Lukas Hoyer

A Robot Localization Framework Using CNNs for Object Detection and Pose Estimation

FAKULTÄT FÜR
INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

A Robot Localization Framework Using CNNs for Object Detection and Pose Estimation

Bachelor Thesis

Lukas Hoyer

January 7, 2019

Supervisor: Prof. Dr. Sanaz Mostaghim, Chair of Intelligent Systems

Advisor: Dr. Christoph Steup

Lukas Hoyer: *A Robot Localization Framework Using CNNs for Object Detection and Pose Estimation*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2019.

Abstract

External localization is an essential part for the indoor operation of small or cost-efficient robots, as they are used, for example, in swarm robotics. We introduce a two-stage localization and instance identification framework for arbitrary robots based on convolutional neural networks. Object detection is performed on an external camera image of the operation zone, providing robot bounding boxes for an identification and orientation estimation convolutional neural network. Additionally, we propose a process to generate the necessary training data. The framework was evaluated with 3 different robot types and various identification patterns. We have analyzed the main framework hyper-parameters providing recommendations for the framework operation settings. We achieved up to 98% mAP@IOU0.5 and only 1.6° orientation error, running with a frame rate of 50 Hz on a GPU.

Contents

List of Figures

List of Tables

1 Introduction

Localization is an essential problem in robotics. The knowledge, where the robot is located and how it is oriented, is substantial for its interaction with the environment. The combination of an object's position and orientation is called pose. Many navigation and motion planning algorithms rely on that information. For outdoor robotics, localization is conveniently feasible using the Global Positioning System (GPS). However, there is usually no GPS reception available indoors due to the weak GPS signal and hence other methods have to be utilized. Furthermore, indoor localization has to deal with smaller error margins as the operation zone is usually more confined.

Indoor robot localization systems can utilize several approaches to overcome these problems. For instance, they can be implemented using radio signals (e.g. Bluetooth, RFID, or WiFi), visual sensors (2D cameras, RGB-D cameras, or stereo cameras), or other on-board sensors such as laser scanners or ultrasonic ranging sensors [9]. This thesis focuses on the localization of multiple robots, in particular in the field of swarm robotics. As swarm robots are usually small and cost-efficiently built, they do not have sophisticated sensors for self-localization such as LIDAR. Therefore, an external localization approach based on cameras was chosen. Even though there already are precise and fast camera frameworks for robot localization such as the commercial Vicon system[1] and WhyCon by Krajnik et al. [25], these require a certain identification marker that is defined by the system. For some robots, the defined markers are not applicable, for instance, due to the robots geometry and size. Therefore, a more adaptable generic alternative with regard to the deployment of such a system would be beneficial for robotic labs.

This thesis presents a more flexible and camera-based robot localization system, which builds upon deep convolutional neural networks (CNN). It is inspired by related work about pose estimation utilizing CNNs as discussed in

[1] `https://www.vicon.com/`

Section 2.4. The proposed framework is built on external cameras monitoring the robot operation zone. It is able to track various robot types and differentiate multiple instances of one robot type using trained identification markers. The usage of a data-driven model-free approach allows an easy adaption of the localization system to different robot types, identification properties, and environments. Particularly, arbitrary identification markers such as colors, numbers, letters, or LED patterns can be used to differentiate robots of the same type. This may be necessary as some marker types are not usable in certain scenarios (e.g. inappropriate lighting, occlusions, or the size of the robot).

In contrast to previous work in the field of CNN pose estimation (see Section 2.4), the proposed framework considers the entire process of robot localization, explicitly including the setup phase of the system. As labeling training data is tedious and time-consuming, we propose a straightforward training data process including automation and augmentation steps (see Section 3.2).

A downside of CNNs is their high computational expense. Especially in robotics, this can be a huge problem as robots often interact with highly dynamic environments. A slow localization framework would induce a delay into the control loop, making it difficult to react fast enough to changes in the environment. We addressed this problem by optimizing our framework towards a low latency using a multi-resolution pose estimation and lightweight CNNs (see Section 3.1). The proposed localization system was evaluated on real-world data with three different robot types and various identification markers. Multiple configurations of the process and architecture were studied to provide recommendations for its operation (see Chapter 4).

This thesis is structured a follows. Chapter 2 provides the state of the art for neural networks, CNN object detection, CNN pose estimation, and training data synthesization. In Chapter 3, the proposed CNN-based robot localization framework including an efficient process for training data acquisition is presented. Different settings of the training data generation pipeline and the architecture are evaluated on real-world data in Chapter 4. Chapter 5 discusses the findings, provides resultant recommendations, and shows topics of future work.

Parts of this thesis were published in the paper "A Robot Localization Framework Using CNNs for Object Detection and Pose Estimation" [18]. All content used here was independently written by the author of this thesis.

2 State of the Art

As the proposed robot localization framework is built upon convolutional neural networks (CNNs), this chapter gives a brief overview about neural networks and convolutional neural networks in particular. Furthermore, it is shown how state-of-the-art CNNs are used to solve the object detection task, where objects in an image have to be found and classified, and how these architectures can be extended to additionally estimate the orientation of the found objects (pose estimation). Finally, it is discussed how synthetic training data can be generated for CNNs to reduce high-effort labeling.

2.1 Artificial Neural Networks in Computer Vision

Neural networks have become the most popular state-of-the-art method for many problems in computer vision. Since their first occurrence in 2012 within the image classification challenge ImageNet [5], they improved the performance of classification algorithms from about 75% accuracy in 2011 to roughly 95% in 2015 [44]. Finally, they were even able to outperform humans on ImageNet. Due to this breakthrough, neural networks were also applied to other tasks such as image segmentation [1, 3, 29, 58] or object detection [12, 28, 38, 40]. These abilities make neural networks popular in a broad field of applications in Biology, Chemistry, Geography, Robotics, Physics, and Medicine [4, 20].

Neural Networks are inspired by the human brain. They consist of multiple layers of artificial neurons, as can be seen in Figure 2.1. These neurons are connected by artificial synapses, which weight the output of neurons of the previous layer ($w_i \cdot x_i$) and transfer the information to the neurons of the next layer [42]. These neurons collect the incoming information and react to it with respect to an activation function f and a bias b, as can be seen in Figure 2.2.

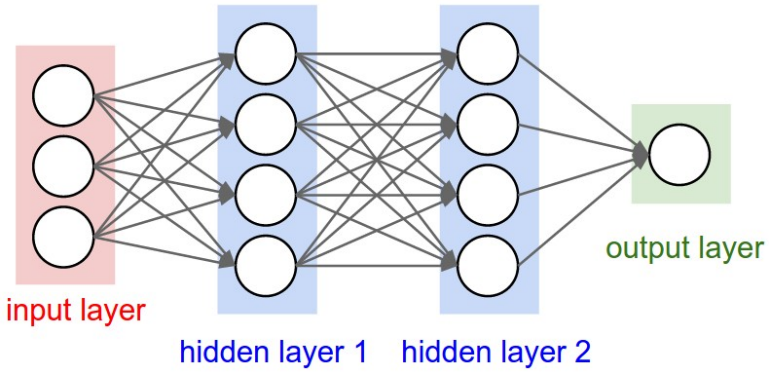


Figure 2.1: Conceptual architecture of a neural network. The input neurons fetch the information from the image, it is processed through weighted connections between the neurons of the hidden layers, and the output layer predicts the category of the image. This figure is obtained from [21].

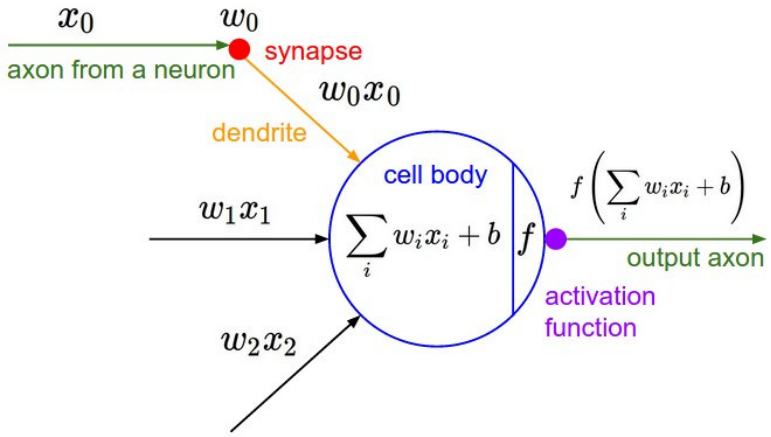


Figure 2.2: Mathematical model of a neuron. The incoming pulses x_i are weighted by synapses w_i. The neuron sums up the weighted inputs and adds a bias b. The output of the neuron is defined by this sum and an activation function f. This figure is obtained from [21].

This forward propagation for a layer with n inputs and m outputs can be described as a matrix multiplication, where $\mathbf{x} \in \mathbb{R}^n$ represents the activations of the previous layer, $W \in \mathbb{R}^{m \times n}$ the *weight matrix*, $\mathbf{b} \in \mathbb{R}^m$ the *biases* of the neurons, f the *activation function*, and $\mathbf{y} \in \mathbb{R}^m$ the output of the layer [14].

