical choice for training Faster R-CNN models [22]. It is similar to gradient descent, but instead of calculating the gradient from the whole set of training data, the gradient is calculated from a random subset, significantly reducing the number of computations of one step at the cost of slower convergence.

Momentum, an often employed modification of the SGD algorithm, adds some "mass" to the gradient, which makes it converge faster and "roll" over small local minima. Its effect on training is illustrated in Figure 2.14. A typical momentum value of 0.9 was used in training in this thesis.

Weight decay is another modification of the base SGD algorithm that makes neural networks "forget" weights during the training, decaying them in the direction towards zero.The weight decay coefficient was set to the relatively low value of 0.0001 since weights for the feature detector were already pre-trained.

[6] https://en.wikipedia.org/wiki/chain_rule

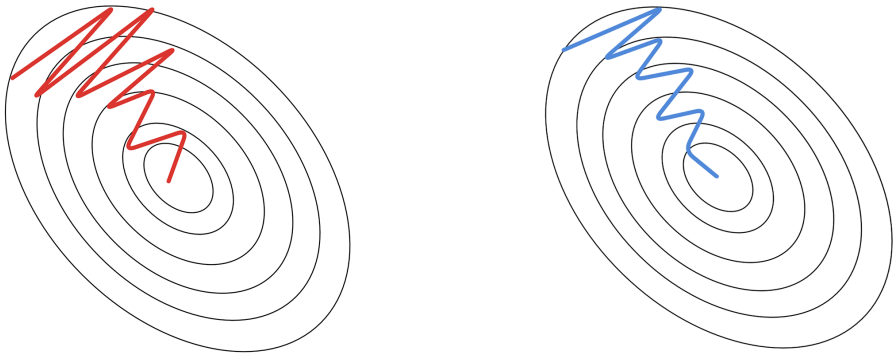


**Figure 2.14:** SGD without momentum (left) and with it (right).

### 2.2.3 Evaluation metrics

To compare performance of different neural networks on the testing dataset, it is necessary to define some evaluation metrics. The typical metrics for object detection are average precision (AP) and average recall (AR).

To calculate these metrics, the confidence score (probability that the bounding box contains the object, estimated by the network) and the intersection over union (IoU) are used. IoU is calculated using the formula:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}, \tag{2.5}$$

where $B_p$ is the predicted bounding box and $B_{gt}$ is the ground-truth box.

Knowing these parameters, the number of true positive results (TP) may be obtained. A detection is counted as a true positive if its confidence score is higher than a specified threshold, the predicted class is correct (not applicable in this case as there was only one class), and the predicted bounding box has an IoU bigger than a specified threshold.

If a detection has the wrong class or its IoU is lower than the threshold, it is counted as a false positive (FP), which means that the network detected something it should not have (for example, a part of the background was detected as a drone).

A false negative (FN) is a situation when the confidence score of a detection is lower than the threshold (for example, there was a drone, but the prediction was made with low confidence, so it is not counted).

Using these concepts, precision $P$ and recall $R$ are calculated as:

$$P = \frac{TP}{TP + FP}, \tag{2.6}$$

$$R = \frac{TP}{TP + FN}. \tag{2.7}$$

The average precision (AP) is an interpolated area under the precision-recall curve (drawing recall on the x-axis and precision on the y-axis). It shows the precision averaged through all recall levels. Similarly, AR is the area under the recall-IoU curve.

Models in this thesis were evaluated using the AP and AR over the IoU interval [0.5, 0.95] with a step of 0.05.

### 2.2.4 Datasets and epochs

Before starting the training, the whole dataset is divided randomly into three parts - training, validation, and testing. The training part is used in training for updating the weights by some gradient descent algorithm. The validation dataset is used to find out at which point to stop the training to prevent overfitting (reaching a state when the model exactly fits the training data, which hampers generalization of the neural network and its performance on other than the training data). The testing dataset is used to compare different models after finishing training.

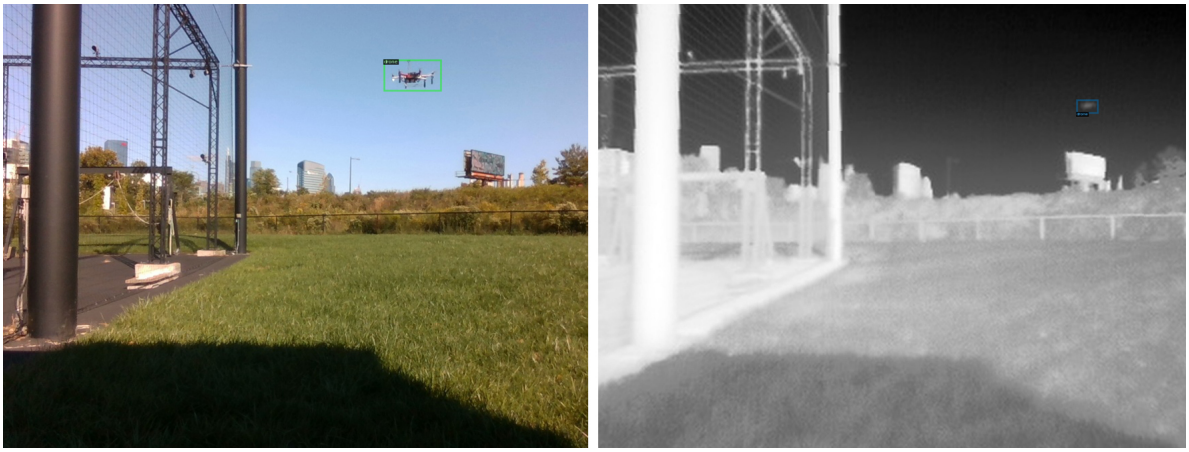


**Figure 2.15:** Examples of input photos from the preliminary experiments with the ground truth bounding boxes. An RGB photo is on the left, and a thermal photo is on the right.

Neural networks are usually trained by epochs. One epoch means one iteration through the entire training dataset. The total number of epochs depends on the size of the dataset and the variety of the data in it. In many cases, it is helpful to use pretrained weights, which are typically pretrained on some large dataset for a long time. Using pre-trained weights helps drastically reduce the training time because the network is then capable of detecting common basic features already. In that case, the number of epochs needed for the training process to converge on the desired specific domain is smaller, as well as the learning rate.

For preliminary experiments and comparisons of the considered CNN architectures, a dataset of 1584 color and thermal photos was used, showing a single drone flying in different positions, in the air and on a grassy surface. An example of an RGB and thermal images from this dataset with the ground truth bounding boxes can be seen in Figure 2.15. Some photos show the drone in front of a video billboard, which proved especially challenging to detect for both the human eye and neural network. All photos were filmed on a sunny day. The resolution of the thermal images is $320\,\text{px} \times 256\,\text{px}$.

For the main experiment, a larger and more challenging dataset was obtained (3754 photos). Examples of the images are shown in Figure 2.16. A single drone was flying far away in the direction of the forest. The drone is easily visible in some RGB photos due to its bright colors. In others, it blends with the trees perfectly. The target's smallest ground truth bounding box was 14 px wide with a $640\,\text{px} \times 512\,\text{px}$ thermal image resolution. Photos were filmed closer to the evening on a cloudy day.

The photos were extracted and synchronized from the thermal and color video streams. Ground truth bounding boxes were then manually labeled with a helper script that was written

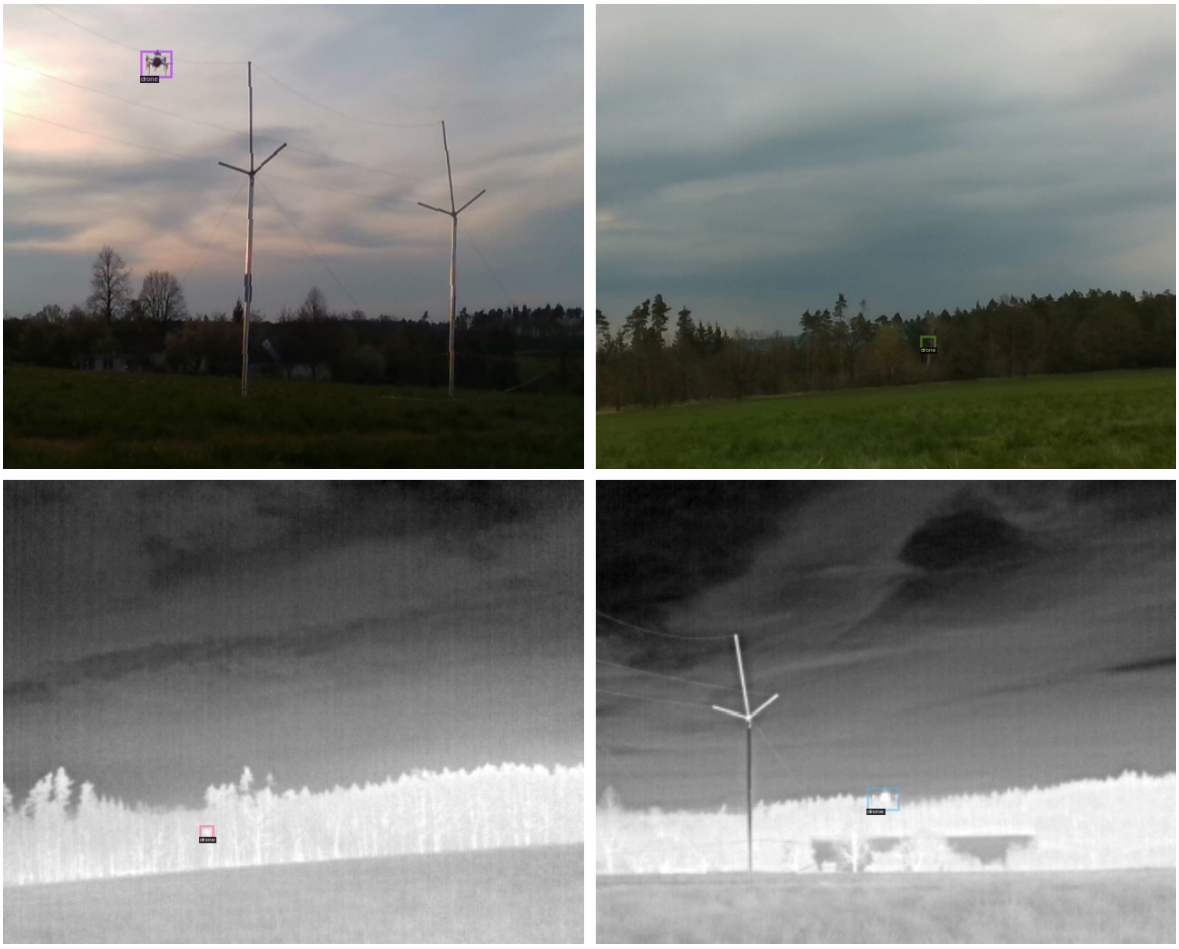


**Figure 2.16:** Examples of input photos from the main experiment with the ground truth bounding boxes, top pictures are from the RGB camera, and bottom are from the thermal camera.

for this purpose.

The photos were split randomly into training, validation, and testing datasets using the proportion 70% + 15% + 15%. The default training scripts from the MMDetection and Detectron2 libraries were used, with the evaluation on a separate validation dataset during training. Random vertical flip with the probability of 0.5 was used as an augmentation[7].

Weight values pre-trained on the Imagenet dataset[8] were used to initialize the weights before training.