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = f\left(\begin{bmatrix} \sum_{i=1}^{n} w_{1,i} \cdot x_i + b_1 \\ \vdots \\ \sum_{i=1}^{n} w_{m,i} \cdot x_i + b_m \end{bmatrix}\right) \tag{2.1}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = f\left(\begin{bmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}\right) \tag{2.2}$$

$$\mathbf{y} = f(W \cdot \mathbf{x} + \mathbf{b}) \tag{2.3}$$

Typical activation functions for deep neural networks are, for example, the ReLU function and the sigmoid function [14].

$$f_{relu}(x) = \max(0, x) \tag{2.4}$$

$$f_{sigm}(x) = \frac{e^x}{1 + e^x} \tag{2.5}$$

There are three different kinds of layers, as visualized in Figure 2.1. The input layer fetches information from the image. For each color channel of each pixel, there is an associated neuron. The brightness of it becomes the value of the neuron. The output layer is responsible for representing the desired output. For instance, in a classification neural network, it predicts the category of the input image. The different classes are usually represented by the output neurons using an one-hot encoding, where the neuron associated with the encoded class has the value one and the others have the value zero. The hidden layers between the input and the output layer process the information and learn to detect features, which are helpful for making predictions [42]. If all neurons of a layer are connected to each neuron of the previous layer, this layer is called *fully connected layer*.

The knowledge of the neural network is stored in the weights of the synapses and the bias values of the neurons, which are adjusted during the training process [14].

For each prediction, the information of the input image is processed due to the current weights, biases, and the network architecture causing a specific output of the network. This output is compared with the desired outcome (so called *ground truth*) and an optimization algorithm adjusts the weights and biases to enforce a correct prediction. This process is repeated with a lot of training images multiple times causing the neural network to learn to solve the given problem. One whole iteration with all available images is called *epoch* and usually multiple epochs are necessary to train a network [16].

Mathematically, a loss is calculated for each training sample's prediction and ground truth using a problem-specific loss function. The gradient of the loss with respect to the network weights is back-propagated [43] layer-wise through the network. An *optimizer* (e.g. stochastic gradient descent [23], RMSprop [52], or Adam [24]) uses the gradients to update the network weights enforcing a more accurate prediction by minimizing the loss function [14]. The *learning rate* controls how much the network weights are adjusted with respect to the loss gradient. Usually several training samples (training steps) are cumulated in a so called *batch* and are processed with the same fixed model weights for performance reasons. After its completion, the network weights are updated using the loss of all training samples in this batch.

Usually, the used dataset is split into two parts. On one part the neural network is trained and on the other one the performance of the neural network is evaluated. This is necessary to verify if the neural network is able to generalize and transfer the knowledge, which it has learned on the training set, to new data it does not know [14]. There are several common metrics to evaluate the performance of a neural network on the validation dataset. For instance, the precision, recall, and accuracy are often used for classification problems. They are calculated using the number of true positives TP, false positives FP, true negatives TN, and false negatives FN:

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.7}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

For regression tasks, a commonly used metric is the mean absolute error (MAE). It is calculated for a series of predictions $\hat{Y}$ and their ground truth Y:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{Y}_i - Y_i| \tag{2.9}$$

Overfitting is a common problem in machine learning. It is caused by a model that is too closely adapted to the training dataset and, therefore, may fail to fit additional data. If overfitting occurs, the training accuracy improves but the validation accuracy drops as the neural network is learning features that are specific to the training set. In some cases, the neural network has enough parameters to memorizes the training images without learning the general concepts to classify them. Therefore, overfitting often develops for neural networks with many parameters and small training datasets [16].

There are several counter measurements. One obvious way is to use smaller networks but it has been shown that especially deep neural networks with a huge amount of training parameters achieve a higher performance than small networks if trained properly [47].

The second possibility is using a larger training dataset. This usually significantly improves the performance of neural networks [50]. Unfortunately, the acquisition of large training datasets requires expensive manual labeling and is very time-consuming. Therefore, the amount of training data is often extended by a method called *data augmentation*, where random transformations such as shift, scale, rotation, flipping, or distortion are applied to copies of the original training images [26]. Another approach is synthetically generating annotated training data, which is described in Section 2.5.

2.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are currently the most popular architecture for image processing. They use convolutions as an image filter and feature extractor. Thereto, a small matrix (the so-called *kernel*) is slid over the image and is multiplied with the underlying pixels, producing a specific

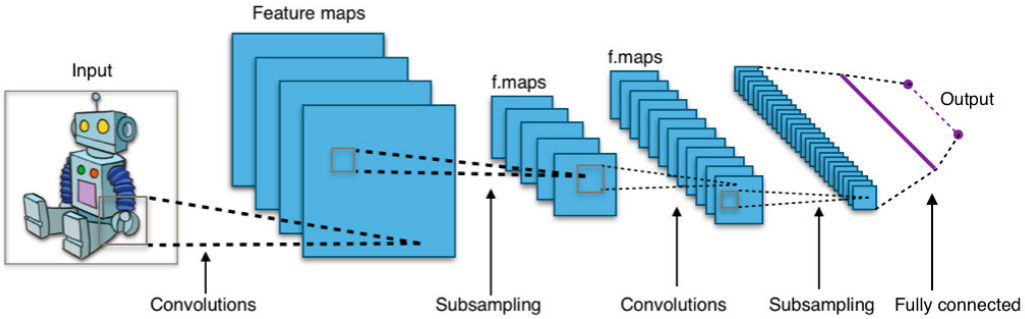


Figure 2.3: Architecture of a convolutional neural network (CNN). Convolutions with different kernels generate a set of feature maps per layer. The feature maps are subsampled to reduce the number of downstream operations. The last fully-connected layer is responsible for the classification. This figure is obtained from [55]

.

feature map of the image. In CNNs, each convolutional layer has multiple kernels and, therefore, generates several feature maps, as it is shown in Figure 2.3. The CNN learns the weights of the kernel during the training [14].

Mathematically, the *convolution* of an image $I \in \mathbb{R}^2$ and a kernel $K \in \mathbb{R}^2$ generating a feature map $F \in \mathbb{R}^2$ is defined as:

$$F(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) \cdot K(m,n) \tag{2.10}$$

The same technique is used in classical computer vision to implement image filters such as the edge detecting Sobel operator [6]. For the Sobel operator, the filter kernels are manually designed to solve the desired task. In contrast to that, CNNs learn filter kernels on their own and do not require manually designed kernels.

A *convolutional layer* sums up the convolutions of each input channel $k \in [0, c_{in})$ of the input feature maps $X \in \mathbb{R}^{c_{in} \times h_{in} \times w_{in}}$ and the corresponding kernels of the learned weight tensor $W \in \mathbb{R}^{c_{out} \times c_{in} \times h_{kernel} \times w_{kernel}}$, where c, h, and w denote the number of feature maps, their height, and their width. This process is repeated for each output channel $l \in [0, c_{out})$ producing the output feature maps $Y \in \mathbb{R}^{c_{out} \times h_{out} \times w_{out}}$ of the convolutional layer. Addition-

ally, a bias $\mathbf{b} \in \mathbb{R}^{c_{out}}$ is added to the output feature maps and an activation function f is applied.

$$Y(l) = f\left(\mathbf{b}(l) + \sum_{k=0}^{c_{in}-1} W(l,k) * X(k)\right) \tag{2.11}$$

Note the similarity to Equation 2.3 of a fully connected layer. Both variants use learned weight matrices to describe the forward pass. However, instead of connecting all neurons of subsequent layers with weighted synapses, in a convolutional layer the connections are spatially structured using the convolution operation.

CNNs usually also contain *pooling layers*, which spatially subsample the feature maps combining several neighboring pixels (see Figure 2.3). For instance, the commonly used max pooling [60] calculates the maximum for input patches of size $N \times N$ sampled with a stride of S for each feature map independently:

$$Y(k,i,j) = \max_{m\in\{0,\dots,N-1\}} \max_{n\in\{0,\dots,N-1\}} X(k, i\cdot S+m, j\cdot S+n) \tag{2.12}$$

Note that i and j are zero-based numbered. Typically, $N = 2$ and $S = 2$ are chosen. In that way, the pooling operation reduces the number of pixels, which have to be processed by higher convolutional layers, decreasing the computational complexity. Even though in this way the exact position of the feature is lost, its existence is still known, which is sufficient for classification tasks [14].

Usually, the number of feature maps increases for deeper layers to grasp more concepts. On top of the convolutional layers, there is often another fully connected network for the classification task (see Figure 2.3) [14].