[7] https://en.wikipedia.org/wiki/Data_augmentation
[8] https://www.image-net.org/

# Chapter 3

# Relative localization

Relative localization of detected objects can be obtained with some trivial calculations. Firstly, the input image has to be undistorted using radian and tangential distortion models with the coefficients obtained by camera calibration. After this, pixels of the image can be projected into 3D rays using the pinhole camera model[1]. Then, known physical dimensions of the detected object can be used to estimate its 3D position relative to the camera from its predicted bounding box in the image.

## 3.1 Calibration

To obtain the relative location of a detected object in a camera image, a camera projection model and its parameters must be known. This thesis uses data from an Intel RealSense depth camera D435i[2] and from a FLIR Hadron 320 and a FLIR Hadron 640[3]. The Realsense provides images with distortion already compensated, but this is not the case for the Hadron. Its camera has a wider lens, and without calibration, outputs from both cameras cannot be properly aligned.

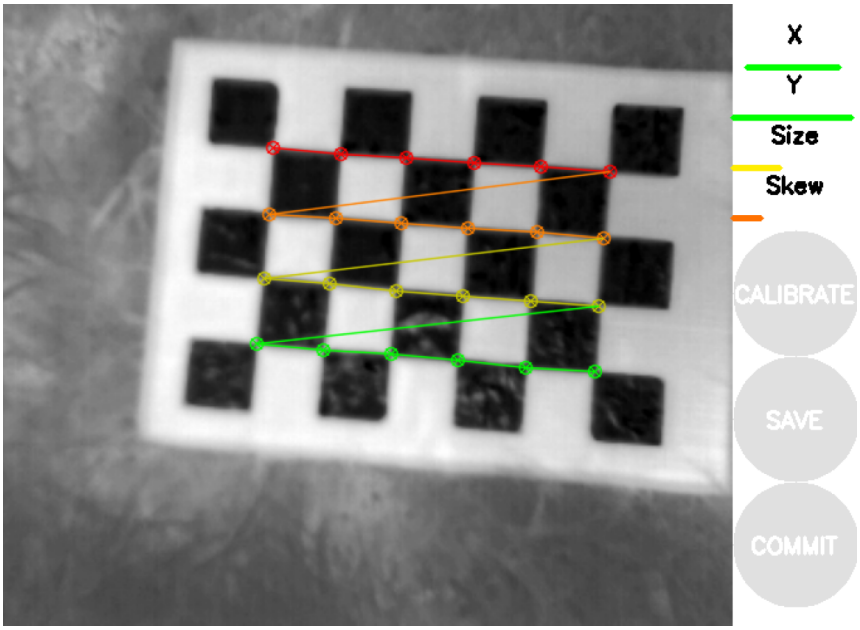



**Figure 3.1:** Camera calibration GUI.

[1] https://en.wikipedia.org/wiki/Pinhole_camera_model
[2] https://www.intelrealsense.com/depth-camera-d435i/
[3] https://www.flir.com/products/hadron/

There are many solutions for automatic calibration of cameras. For this thesis, the ROS Camera calibration package[4] was used. It utilizes the OpenCV[5] library and a chessboard-like pattern. Its graphical user interface (GUI) is shown in Figure 3.1. A special "thermal" chessboard that was created for a different project was utilized for this task. It has squares made from an aluminum foil to reflect the cold sky and to have a black color in the thermal image. The chessboard background is black, which in its turn shows in white color on the thermal image.

Images from the camera are distorted mainly by radial distortion and tangential distortion. Radial distortion makes straight lines in the image look curved. This distortion is represented in the OpenCV library using the equations:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{3.1}$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{3.2}$$

where $x_{distorted}$ and $y_{distorted}$ are pixel coordinates in the raw image, $k_1, k_2, k_3$ are parameters of the model.

Tangential distortion adds a rotation and skew, making some parts of the image look closer or farther away. The OpenCV library models this distortion using the equations:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)], \tag{3.3}$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 x], \tag{3.4}$$

where $x_{distorted}$ and $y_{distorted}$ are pixel coordinates in the raw image, $p_1, p_2$ are parameters of the model.

Aside from the five coefficients $k_1, k_2, k_3, p_1$, and $p_2$, it is necessary to calculate the camera matrix, which contains focal length $f_x, f_y$ and optical centers $c_x, c_y$. The camera matrix is then used for projecting the undistorted pixel coordinates to rays in the 3D space.

## 3.2 Relative localization

After the calibration, pixels from the original image can be mapped to an undistorted image using these formulas and the coefficients calculated before:

$$\begin{aligned}
x &= (u - c_x)/f_x \\
y &= (v - c_y)/f_y \\
r^2 &= x^2 + y^2 \\
x' &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2) \\
y' &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy \\
u' &= x' f_x + c_x \\
v' &= y' f_y + c_y,
\end{aligned} \tag{3.5}$$

where $u, v$ are coordinats of a pixel in an undistorted image, $u', v'$ are coordinats of a pixel in the original image.

[4] https://wiki.ros.org/camera_calibration
[5] https://docs.opencv.org/4.x/index.html

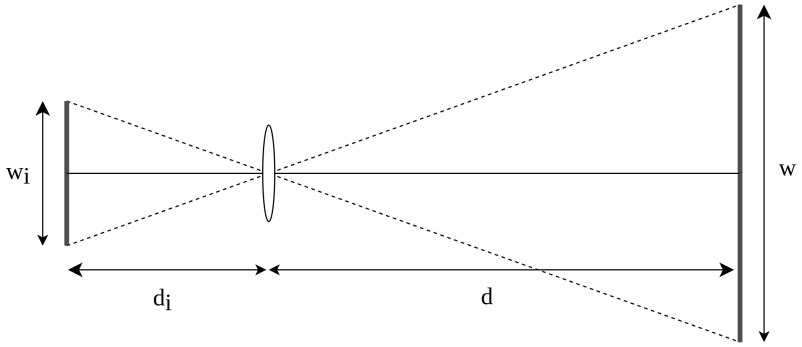



**Figure 3.2:** Real-world object and object in the image, distance to the object is $d$, distance to the object in the image is $d_i$, size of the object is $w$, size of the object in the image is $w_i$.

For the relative localization from a monocular camera image, it is necessary to know also the physical dimensions of the detected object. Figure 3.2 shows how a real-world object is projected to the camera image using the pinhole camera model. Based on this geometrical projection model, the distance of the object from the camera can be obtained using the formula:

$$d = \frac{wd_i}{w_i}. \tag{3.6}$$

# Chapter 4

# Implementation

## 4.1 ROS, the MRS UAV system

Robot Operating System (ROS)[1] is an open-source framework for robots [27]. It provides libraries, drivers, visualizers, tests, and many other necessary things to create applications to control drones, robotic arms, automated guided vehicles, etc.

Processes in this framework are distributed and can run separately. As an example, a camera has its own process, which reads images and can send them to other processes, including an image rectifier that receives raw images and sends rectified images as an output. In the context of ROS, these processes are called nodes.

Nodes communicate with each other using topics. A node can post messages on some topic, and all other nodes who subscribed to this topic will receive them.

ROS supports different programming languages, including Python, which was used for the implementation of the algorithms described in this thesis due to the reason that libraries used for the object detection task have a Python API.

The software, written as part of this thesis, is designed to run with the Multi-robot Systems (MRS) Group UAV system[2], which is built using ROS. This system allows to control multi-rotor vehicles with PX4-compatible flight controller[3].

## 4.2 Implementation

The program was implemented to run as a node, which receives images from one or two cameras and posts output into topics “detections”, “poses” and “visualization”. There are two options on which feature extractor to use in the Faster RCNN: ResNet-50 or ResNeXt-101, first one is roughly two times faster, but the second provides more accurate and robust detections (see chapter 5). Also, the program can switch between RGB and thermal cameras if no drone was detected for some number of received images, as shown in algorithm 1.

In the preliminary experiment, all models were implemented using the Detectron2 library, speed was not tested.

For the main experiment, models from the MMDetection toolbox were used. MMDetection provides a possibility to translate models to the TensorRT framework, which increases the speed of inference on NVIDIA hardware. TensorRT models were tested as well as the original PyTorch[4] implementations. Results of the speed test can be seen in Table 4.1. The networks

[1] https://wiki.ros.org/
[2] https://github.com/ctu-mrs/mrs_uav_system
[3] https://github.com/ctu-mrs/px4_firmware
[4] https://pytorch.org/

**Algorithm 1** Main algorithm

**Require:** $Model, Subscibers, Publishers, NoDetectionsCounter, Threshold$

**while** True **do**
  **if** got *Image* from *Subscriber* **then**
    $Detections \leftarrow$ INFERENCE($Model, Image$) ▷ Use object detection model on image
    **if** no *Detections* **then**
      $NoDetectionsCounter \leftarrow +1$
    **else**
      $NoDetectionsCounter \leftarrow 0$
      $Poses \leftarrow$ GETPOSES($Detections$) ▷ Calculate poses from the bounding boxes
      $Visualization \leftarrow$ VISUALIZE($Detections, Image$) ▷ Paint bounding boxes
      publish outputs
    **if** $NoDetectionsCounter$ is bigger than $Threshold$ **then**
      $Model \leftarrow$ CHANGEMODEL()
  **else**
    wait

run on the Nvidia Jetson Xavier NX[5] platform that is designed for embedded neural network inference to simulate the usage of the networks onboard a UAV. Fusion with ResNeXt-101 was not transformed to the TensorRT engine because it is implemented using Detectron2, which does not support the translation of Faster R-CNN to a TensorRT, so its inference time in TensorRT was not measured.

| | PyTorch | TensorRT |
|---|---|---|
| RGB with ResNet-50 | 2.65 | 3.75 |
| RGB with ResNeXt-101 | 1.37 | 1.91 |
| Thermal with ResNet-50 | 2.74 | 3.76 |
| Thermal with ResNeXt-101 | 1.38 | 1.95 |
| Fusion with ResNeXt-101 | 0.71 | - |

**Table 4.1:** FPS (frames per second) of the different CNN architectures from the main experiment.

[5] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/

# Chapter 5

# Experiments

## 5.1 Thermal camera

The main aim of this thesis was to evaluate the usage of a thermal camera for drone detection, which is a topic that is still not widely covered in the published scientific literature. All objects emit infrared radiation. This radiation may be measured by a thermal camera to create visual images. Such images represent the surface temperature and emissivity of the observed objects. The higher the temperature and emissivity, the more heat the object emits. Since thermal cameras do not need any visible light, it makes them a good choice for object detection in challenging lighting conditions. Examples of thermal photos are shown in Figure 5.1.

**Figure 5.1:** Examples of input from a thermal camera.

The main difference between the camera working in the visual light spectrum is the use of a microbolometer. It is a sensor, which detects infrared radiation using a temperature resistive sensing element. Changes in electrical resistance are mapped into temperatures which are translated into an image. Microbolometer also does not require cooling, unlike older thermal sensors.

Another difference is the lens material. Glass blocks long-wave infrared light, for this reason lenses are usually made of special materials, such as calcium fluoride or germanium. Due to this, thermal cameras have a low resolution, compared to RGB cameras (the thermal camera used in the main experiment has a resolution of 640 px × 512 px).

According to the information provided by the manufacturer[1], FLIR Hadron 640 used in

[1] https://www.flir.com/discover/rd-science/how-do-thermal-cameras-work/

this thesis can detect differences in heat signature (radiation emitted by the object) as small as 0.01°C.