A commonly used CNN architecture following the general structure of Figure 2.3 is VGG-16 [47]. It has 16 convolutional layers (kernel size 3x3) starting with 64 feature maps per layer and increasing up to 512 feature maps followed by 3 fully connected layers with 4096, 4096, and 1000 neurons. This architecture results in 138 million trainable weights. In more recent architectures, the number of trainable parameters has been reduced for the sake of convergence and inference speed. For instance, MobileNet [17], which was

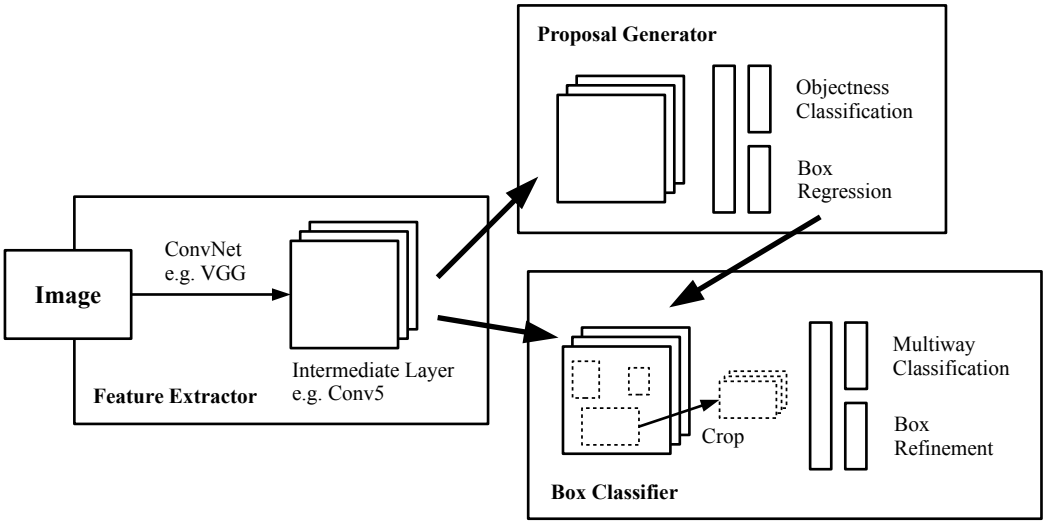
optimized for fast inference speed, only has 4.2 million parameters. This was achieved by omitting fully connected layers and utilizing depthwise separable convolutions [46].

It has been shown that the convolutional layers learn hierarchical information [59]. For instance, if a CNN is trained to recognize faces, the first layers learns to recognize simple features such as edges. They can be used by higher layers to recognize more complex shapes such as eyes, mouths, and in the highest layers even faces.

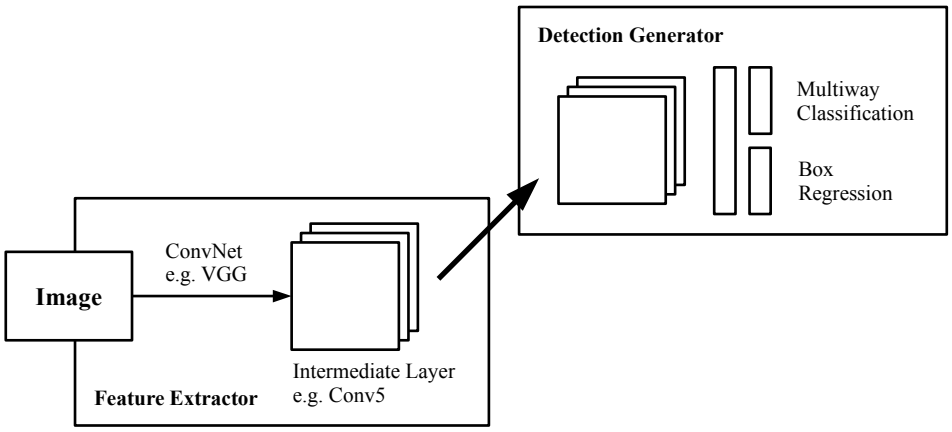
Due to the large number of trainable parameters, deep convolutional neural networks require a vast amount of training data to generalize properly [47]. As there is not enough labeled data available for most problems, a technique called transfer learning is often used. The neural network is pre-trained on a generic dataset such as ImageNet [5]. This makes sure that a sufficient amount of training data is available to train the feature detectors properly. Most of the learned knowledge like the edge and shape detection of the lower layers can be easily transferred to a new problem [34]. Therefore, only the last layers have to be trained. This drastically reduces the amount of training parameters, which makes it possible to train the neural network with a small amount of training data in less time [33].

2.3 CNN Object Detection

For object detection, an image usually contains multiple objects with potentially different classes. Object detection algorithms try to predict the bounding box and the class of each object. In recent years, neural networks have also become the state-of-the-art method for this task. One of the first successful CNN object detectors was R-CNN [12]. It utilizes an external box proposal algorithm to crop the input image and classify each crop using a neural network. However, this method is quite expensive as features have to be computed multiple times. To circumvent this problem, Faster R-CNN [40] reuses already computed features. It consists of three modules, as can be seen in Figure 2.4a. The region proposal network uses the intermediate layer of a feature extractor (e.g. VGG-16 [47]) to predict class-agnostic box proposals. The box classifier uses these box proposals to crop features from the intermediate layer and applies the remainder of the feature extractor to predict the class and a box



(a) Faster R-CNN.



(b) Single Shot Detector (SSD).

Figure 2.4: Comparison of two different CNN object detection architectures. Both use a feature extractor (e.g. VGG-16) to obtain features from an intermediate layer. While Faster R-CNN uses a proposal generator to produce class-agnostic object proposals that are classified by a box classifier, SSD performs both steps in a single detection network. This figure is adapted from [19].

refinement for each proposal. This approach significantly reduces the computation time [19]. To reduce the runtime even further, there are several neural networks that use a single feed-forward convolutional neural network such as Single Shot Multibox Detector (SSD) [28] or You only look once (YOLO) [38]. Both predict the class as well as the bounding box coordinates directly without requiring an upstream region proposal network (see Figure 2.4b). To deal with differently sized objects, Single Shot Multibox Detector also utilizes the output of intermediate feature maps with higher resolutions.

A common metric to evaluate the performance of an object detection algorithm is the *mean average precision (mAP)* for a certain minimum *intersection over union (IoU)* [8]. Using the IoU, predicted object bounding boxes are matched with ground truth bounding boxes allowing a classification into true/false positives. To be considered as correct prediction, the overlap of a prediction bounding box B_P with a ground truth bounding box B_{GT} of the same class must exceed a certain value (e.g. 0.5), where the overlap is defined as IoU:

$$IoU = \frac{area(B_P \cap B_{GT})}{area(B_P \cup B_{GT})} \tag{2.13}$$

The average precision (AP) is calculated for the precision-recall-curve as mean precision of a set of equally spaced recall levels. Afterwards, the APs for all classes are averaged resulting in the mAP.

2.4 CNN Pose Estimation

For robot localization, the position as well as the orientation of an object is required. The combination of both is called pose. In the last few years, several approaches for pose estimation, utilizing CNNs, were proposed. Most of them are based on previous work about object detection. While Kehl et al. [22], Poirson et al. [37], and Tekin et al. [51] utilize a single shot architecture as outlined in [28], the approaches described by Massa et al. [30], Oñoro-Rubio et al. [32], and Su et al. [49] are built upon R-CNN [12] and its successors Fast R-CNN [11] and Faster R-CNN [40].

Besides the underlying object detection system, these approaches can be divided into the separate and the simultaneous approach, which differ in the

integration of the orientation estimation step into the object detection framework. Glasner et al. [13], Tulsiani and Malik [54], Tulsiani et al. [53], Redondo-Cabrera and López-Sastre [39], and Poirson et al. [37] execute the orientation estimation separately after the object detection is done, while Pepik et al. [36], Xiang et al. [56], Massa et al. [30], and Xiang et al. [57] integrate the pose estimation directly into the object detection network. Both concepts were compared by Oñoro-Rubio et al. [32] in a unified framework, who found out that the separated detection and orientation estimation achieves a better performance. Additionally, some more specific aspects were researched in literature. For instance, Su et al. [49] showed that training only on synthetic data achieves comparable results to systems trained on real-world data.

Finally, these approaches can be grouped by the dimension of the estimated pose, which is determined by the dimension of the position and orientation. While [13, 30, 32, 37, 49] study 3D pose estimation (2D position and 1D orientation), [22, 51] estimate 6D poses (3D position and 3D orientation).

2.5 Training Data Synthesization

A large amount of annotated data is crucial for training deep neural networks [50]. Unfortunately, the data acquisition is a huge effort making it difficult to train deep CNNs for custom applications. To overcome this problem, training data can be synthetically generated for various problems in computer vision.

One approach is photorealistically rendering the required data as it was done by Handa et al. [15], Movshovitz et al. [31], Peng et al. [35], Richter et al. [41], and Su et al. [49] for semantic segmentation [15, 41], object detection [35], and pose estimation [31, 49]. 3D models of the relevant objects are rendered with different projections and orientations either on randomly sampled background images [31, 35, 49] or directly into entire 3D scenes [15, 41]. In that way, the ground truth for object detection and orientation estimation can be easily obtained from the placement algorithm and does not have to be labeled manually. However, it can be tedious to create realistic scenes and the image statistics may differ between rendered and real data, making generalization difficult [35].

Another approach is compositing real images by pasting image object crops into background images as described by Dwibedi et al. [7]. Georgakis et al. [10] additionally considers the scene geometry for context-sensitive placement. By using real images as source material for the composition algorithm, the image statistics may be more similar to entirely real data [7].

3 Proposed Localization Framework

In this chapter, the proposed robot localization framework is presented. First, the two-stage identification and pose estimation approach is introduced and discussed. Second, the process of acquiring training data for the system setup, which is based on superimposing background images with robot crops, is described. And third, the usage of the framework is summarized.

This chapter is based on the paper "A Robot Localization Framework Using CNNs for Object Detection and Pose Estimation" [18]. The taken passages were independently written by the author of this thesis.