**Figure 5.2:** The RGB and thermal cameras mounted together.

This camera was mounted together with an Intel RealSense D435i, as can be seen in Figure 5.2. To achieve a proper alignment of the two types of images, the different resize, rotation, and crop operations were applied to an RGB image. The thermal image was undistorted using the distortion model and parameters of this model were found using calibration as described in section 3.1.

## 5.2 The preliminary experiment

Target of the preliminary experiment was to compare different fusion methods to only using RGB or thermal images alone. Therefore, it was performed using only one feature extractor, the ResNeXt-101.Among the compared models (subsection 2.1.6), the half-way fusion architecture demonstrated the best results, which may be explained by the strong features from the RGB input amplified by the same strong features from the thermal input. All results can be seen in Table 5.1.

| CNN architecture | AP | AR |
|---|---|---|
| RGB | 0.78 | 0.82 |
| Thermal | 0.757 | 0.79 |
| Input fusion | 0.769 | 0.81 |
| Half-way fusion | 0.81 | 0.85 |

**Table 5.1:** Preliminary experiment results.

## 5.3 The main experiment

### 5.3.1 Day dataset

The different neural network architectures were tested on a separate dataset than was used for the training. Average precision (AP), average recall (AR), and precision of the relative localization were evaluated. AP and AR metrics are shown in Table 5.2. It can be seen that although the CNN architectures that use ResNeXt-101 as a feature extractor show better performance according to the AP and AR metrics, they are significantly slower (Table 4.1), which might be critical for the onboard applications. The inference speed may be expected to improve using a stronger computer such as the Nvidia Jetson Xavier AGX[2], potentially enabling to use of the presented system in dynamic multi-robot scenarios.

| CNN architecture | AP | AR |
|---|---|---|
| RGB with ResNet-50 | 0.744 | 0.796 |
| RGB with ResNeXt-101 | 0.773 | 0.831 |
| Thermal with ResNet-50 | 0.663 | 0.723 |
| Thermal with ResNeXt-101 | 0.705 | 0.767 |
| Half-way fusion with ResNeXt-101 | 0.74 | 0.806 |

**Table 5.2:** Evaluation results of detections in the main experiment, day dataset.

For the fusion method to work well, a good alignment of the images and good time synchronization are necessary. However, these requirements were not perfectly met in this experiment, as can be seen in Figure 5.3 (ground truth boxes were painted using RGB images, on some photos they were not aligned with the thermal images). This highlights the potential problems when deploying similar methods in practice. Furthermore, using the mean value of features from both inputs might be a too simple strategy for this more complicated dataset. In the RGB images, the drone was not always visible which affected the fusion result.

**Figure 5.3:** Examples of wrong ground truth boxes due to misaligned photos.

Otherwise, all detection models showed good results on the testing dataset. As can be seen in Figure 5.4, even the most faraway positions were detected clearly with high precision even by the simpler detector with ResNet-50 as a feature extractor.

[2] https://developer.nvidia.com/embedded/jetson-benchmarks

**Figure 5.4:** The RGB (left) and the thermal (right) detectors in the day dataset.

### 5.3.2 Evening dataset

Models were also tested on another dataset that was filmed later in the evening. This dataset proved more challenging for the detectors, as illustrated in Figure 5.5. Sometimes the drone was not detected at all because the target visually blended with the background trees in both the visual and thermal spectra, being too distant from the cameras.

**Figure 5.5:** Examples of false positive results by the RGB (left) and the thermal (right) detectors in the evening dataset.

Each 10th photo (248 in total) from this dataset were used for testing, results can be seen in Table 5.3.

Larger networks showing worse results than smaller ones can indicate that the input was too different from the training dataset. As an example, the training dataset did not include a close view of a drone, which caused false positive results when the part of the drone was detected as another drone (can be seen on the thermal image in Figure 5.5). The imprecise bounding boxes could be explained by the drone flying up to 80 meters away, although in the training dataset, the maximal distance was 60 meters.

| CNN architecture | AP | AR |
|---|---|---|
| RGB with ResNet-50 | 0.112 | 0.176 |
| RGB with ResNeXt-101 | 0.109 | 0.184 |
| Thermal with ResNet-50 | 0.201 | 0.280 |
| Thermal with ResNeXt-101 | 0.162 | 0.266 |
| Half-way fusion with ResNeXt-101 | 0.2 | 0.256 |

**Table 5.3:** The quality of detections in the main experiment, evening dataset.

### 5.3.3 Relative localization

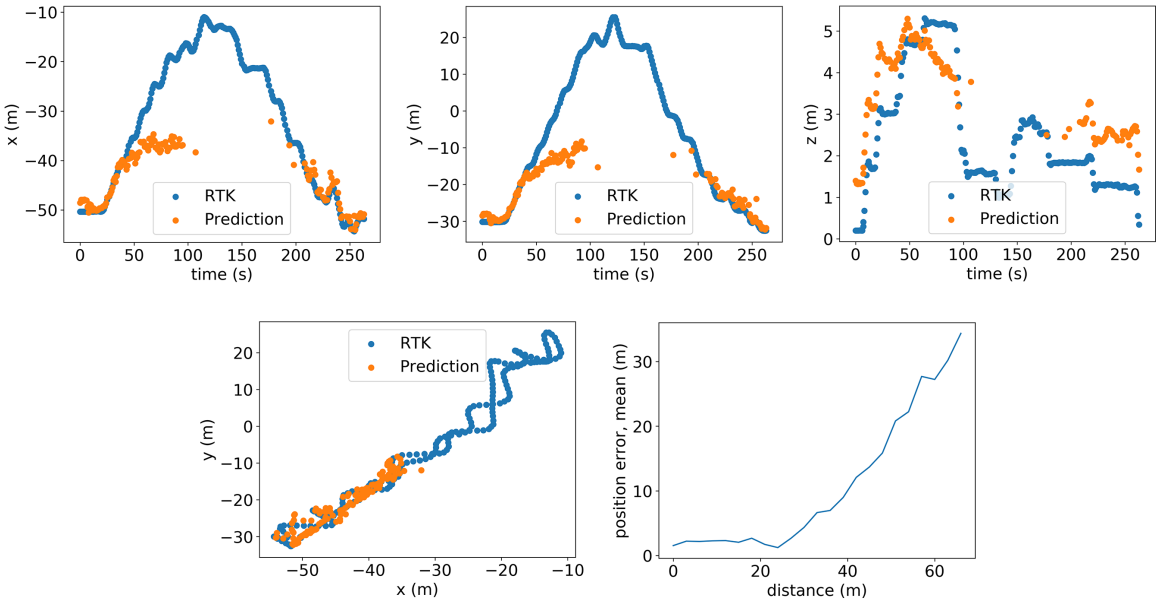



**Figure 5.6:** First row, from left to right: X, Y, Z coordinates in meters changing by time. Second row, from left to right: top view of the ground truth and estimated trajectory of the UAV, error of the estimated position over distance. These are the results on the evening dataset of the CNN model using only RGB images as the input.

The quality of localization was tested using RTK (Real-time kinematic positioning)[3] data as a ground truth. It was done using the evening dataset. Results of the model with the ResNet-50 feature extractor and an RGB input are shown in Figure 5.6, results for the thermal model are in Figure 5.7. The position error is calculated as the average of the Euclidean distance between the predicted position of the target and the ground truth over every 3 meters.

Due to imperfect ground truth bounding boxes in the training dataset, if the drone was more than 23 meters from the camera, the detected bounding boxes were slightly larger than the real drone, which led to a wrong distance prediction. This problem was addressed with an empirical correction. If $e$, the estimared distance, is larger than 23 meters:

$$e = e + (e - 23)^{1.2}. \tag{5.1}$$

[3] https://en.wikipedia.org/wiki/Real-time_kinematic_positioning

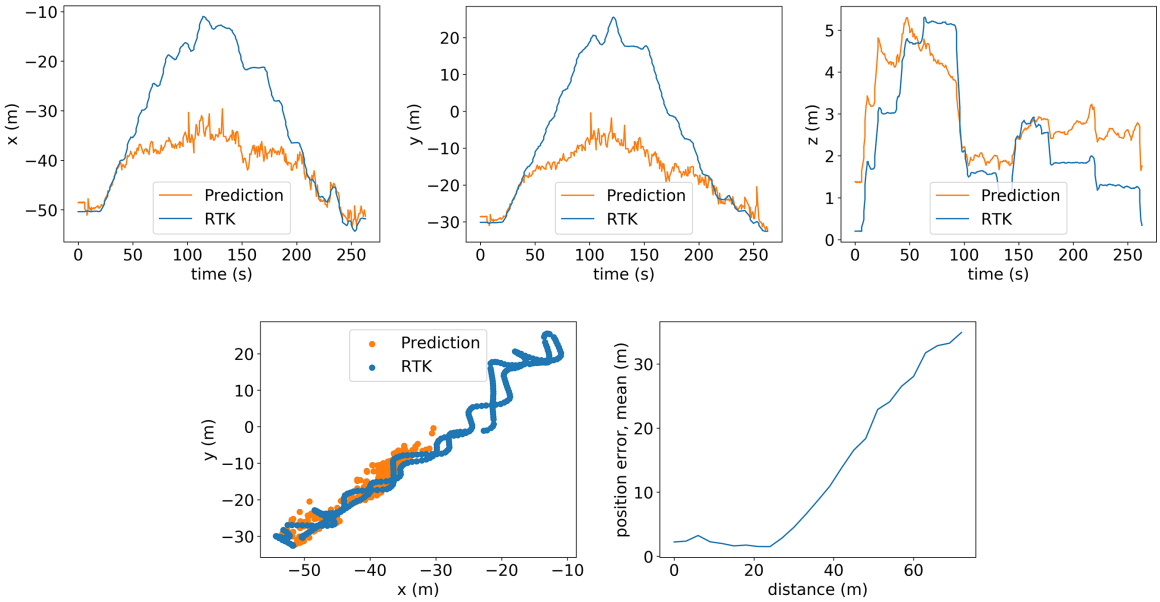



**Figure 5.7:** Same as in Figure 5.6, for the CNN model using only thermal images.

It amplified longer distances and successfully mitigated that problem on distances up to 45 meters, as shown in Figure 5.8 and Figure 5.9.