3.1 Architecture

The proposed robot localization framework is based on fixed RGB cameras looking at the robot operation zone. On the recorded camera frames, we perform a two-stage image processing, consisting of an object detection localizing the robots within the scene (first stage) and an instance classification as well as an orientation estimation (second stage), as shown in Figure 3.1.

The first stage performs a low-resolution robot detection. It samples down the camera image and uses a CNN object detection to classify the type (e.g. quadcopter or wheeled robot) and to estimate the bounding box of a robot. The results of the first stage are used to feed the second stage. The original, high-resolution camera image is cropped according to the estimated bounding boxes. Depending on the predicted robot type, a corresponding second stage neural network is selected. For each robot type, there are a single instance identification and orientation estimation CNN. In the end, the output of both stages is merged resulting in a bounding box, the type, the instance identification, and the orientation of each robot.

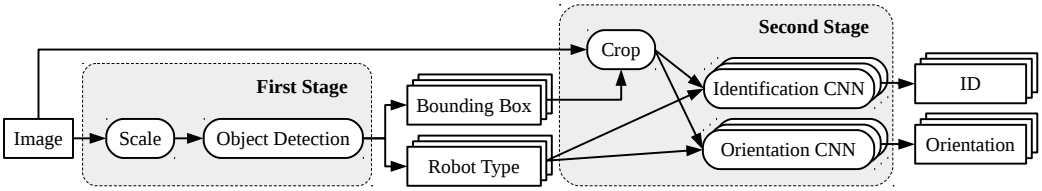


Figure 3.1: Architecture of the robot localization framework consisting of two stages. The first stage samples down the camera frame and looks for robots. The detection results are processed by the second stage. For each robot, the original high-resolution frame is cropped according to the bounding box and fed into a robot type specific identification and an orientation estimation CNN.

The orientation estimation neural network supports, both, continuous and discrete estimation. To provide a continuous orientation, a linear output activation function for regression and the mean squared error of the smallest angle difference as loss function are used. For the discrete approach, the orientation angles are separated into 360 bins resulting in a classification problem. The performance of both approaches is compared in Section 4.

As the cameras are fixed, their position relative to the ground plane can be determined. For robots moving on the floor, this information is sufficient to calculate the 3D position and the projected 1D orientation. For flying robots such as quadcopter, their height can be provided by an on-board sensor. In the experiments, only one camera is used but multiple cameras can be supported by merging the output of the framework after localization.

We have decided to use the two-stage approach as it provides several advantages over an integrated one-stage approach. Normally, robots represent only a small part of the camera frame. Therefore, most of the pixels do not contain relevant information. As comparably low-resolution features are sufficient to recognize the robot itself, the first stage robot detection provides a fast way to find significant image sections. This information can be utilized to provide high-resolution image crops for identification and orientation estimation of a detected robot, which depend on smaller features than the detection of the whole robot. The speed increase due to the first stage screening justifies the recalculation of convolutional layers with a higher resolution in the second stage. Moreover, this approach allows that the neural networks of first and sec-

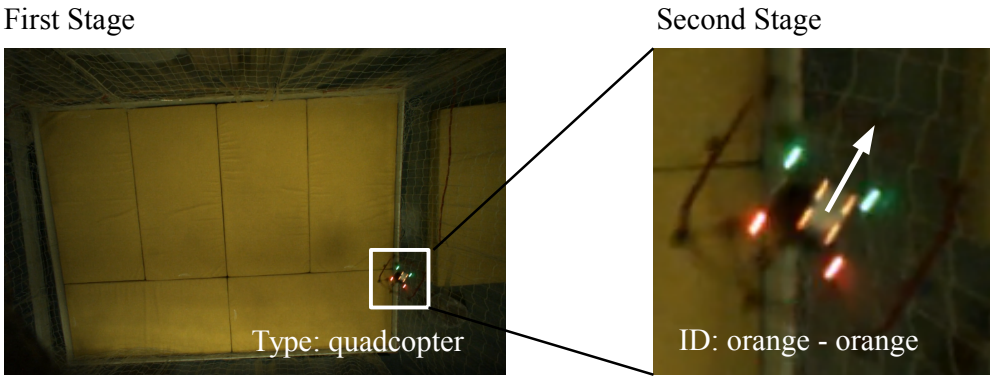


Figure 3.2: Example two stage pose estimation. The first stage detects the quadcopter in the downsampled image and the second stage determines its ID as well as its orientation in a high-resolution robot crop.

ond stage can be developed and optimized independently. Furthermore, due to the separation of identification and orientation estimation for each robot type, the CNNs are more specialized and can be chosen with less capacity as they handle less features and classes. As result of the reduced number of feature calculations, they have a lower runtime.

To provide a fast inference speed, we use state-of-the-art lightweight CNNs. For the first stage, a SSD [28] object detection architecture (see Section 2.3) with MobileNet v2 [45] as feature extractor is used. This decision is based on the comparison of different object detection meta-architectures and their feature extractors in [19]. The chosen architecture/model is the fastest configuration in the Pareto-front. For the second stage, a MobileNet is used for the same reasons. Moreover, the width of MobileNet can be easily adjusted by one parameter, allowing speed/performance adjustments [17]. We have used the TensorFlow Object Detection API[1] for the implementation of the first stage and Keras[2] to realize the second stage as they provide implementations of the used network architectures and pre-trained network weights. Problem-specific adaptations of the framework as well as the used hyper-parameters for training are discussed in Section 4.3.

[1] `https://github.com/tensorflow/models/tree/master/research/object_detection`

[2] `https://keras.io`

3.2 Training Data Acquisition Process

A crucial contribution to the performance of a neural network is the amount and quality of its training data [50]. Usually, labeling enough images to train a CNN requires a lot of effort, which is not reasonable for setting up a robot localization framework. To avoid this problem, we synthesize training data by superimposing image crops, containing robots, on background images. The necessary robot crops are extracted from images of robots in the working area, which were recorded with the localization camera during a setup phase.

3.2.1 Compositor

The compositor is an algorithm that superimposes robot crops on background images with random scale, rotation, and position. It is similar to the approach described in [7] but it uses a model-driven crop algorithm (see Section 3.2.2) instead of a trained semantic segmentation. The compositor can generate a large amount of data with multiple robots in one frame and arbitrary robot

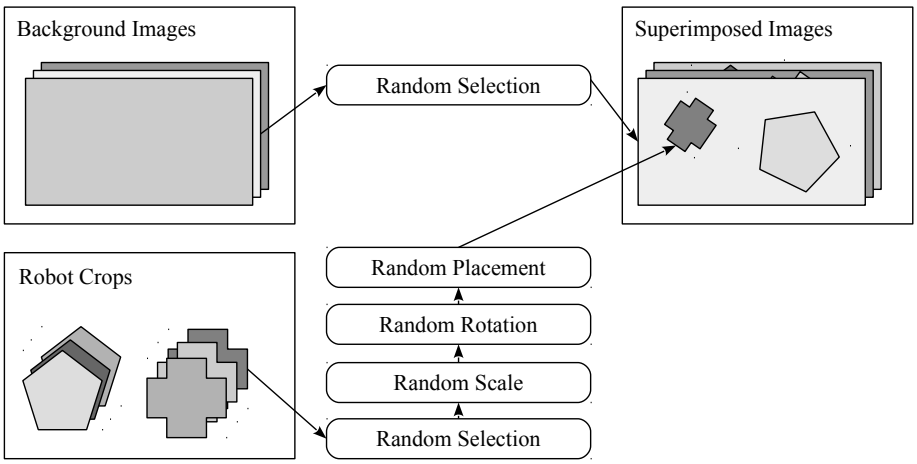


Figure 3.3: Compositor flowchart. The compositor randomly selects background images and robot crops to superimpose them. Random transformations (scale, rotation, and translation) are applied to the crops according to the properties of the respective robot type.

position, orientation, size, and occlusion (see Figure 3.3). Its main advantage is the known ground truth information of the synthesized data, which saves a huge amount of time for labeling the images.

There are other methods to synthesize data such as rendering training images (see Section 2.5). But they have the disadvantage that it is quite difficult to find parameters to match the lighting in the robot working zone and the camera properties. Moreover, they require 3D models of all robots and material models for photo-realistic rendering. Therefore, it is less effort to use real images of robots and to crop them.

Even though the amount of generated data is theoretically unlimited, the diversity of the synthesized data is restricted by the provided backgrounds and robot crops. Moreover, the robot crop does not necessarily match with the background as in a real image. This can affect lighting, blur/blending, and crop/compositing artifacts as well as contextual information, which could influence the performance of the CNNs [10]. For example, an occluded robot could be detected by its shadow that isn't generated by the compositor. However, these are only minor problems that are worthwhile to minimize the effort of labeling real images.

In order to minimize the bias induced by the class imbalance of the training data [2], the compositor balances all important factors for the experiments such as background types, robot types, and identification patterns within one robot type. For each generated frame, a random background is chosen on which one to four randomly selected robot crops are composited with random scale, orientation, and translation. The bounds of the robot size depend on possible distances of the robot to the camera. For each robot in a frame its type, identification, bounding box, and orientation are stored as ground truth for the training.

3.2.2 Robot Crops

Robot crops are the essential factor for the quality and the necessary setup effort generating the final composited data. To extract them, a manual or an automatic approach may be used.

For the manual robot crop (see Figure 3.4), an image sequence of a non-moving robot with a static or a changing identification pattern (e.g. realized

Figure 3.4: For the manual crop a binary mask (middle) is manually created for each image sequence. It is applied to every frame of the image sequence of the stationary robot (left), generating multiple robot crops (right) with a single mask.

Figure 3.5: For the automatic crop an image of the background without robot (left) is recorded. It is used in a background subtraction algorithm to extract the robot crops (right) from the image sequence with robot (middle). Note that the automatic crop still contains some background due to the shadow of the robot on the ground.

with LEDs) is captured. For each image sequence, one single instance mask is manually created using an image editing tool. Later, this mask is applied to all images of the sequence.

The automatic robot crop (see Figure 3.5) is based on background subtraction. First, an image of the background without robot is captured. After that, the robot is placed in the scene and an image sequence is recorded. Finally, the background image is subtracted from the image sequence followed by thresholding and morphological operations to generate an instance mask for each frame. Although, using the automatic method, robots could even move during the image sequence, only non-moving robots were captured due to comparability between both techniques.

To properly superimpose the robot crops on arbitrary backgrounds, the crop itself should not contain any background of the operation zone. Therefore, in both methods, the generated masks are used as alpha channel for each image in the sequence with the robot. The image is cropped using the bounding box of the outer contour of the robot mask. It has to be ensured that the robot crops are aligned in the same direction. This can either be done before the recording or by rotating the crops later. We recommend aligning the robot in scene as it saves some time.

In the end, the crops are grouped according to robot type and identification pattern. For static patterns such as printed letters, frame colors and so on, the image sequence should be labeled accordingly during the recording process. If the identification pattern is changing during the image sequence (e.g. a sequence of LED identification patterns), the identification of the first frame is determined manually and an algorithm, which detects changes of the identification pattern, labels the rest of the frames according to a predefined sequence of identifications (e.g. black, red, and blue).

3.3 Framework Usage Overview

The proposed robot localization framework can easily be applied to a custom scenario. First, the compositor data has to be acquired. The necessary background images can be obtained from a generic dataset such as MS COCO [27] or from recordings of the operation zone without robot. Depending on the user's preference, automatic or manual crops can be used. For automatic crops, an image of the operation zone without robot and an image sequence with robot (arbitrary location but defined orientation) are captured. For manual crops, an image sequence of a non-moving robot (arbitrary location but defined orientation) is recorded and one mask for the entire sequence is created using image editing software. Independently of the crop method, the sequences are labeled with robot type and identification pattern and the respective crop algorithm is run.

Second, the compositor is executed with the preferred settings for the first and the second stage (recommendations based on our evaluation are listed in Table 4.4). After that, both stages are trained on the generated compositor data. Finally, the trained CNNs are deployed in the framework pipeline (see Figure 3.1).

4 Evaluation

In this chapter, the evaluation of the proposed robot localization framework is discussed. First, the robot types and identification patterns used for evaluation are presented. Second, the generation of the training data and the acquisition of the validation data are described. After that, the training setup for the experiments is explained. And finally, the results for the first stage, the second stage, and a combined setup are analyzed.

This chapter is based on the paper "A Robot Localization Framework Using CNNs for Object Detection and Pose Estimation" [18]. The taken passages were independently written by the author of this thesis.

4.1 Evaluation Environment

We used three different robot types in our experiments: quadcopters, Spheros[1], and a Kuka YouBot[2] (see Figure 4.1). The quadcopters have 4 position LEDs (green and red) next to their rotors as well as four RGB identification LEDs in the center. The Spheros are spherical robots driven by a weight moving inside their transparent hull. They provide one blue position LED at the rear and one RGB identification LED in the center. The YouBot is a wheeled robot with a manipulation arm in the front. The robot operation zone was recorded by a Basler acA1600-60gc standard industry camera, which was mounted on the ceiling.

As identification properties for our experiments, we used 15 different LED patterns as well as three letters (A, B, and C) for the quadcopter, eight different LED colors for the Sphero, and three letters for the YouBot. The identification

[1] `https://www.sphero.com/sprk-plus`
[2] `http://www.youbot-store.com/`

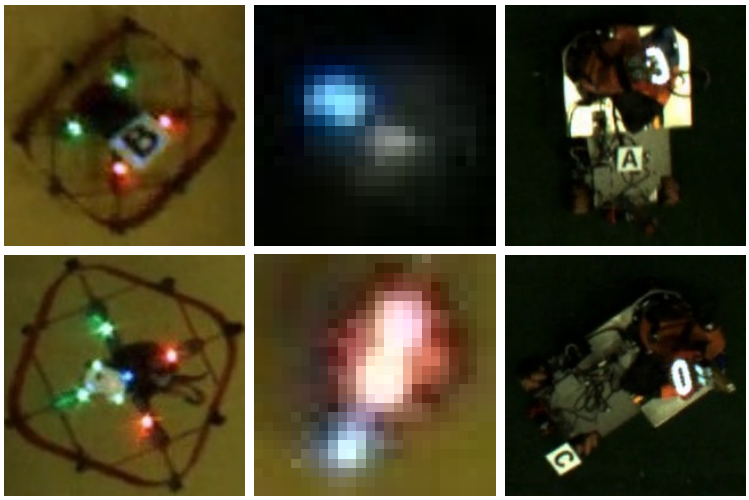


Figure 4.1: Examples of each robot type with different identification patterns. From left to right: quadcopter (marker B and LED pattern blue-green), Sphero (LED black and LED bright red), and YouBot (maker A and marker C). Notice that the blue LED at the Spheros marks their rear, that the marker is attached at different positions at the YouBot, and that the green position LED outshines a blue identification LED of the bottom quadcopter.

markers of the YouBot were attached either in the center or on top of a pole at the back. Some example patterns are shown in Figure 4.1.

We have chosen these robot types and identification patterns as they provide versatile challenges for the localization framework and, therefore, indicate that it can be applied to a high variety of robots. The Spheros are quite small (about 25x25 pixels in the camera image) and do not offer the possibility to attach an identification marker due to their shape and locomotion. The quadcopters have a medium but varying size (depending on their altitude) and provide many identification patterns, which are in some cases difficult to distinguish due to overexposure, reflections, spatial separation of the single LED color channels, blending with the position LEDs, and occlusion (see Figure 4.1). Finally, the YouBot provides an example for a bigger robot (about 200x120 pixels) with many visual features.

4.2 Training and Evaluation Dataset

For the training process, we have captured robot crops under different lighting conditions (artificial lighting and natural lighting), different locations of the robot in the operation zone (each corner and in the middle), and different floor colors (quadcopter yellow, Sphero yellow and green, YouBot green). For the compositor, 1524 images of the operation zone and 25,608 randomly sampled images from the 2017 training dataset of MS COCO [27] were used as backgrounds (see Figure 4.2 for examples). Which source material should be used for the compositor and how many training images should be generated, is analyzed in the experiments (see Section 4.4).

Depending on the stage of the framework, the training images are further processed. The first stage exploits random horizontal and vertical flips and SSD random crop [28] to augment the dataset. The data for the second stage is cropped using the ground truth boxes with a variance from -10% (inwards) to +15% (outwards) and prepared according to the second stage pipeline (see Section 3.1).

The evaluation dataset for the first stage consists of 1400 images per robot type. The second stage evaluation set contains 110 images per identification pattern of each robot. Both datasets consist of real-world images and they contain the same amount of frames with natural/artificial lighting, floor colors (Sphero) and pattern position (YouBot) for each robot type. The images of the evaluation set were semi-automatically labeled and manually corrected.

As the normal use case contains few objects in the robot working zone, additional objects were added to mislead the first stage and evaluate its specificity. We have chosen to use synthetic objects, which were extracted from the eval2017 dataset of MS COCO [27], as they provide a good variety of decoy objects. Three randomly chosen decoys were superimposed at a random position and rotation on each validation image while it was ensured that they do not occlude the robots.

4.3 Experiment Setup

We have chosen the experiments, presented in this section, to provide recommendations for a good configuration of the localization framework. Therefore,

(a) Background Images

(b) Robot Crops

(c) Superimposed Images

Figure 4.2: Example backgrounds, robot crops, and generated images for training. (a) The necessary background images for the compositor were obtained from scenario-specific recordings (top) and a generic image dataset (bottom). (b) A selection of manual robot crops with different robot types, identification patterns, and lighting conditions is shown. (c) The training data is generated by superimposing background images with robot crops.

Table 4.1: Default experiment configuration for first and second stage

	First Stage	Second Stage
Batch size	16	32
Optimizer	RMSprop [52]	Adam [24]
Learning rate	0.004/0.0004 (after 15k steps)	0.0004
Input size	400x300	128x128
Training data	5k images	2k crops per id
Experiment repetitions	5	8
Evaluation period	step 20k - 25k	epoch 15 - 25

we have evaluated the main parameters of the compositor as well as important hyper-parameters of the two framework stages. For both stages, a default/reference configuration was chosen as base level. It is annotated with F/S 0. In all experiment abbreviations, F means first stage and S stands for the second stage. Table 4.1 lists all important default settings and Table 4.2 illustrates which parameters were changed in each experiment configuration. The optimizer and the learning rate (see Section 2.1) were both determined in preliminary experiments. The batch size, which specifies the number of steps used for a weight update, was chosen in order to optimize training speed. The experiments were repeated several times (see Table 4.1 row experiment repetitions) using different random seeds for training data shuffling and weight initialization to analyze the frameworks robustness with respect to these factors. After the network had converged, a certain number of steps was used for evaluation (see Table 4.1 row evaluation period).