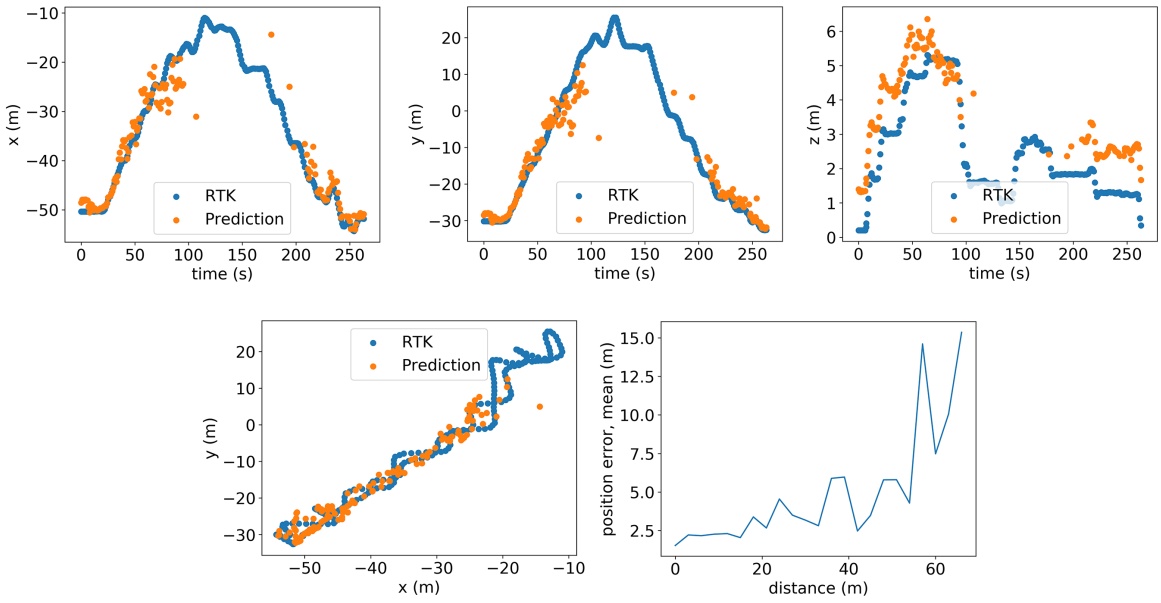



**Figure 5.8:** First row, from left to right: X, Y, Z coordinates in meters changing by time. Second row, from left to right: top view of the ground truth and estimated trajectory of the UAV, error of the estimated position over distance. These are the results on the evening dataset of the CNN model using only RGB images as the input with empirical correction.

Although the evaluation metrics on the evening dataset are not too impressive, the thermal network still was able to provide accurate localization with the mean error under 4 meters on a distance up to 40 meters. With a better training dataset (larger variety of photos

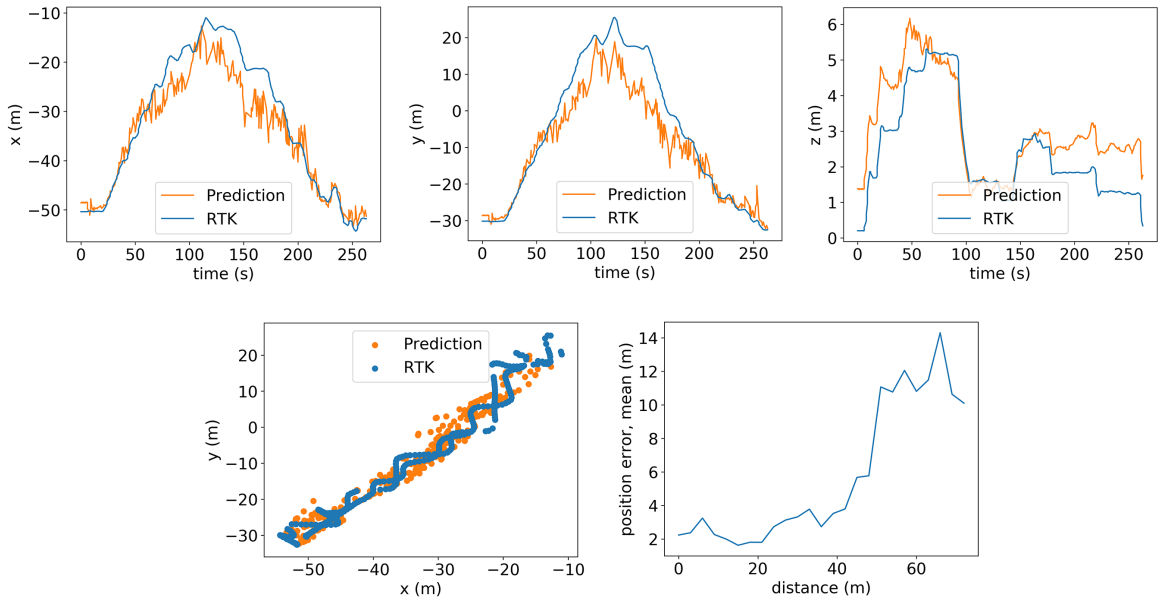



**Figure 5.9:** Same as in Figure 5.8, for the CNN model using only thermal images.

and proper ground truth bounding boxes on longer distances), these results can be potentially improved even further.

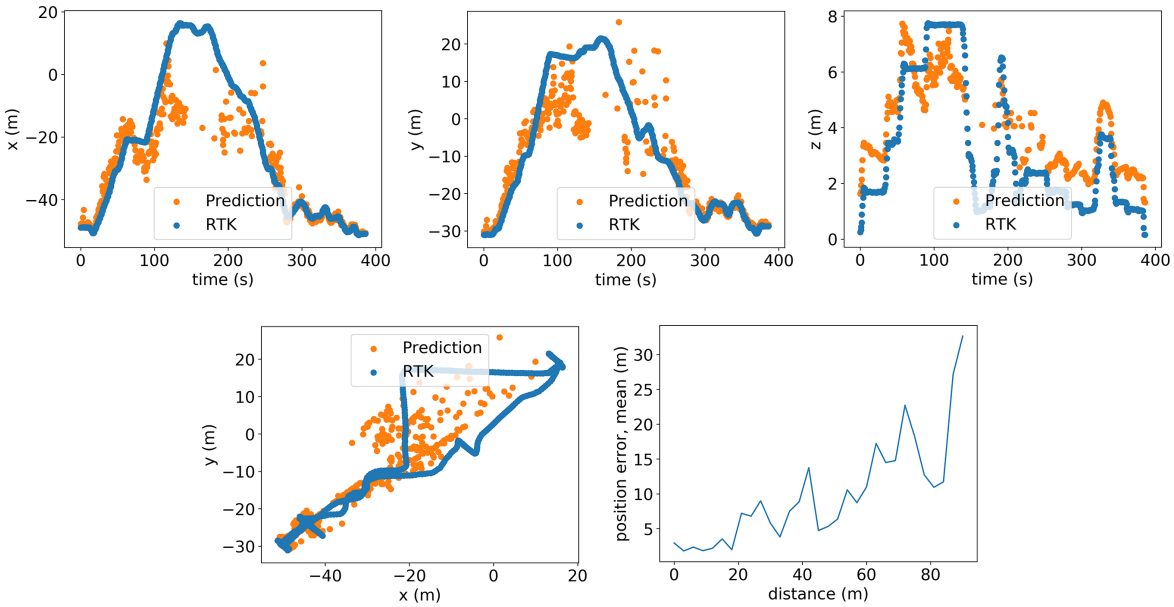



**Figure 5.10:** First row, from left to right: X, Y, Z coordinates in meters changing by time. Second row, from left to right: top view of the ground truth and estimated trajectory of the UAV, error of the estimated position over distance. These are the results on the night dataset of the CNN model using only thermal images as the input with empirical correction.