The first three experiment configurations (F/S 1 – F/S 3) analyzed the behavior of both stages with respect to the compositor settings.

- In F/S 1, the influence of the background diversity of the generated images was researched. Therefore, training data generated with backgrounds only of the robot working zone (SwarmLab), only of MS COCO, and of both combined was studied. We wanted to figure out whether a background set with higher diversity (MS COCO) can improve the learning process and if the framework has to be trained with scenario-specific backgrounds.

- F/S 2 examined the effect of the amount of composited images to analyze the number of useful images when their impact on the performance is limited by the diversity of crops and backgrounds.
- In the third experiment (F/S 3), the influence of the crop method was surveyed determining whether a more precise mask generated by manual labeling justifies its additional labeling cost or if the more versatile automatic crops can even improve the performance of the framework.

The subsequent experiments were conducted to analyze hyper-parameters of the framework architecture.

- In F 4, the input resolution of the first stage was studied to find a good speed-accuracy trade-off.
- For the same reason, in S 4, the influence of the MobileNet width multiplier α [17] of the second stage was researched.
- Finally, S 5 evaluated the performance of continuous and discrete pose estimation.

For the first stage, SSD with MobileNet v2, which was pre-trained on MS COCO from [19], was used. To adapt the pre-trained network to our problem, the number of neurons of the classification output layer was adjusted to match the number of robot types and the entire network was fine-tuned with a low learning rate. The chosen hyper-parameters for the training can be seen in Table 4.1. After 20,000 training steps (one batch iteration), the network wasn't improving anymore. The evaluation data for the experiments was extracted from step 20,000 to step 25,000 every 250 steps.

The second stage is based on network weights that were pre-trained on ImageNet provided by Keras. As the network wasn't improving after 15 dataset iterations (epochs), the evaluation was based on epoch 15 to 25. The MobileNet input size is 128 because bigger robot crops do not provide more necessary details as the robots itself are usually smaller than 200x200 pixels.

The first stage detection performance was evaluated with Pascal VOC 2010 mAP@0.5IOU [8] (see Section 2.3) and for the second stage the classification accuracy as well as the mean absolute error of the smallest angle difference were used.

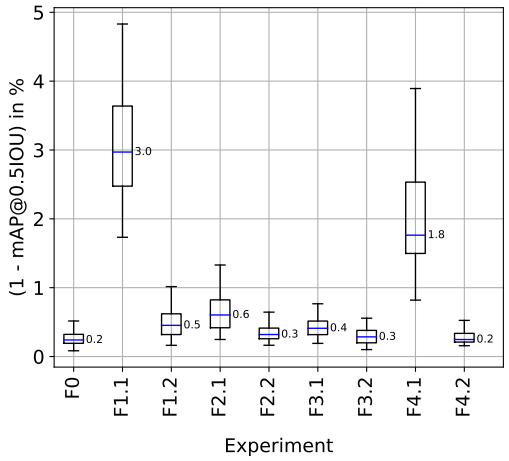
Table 4.2: Experiment overview

Experiment	Description
F/S 0	SwarmLab and COCO compositor backgrounds
F/S 1.1	only SwarmLab compositor backgrounds
F/S 1.2	only COCO compositor backgrounds
F/S 2.1	1/5 of the default image amount
F 0	default image amount (see Table 4.1)
F/S 2.2	4 times of the default image amount
F/S 0	manual crop
F/S 3.1	automatic crop
F/S 3.2	automatic and manual crop
F 4.1	first stage resolution of 200x150 pixels
F 0	first stage resolution of 400x300 pixels
F 4.2	first stage resolution of 800x600 pixels
S 4.1	MobileNet width multiplier $\alpha = 0.25$
S 0	MobileNet width multiplier $\alpha = 0.5$
S 4.2	MobileNet width multiplier $\alpha = 0.75$
S 4.3	MobileNet width multiplier $\alpha = 1.0$
S 0	discrete orientation estimation (classification)
S 5	continuous orientation estimation (regression)

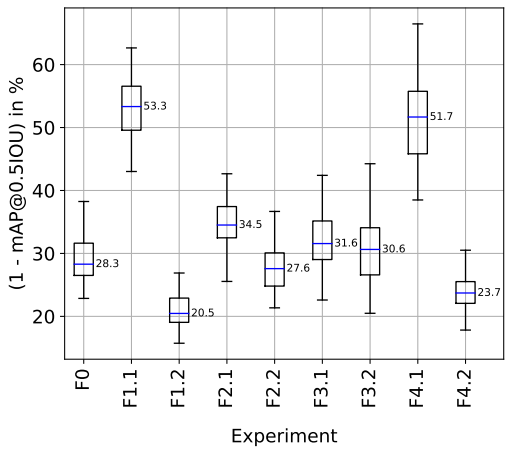
4.4 Results

4.4.1 First Stage

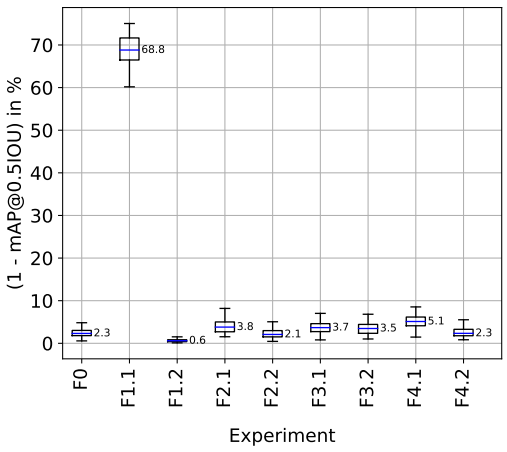
In this section, the performance of the first stage is evaluated. With the best configuration, it reaches a median mAP@0.5IOU of 99.8% (copter) / 79.5% (Sphero) / 99.4% (YouBot). Compared to the other robot types, the detection



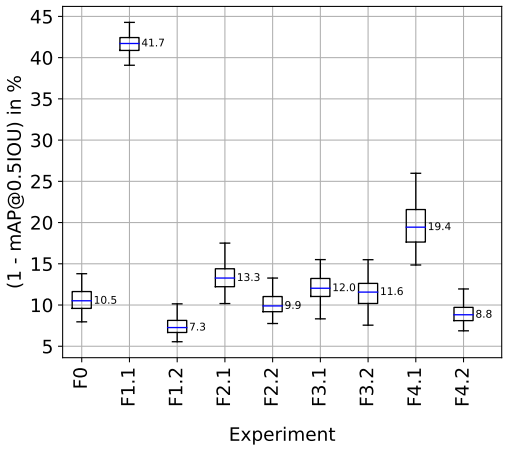
(a) First stage detection mean average precision for quadcopters.



(b) First stage detection mean average precision for Spheros.



(c) First stage detection mean average precision for YouBots.



(d) First stage detection mean average precision for all robots.

Figure 4.3: Performance comparison of the first stage.

rate of the Sphero is quite low. This has several reasons. First of all, the Spheros are considerably smaller, which means that more precise bounding boxes in relation to the image size are necessary to achieve an intersection over union of at least 50%. Moreover, the Spheros have only few features and, in addition, there are some objects containing similar features such as LEDs, light bulbs, or light reflections. So it is a quite challenging task to detect the Spheros among other objects.

In the experiments, the first stage performance with respect to different configurations was compared, as can be seen in Figure 4.3. One important factor is the background set for the compositor. It was figured out that only MS COCO backgrounds (F1.2) work better (lower median and less variance) than only SwarmLab backgrounds (F1.1) as the SwarmLab backgrounds do not contain other decoys while COCO contains a variety of different objects resulting in a better generalization. Another important factor is the number of composited images (F2.1 / F0 / F2.2). Generally, the performance of the first stage was increased by using more images, as it is also suggested in [50].

As crop method, manual cropping (F0) was figured out to achieve a lower median and variance in comparison to automatic crop (F3.1) and both (F3.2). Nevertheless, the difference is tolerable and, therefore, it may be reasonable to use automatic crops to speed up the setup of the framework.

In the last experiment for the first stage, different input resolutions were compared. The presumption that larger robots need less resolution for detection was confirmed. Even with 200x150 pixels (F4.1), the YouBot achieved good results while 400x300 pixels (F0) provided the best mAP@0.5IOU for copters. Small robots such as the Sphero need a high resolution of 800x600 pixels (F4.2) to provide the best performance as more small features can be exploited by the CNN. However, the input resolution substantially affects the inference speed of the first stage, as can be seen in Table 4.3. Therefore, we have chosen 400x300 pixels as default configuration to provide the best speed/accuracy trade-off for

Table 4.3: Benchmark of the first stage

First stage resolution	200x150	400x300	800x600
Runtime on Xeon E3-1230 v3 (ms)	27.36	76.35	322.11
Runtime on GTX 1080 (ms)	10.27	12.26	22.89

the first stage. In that case, the first stage takes 12 ms on a Nvidia GeForce GTX 1080 or 76 ms on an Intel Xeon E3-1230 v3 @ 3.30GHz.

All in all, we recommend to generate about 20,000 images (depending on scenario, number of crops, and backgrounds), use a diverse background set (no specific backgrounds of robot working zone are necessary), and an input resolution of 400x300 pixels for the first stage.

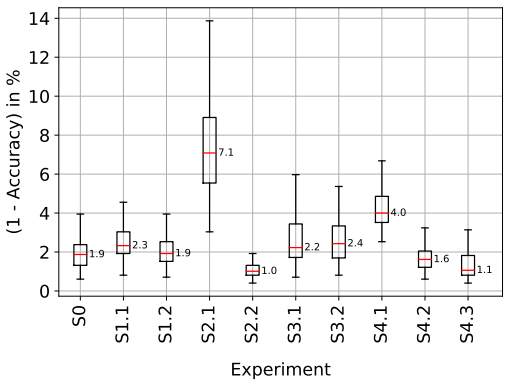
4.4.2 Second Stage

The second stage achieves good results for its instance identification and orientation estimation. With the best configuration, it reaches a median instance identification accuracy of 98.9% (copter) / 96.4% (Sphero) / 98.8% (YouBot) and a mean absolute orientation error of 1.6° (copter) / 11.2° (Sphero) / 2.3° (YouBot).

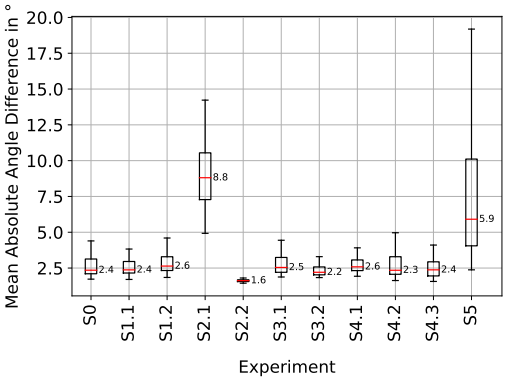
The comparably low instance identification and orientation estimation results of the Sphero are probably caused by its locomotion as the LED board is sometimes lopsided and the identification LED is not properly visible. The same is true for the position LED, which complicates the orientation estimation. Moreover, the position LED is in some cases outshone by a bright identification LED color.

In the experiments, it was found out that a higher amount of composited images (S2.1 / S0 / S2.2) significantly improves the instance identification accuracy and the orientation estimation for all robot types, as shown in Figure 4.4. The number of images is the most important factor influencing the performance of the framework decreasing both the median and the variance of the error. It should be noted that even the lowest amount of generated images exceeds the number of crops by far.

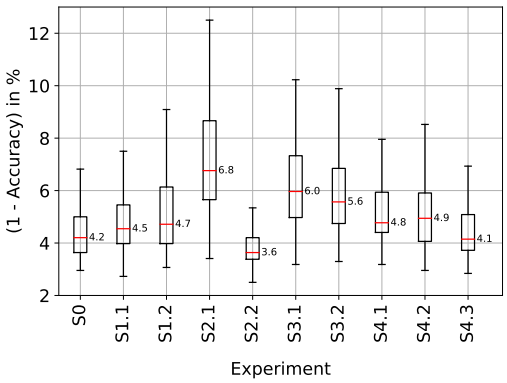
The second most important factor is the network size researched in S4.1 / S0 / S4.2 / S4.3. A network with more feature maps improves the instance identification performance especially for the YouBot a lot while it has less influence on Spheros (see Figure 4.4a - 4.4c). This could be caused by the amount of visual features of the different robot types. There are more features necessary to classify the letter markers of the YouBot than the single RGB identification LED of the Sphero. Therefore, more features must be learned by the neural network, which requires a broader architecture with more feature maps. For



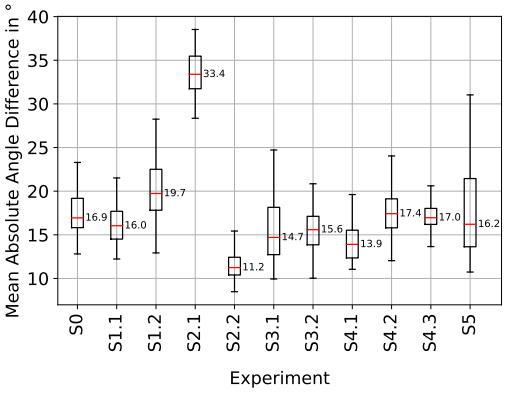
(a) Second stage identification performance for quadcopters.



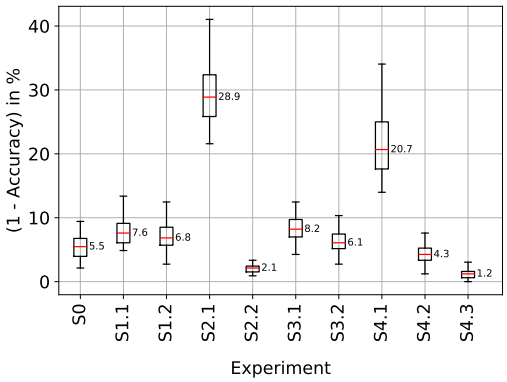
(d) Second stage orientation estimation performance for quadcopters.



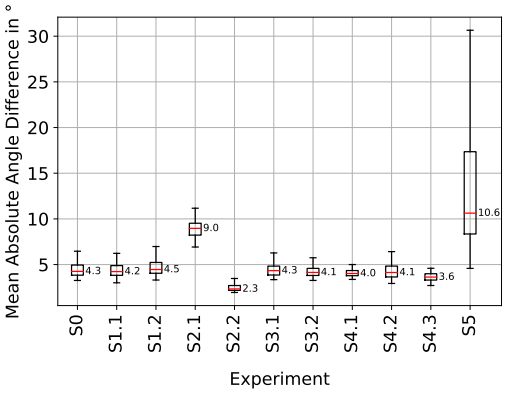
(b) Second stage identification performance for Spheros.



(e) Second stage orientation estimation performance for Spheros.



(c) Second stage identification performance for YouBots.



(f) Second stage orientation estimation performance for YouBots.

Figure 4.4: Performance comparison of the second stage for identification and orientation estimation with respect to Table 4.2.

Table 4.4: Recommended configurations for the framework

	First stage	Second stage
Composited images	20k per type	8k per ID
Backgrounds	MS COCO	MS COCO + SwarmLab
Crop method	user preference	user preference
Input resolution	depending on robot size	—
Network width	—	depending on problem complexity

orientation estimation, a bigger network for Spheros even deteriorates the performance (see Figure 4.4e), probably as there are less features necessary for the orientation estimation and larger networks tend to overfit the problem.

The influence of the background and the crop method are less substantial. Using all available backgrounds (S0 vs S1.1 and S1.2) improves the instance identification performance slightly (see Figure 4.4a - 4.4c) as it adds variance to the training dataset. For identification purpose, only manual crops (S0 vs S3.1 and S3.2) work slightly better. In experiment S5, continuous and discrete pose estimation were compared. It was found out that regression works significantly worse. This tendency is also described in [32], especially, when the amount of training images is not large enough.

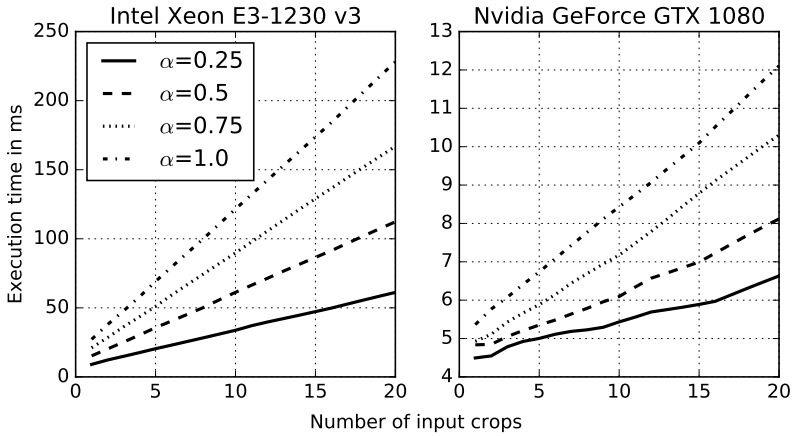


Figure 4.5: Runtime analysis of the second stage for different MobileNet width multipliers α and number of robots (100 runs; standard deviation about 45% on CPU and 24% on GPU).

The inference time of the second stage is mainly affected by the width multiplier α and the number of robots detected by the first stage (see Figure 4.5). For the default configuration ($\alpha = 0.5$) and an amount of 10 robots within the field of view of the camera, the second stage consumes 6 ms on a Nvidia GeForce GTX 1080. So both stages spend about 20 ms for one camera frame allowing a frame rate of 50 Hz, which is similar to [37].

To put it in a nutshell, we recommend the following settings to achieve a good second stage performance. Generating more images can increase the performance a lot. The network size should be adapted to the task complexity providing enough feature maps to tackle the problem but not too much to prevent overfitting. If possible, multiple background sources should be used. The crop method does not influence the performance significantly and can be chosen according to the own preference towards setup effort and system performance. An overview of the recommended settings for both stages is shown in Table 4.4.