Finally, the thermal model was tested on a night dataset. Figure 5.11 shows that the RGB picture on the left does not have any noticeable objects except the lights of the drone, which is not a mandatory part of its construction, so the RGB data is (unsurprisingly) unusable

for detection of drones during the night. On the other hand, the thermal image on the right is not too different from the evening dataset.

**Figure 5.11:** An RGB (left) and thermal (right), both from the night dataset.

Results of the localization tests for the night dataset are showed in Figure 5.10. There are more false positives than in the previous datasets, but the presented system was still able to localize the drone with a mean error lower than 6 meters on a distance up to 20 meters.

# Chapter 6

# Conclusion

The collected data has established that thermal images are a reliable source of information about drones and their position in any lighting conditions, even at night. Furthermore, because thermal input utilizes only one channel, potentially it can run three times faster than an RGB network with three channels, making it a good choice for a Faster R-CNN and onboard computer. On the other hand, it has to be trained on a wider variety of thermal photos, because different drone components have variable temperatures and can be invisible due to blending with different backgrounds. Thermal cameras are also more expensive than their RGB counterparts.

An RGB input, used as a benchmark, performed well in good lighting conditions, but much worse with insufficient light in the evening and at night.

A fusion of both networks seemed promising after the preliminary experiment with the photos filmed in the middle of the day. Unfortunately, the employed method is not sophisticated enough to handle situations in which one input has noticeably less information than the other. The fusion method can be modified to use lighting-dependent coefficients for both inputs, as in [9]. For onboard operations, it might not be practical to use two cameras and calculate features for two images if one thermal camera provides only slightly worse results much faster. Another challenge that was identified during the implementation is achieving proper alignment of the images and good time synchronization, given the inherent complications of using two input streams.

Future research might focus on implementing a one-channel TensorRT model for thermal input, which could benefit from a superior speed. A more reliable and thorough dataset could increase accuracy and recall against backgrounds with competing thermal signatures.

# Chapter 7

# References


[1] Y. Fan, H. Lou, and S. Yu, "Review of the development status of uav countermeasures," in *Seventh Asia Pacific Conference on Optics Manufacture and 2021 International Forum of Young Scientists on Advanced Optical Manufacturing (APCOM and YSAOM 2021)*, SPIE, vol. 12166, 2022, pp. 463–476.

[2] F. Svanström, C. Englund, and F. Alonso-Fernandez, "Real-time drone detection and tracking with visible, thermal and acoustic sensors," in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 7265–7272.

[3] N.-D. Nguyen, T. Do, T. D. Ngo, and D.-D. Le, "An evaluation of deep learning methods for small object detection," *Journal of Electrical and Computer Engineering*, vol. 2020, 2020.

[4] M. Vrba and M. Saska, "Marker-less micro aerial vehicle detection and localization using convolutional neural networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2459–2466, 2020. DOI: 10.1109/LRA.2020.2972819.

[5] H. Wu, W. Li, W. Li, and G. Liu, "A real-time robust approach for tracking uavs in infrared videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

[6] K. Chen, J. Wang, J. Pang, *et al.*, "Mmdetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[7] S. Huang, R. S. H. Teo, and K. K. Tan, "Collision avoidance of multi unmanned aerial vehicles: A review," *Annual Reviews in Control*, vol. 48, pp. 147–164, 2019.

[8] B. Khalid, A. M. Khan, M. U. Akram, and S. Batool, "Person detection by fusion of visible and thermal images using convolutional neural network," in *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, 2019, pp. 143–148. DOI: 10.1109/C-CODE.2019.8680991.

[9] C. Li, D. Song, R. Tong, and M. Tang, "Illumination-aware faster r-cnn for robust multispectral pedestrian detection," *Pattern Recognition*, vol. 85, pp. 161–171, 2019.

[10] V. Magoulianitis, D. Ataloglou, A. Dimou, D. Zarpalas, and P. Daras, "Does deep super-resolution enhance uav detection?" In *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, 2019, pp. 1–6.

[11] E. Unlu, E. Zenou, N. Riviere, and P.-E. Dupouy, "An autonomous drone surveillance and tracking architecture," *Autonomous Vehicles and Machines Conference*, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02864552.

[12] A. Carrio, S. Vemprala, A. Ripoll, S. Saripalli, and P. Campoy, "Drone detection using depth maps," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2018, pp. 1034–1037.

[13] M. Z. Alom, M. Hasan, C. Yakopcic, and T. M. Taha, "Inception recurrent convolutional neural network for object recognition," *arXiv preprint arXiv:1704.07709*, 2017.

[14] S. Hengy, M. Laurenzis, S. Schertzer, *et al.*, "Multimodal uav detection: Study of various intrusion scenarios," in *Electro-Optical Remote Sensing XI*, International Society for Optics and Photonics, vol. 10434, 2017, 104340P.

[15] J. Huang, V. Rathod, C. Sun, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7310–7311.

[16] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[17] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[21] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[26] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *University of Montreal*, vol. 1341, no. 3, p. 1, 2009.

[27] M. Quigley, K. Conley, B. Gerkey, *et al.*, "Ros: An open-source robot operating system," in *ICRA workshop on open source software*, Kobe, Japan, vol. 3, 2009, p. 5.