4.4.3 Total Performance

Finally, we have chosen the best performing configurations of the experiments of both stages (see Section 4.4.1 and 4.4.2) for an integrated performance evaluation of the framework. For the first stage, the setting of F1.2 (trained only on COCO backgrounds) and, for the second stage, the setting of F2.2 (8,000 training images per identification) were deployed. The framework was evaluated on complete video streams of the camera under application conditions with 10 Hz frame rate. Even though the evaluation was done offline on a recorded video, the image characteristics of the video are similar to the camera stream itself, so that the results of this evaluation can be transferred to a live system.

In Table 4.5, the mAP@0.5IOU for the robot type and instance detection of the whole framework as well as the mean absolute error (MAE) of the orientation

Table 4.5: Performance of the whole framework

	Copter	Sphero	YouBot
mAP@0.5IOU	97.9%	70.0%	96.6%
Orientation MAE	1.6°	11.9°	2.6°

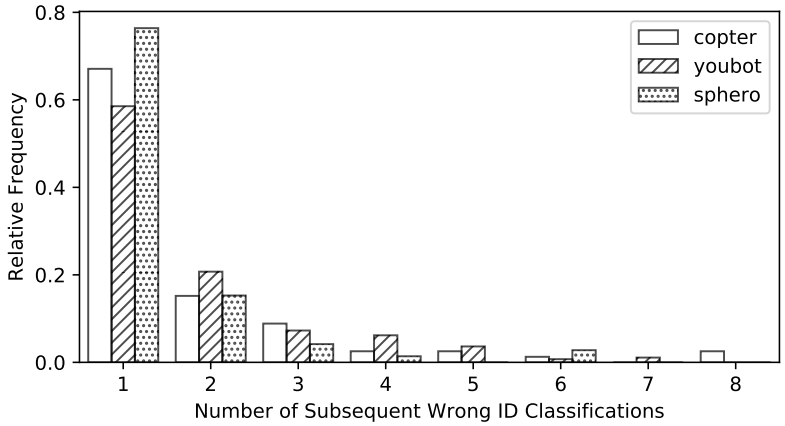


Figure 4.6: Distribution of successive wrong robot type/instance detections.

for correctly detected robots are shown. The good results of both stages are maintained in the framework. Depending on robot type, it achieves 70% - 98% detection mAP@0.5IOU and 2° - 12° orientation error.

Moreover, the number of successive wrong detections/classifications in a row was evaluated. The results are illustrated in Figure 4.6. It can be seen that the framework misses the correct classification of a robot only a few frames in a row (usually less than five), which can easily be post-processed e.g. by an Extended Kalman Filter [48] to improve the performance even further.

5 Conclusions

In this thesis, an adaptive localization and identification framework for swarm robots is presented. It is based on cameras placed in the operation zone of the robots. Its key features can be summarized as follows:

- Due to the utilization of deep convolutional neural networks, the framework is able to handle arbitrary robot types as well as identification markers.
- A video stream with and without robot (known type, ID, and defined orientation) is sufficient for the proposed training data generation process, allowing a straightforward system setup.
- By using a multi-resolution approach as well as lightweight CNNs, the framework consumes only 20 ms processing time on a GPU, enabling control tasks.

We have evaluated the framework on real-world data of three robot types and various identification markers to show its adaptability. Overall, the performance of the system is very good and provides robot pose tracking accuracy for a minimum intersection over union of 0.5 with less than 4% of identification error and an orientation error lower than 3° for two of three robot types. The higher error of the third robot type is caused by its physical properties and is not induced by the system itself.

During the analysis the properties of the training data generation process and the localization framework, we have found out that the amount of generated images is an essential factor for the system's performance, especially for the second stage, even though it highly outnumbers the number of robot crops. Furthermore, a generic background dataset is sufficient for training the localization framework. Scenario-specific backgrounds do not have to be recorded, simplifying the system setup. The manual crop slightly outperforms the automatic crop. However, the performance difference is relatively small, so the user may justifiably choose automatic crop for a lower setup effort.

In future work, the localization framework can be generalized to use multiple cameras extending the possible robot operation area as well as avoiding occlusions. Moreover, the compositor can be extended to make it aware of perspective and consequently allowing more flexible camera positioning. In this context, the framework could be extended to visually estimate the distance of the robots to the camera without additional sensor input. Further, we aim to improve the automatic crop to reach the performance of the manual crop as well as to analyze a single stage pose estimation architecture and other CNNs.

Bibliography

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[2] Gustavo EAPA Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[4] Paul Dan Cristea. Application of neural networks in image processing and visualization. In *GeoSpatial Visual Analytics*, pages 59–71. Springer, 2009.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[6] Richard O Duda and Peter E Hart. Pattern classification and scene analysis. *A Wiley-Interscience Publication, New York: Wiley, 1973*, 1973.

[7] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *The IEEE international conference on computer vision (ICCV)*, 2017.

[8] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[9] HU Feng. *Emerging Techniques in Vision-based Indoor Localization*. PhD thesis, The City University of New York, 2015.

[10] Georgios Georgakis, Arsalan Mousavian, Alexander C. Berg, and Jana Kosecka. Synthesizing training data for object detection in indoor scenes. *arXiv preprint arXiv:1702.07836*, 2017.

[11] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE CVPR*, pages 580–587, 2014.

[13] Daniel Glasner, Meirav Galun, Sharon Alpert, Ronen Basri, and Gregory Shakhnarovich. Viewpoint-aware object detection and continuous pose estimation. *Image and Vision Computing*, 30(12):923–933, 2012.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[15] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding real world indoor scenes with synthetic data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4077–4085, 2016.

[16] Simon S. Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[18] Lukas Hoyer, Christoph Steup, and Sanaz Mostaghim. A robot localization framework using cnns for object detection and pose estimation. In *IEEE Symposium Series on Computational Intelligence*, 2018.

[19] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, 2017.

[20] Chi Leung Patrick Hui. *Artificial Neural Networks-Application*. InTech, 2011.

[21] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition - neural networks part 1. `http://cs231n.github.io/neural-networks-1/`, 2015. MIT License; Accessed: 2018-10-07.

[22] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *IEEE ICCV*, 10 2017.

[23] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Tomáš Krajník, Matías Nitsche, Jan Faigl, Petr Vaněk, Martin Saska, Libor Přeučil, Tom Duckett, and Marta Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3-4):539–562, 2014.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[28] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, pages 21–37. Springer, 2016.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[30] Francisco Massa, Renaud Marlet, and Mathieu Aubry. Crafting a multitask CNN for viewpoint estimation. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 91.1–91.12. BMVA Press, 2016.

[31] Yair Movshovitz-Attias, Takeo Kanade, and Yaser Sheikh. How useful is photo-realistic rendering for visual learning? In *European Conference on Computer Vision*, pages 202–217. Springer, 2016.

[32] Daniel Oñoro-Rubio, Roberto J López-Sastre, Carolina Redondo-Cabrera, and Pedro Gil-Jiménez. The challenge of simultaneous object detection and pose estimation: a comparative study. *arXiv preprint arXiv:1801.08110*, 2018.

[33] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[34] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[35] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286, 2015.

[36] Bojan Pepik, Michael Stark, Peter Gehler, and Bernt Schiele. Teaching 3d geometry to deformable part models. In *IEEE CVPR*, pages 3362–3369, 6 2012.

[37] Patrick Poirson, Phil Ammirato, Cheng-Yang Fu, Wei Liu, Jana Kosecka, and Alexander C. Berg. Fast single shot detection and pose estimation. In *International Conference on 3D Vision (3DV)*, pages 676–684, 10 2016.

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[39] Carolina Redondo-Cabrera and Roberto López-Sastre. Because better detections are still possible: Multi-aspect object detection with boosted hough forest. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 63.1–63.12. BMVA Press, 9 2015.

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[41] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV*, pages 102–118. Springer, 2016.

[42] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[43] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE CVPR*, pages 4510–4520, 2018.

[46] Laurent Sifre and Stéphane Mallat. *Rigid-motion scattering for image classification*. PhD thesis, Citeseer, 2014.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Harold Wayne Sorenson. *Kalman filtering: theory and application*. IEEE, 1985.

[49] Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *IEEE ICCV*, pages 2686–2694, 2015.

[50] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *IEEE ICCV*, pages 843–852, 2017.

[51] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. *CoRR*, abs/1711.08848, 2017.

[52] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[53] Shubham Tulsiani, João Carreira, and Jitendra Malik. Pose induction for novel object categories. In *IEEE ICCV*, pages 64–72, 12 2015.

[54] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *IEEE CVPR*, pages 1510–1519, 2015.

[55] Wikimedia Commons. Typical cnn architecture. `https://commons.wikimedia.org/wiki/File:Typical_cnn.png`, 2015. CC BY-SA 4.0 License; Accessed: 2018-10-07.

[56] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision*, pages 75–82, 3 2014.

[57] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *CoRR*, abs/1711.00199, 2017.

[58] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[59] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[60] Y-T Zhou, Rama Chellappa, Aseem Vaid, and B Keith Jenkins. Image restoration using a neural network. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1141–1151, 1988.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Lukas Hoyer

Magdeburg, January 7, 2019