TECHNISCHE UNIVERSITÄT DRESDEN

Factuly of Computer Science Department of Artificial Intelligence, Institute of Computer Vision

ANNOTATION OF LAPAROSCOPIC SURGERY IMAGES WITH ONLINE PROPOSAL GENERATION

Florian Blume

Born on: 12.08.1991 in Karlsruhe

Matriculation year: 2012

GROSSER BELEG

Referee

Dr. Eric Brachmann

Supervisor

Dr. Dmitrij Schlesinger

Supervising professor

Dr. Dmitrij Schlesinger

Submitted on: August 14, 2018

Defended on: Not yet

TECHNISCHE UNIVERSITÄT DRESDEN

Factuly of Computer Science Department of Artificial Intelligence, Institute of Computer Vision

TASK FOR THE PREPARATION OF A GROSSER BELEG

Name: Florian Blume

Matriculation year: 2012

Title: Annotation of Laparoscopic Surgery Images with Online Proposal Generation

OBJECTIVES OF WORK

During laparoscopic surgery, a surgeon operates with special tools through the closed abdominal wall based on the view of an endoscopic camera. Although this type of surgery is beneficial for the patient, it is also very involved for the surgeon.

Augmented reality could potentially help the surgeon by overlaying diagnostic information or navigation cues. One particular task that has to be solved to achieve this goal is the real-time 6D pose tracking (3D rotation + 3D translation) of all surgical tools in the endoscopic live stream.

In recent years, major breakthroughs in computer vision, including object pose estimation, were driven by techniques from machine learning. However, these methods require a large amount of training data which is not readily available in the medical domain. The goal of this project is the development of an interactive tool for the annotation of laparoscopic surgery footage.

The user should be able to accurately annotate the 3D rotation and 3D translation of all surgical tools visible in a random surgery frame. The annotation results should be stored, the frame removed from the stack of unannotated data, and a new frame should be presented. Since a large amount of data has to be processed, the annotation process must be very efficient in design.

Based on previous frames already annotated, the system should propose an initialization of the 6D poses in the current frame. For this purpose the tool should include an online learning component, e.g. a CNN that learns to predict correspondences between the image and 3D models of the surgical tool.

FOCUS OF WORK

- Survey related literature regarding data annotation and online learning.
- Create interaction concepts to enable a user to annotate the 6D pose of surgical tools very efficiently. The images are already annotated with segmentation masks and tool IDs. 3D models of the tools are also given.
- Implement an annotation tool based on the interaction concept.

Optional:

- Implement a component which can propose 6D poses of surgical tools as an initialization for the user. The user should be able to dismiss or refine the proposal.

TECHNISCHE UNIVERSITÄT DRESDEN

Factuly of Computer Science Department of Artificial Intelligence, Institute of Computer Vision

- The component should be trained based on data already annotated (in a previous session or another user) and be further refined through online learning during the annotation process.

Referee:	Dr. Eric Brachmann
Supervisor:	Dr. Dmitrij Schlesinger
Issued on:	30.02.2018
Due date for submission:	30.08.2018

Dr. Eric Brachmann
Chairman of the Audit Committee

Dr. Dmitrij Schlesinger
Supervising professor

Here is an abstract.

CONTENTS

1. INTRODUCTION

About three decades ago, the first minimally invasive surgery was performed. This was a huge step forward, as minimally invasive surgery offers several advantages compared to traditional surgery, such as less pain for the patient and less recovery time needed afterwards [1]. The surgeon conducts the operation only through a small hole in the otherwise closed abdominal wall. This makes it more involved than former procedures, of course. The executing surgeon inserts an endoscope into the patient and only sees the 2D images without any depth. Medical personnel could profit from computers assisting with augmented reality. Next to navigation cues and other vital information, such an assistance system could render the surgical tools into the endoscopic videos to compensate the missing depth to some extend and facilitate the operational process.

This task of automatically determining the location and rotation of 3D objects on an image is called 6D pose estimation. Robotics, augmented reality and medical imaging already apply 6D pose estimation, to name just a few examples. There exist various ways to recover the 6D poses on an image.

For well-textured clearly visible objects the task is considered solved. Methods like [2] by Lowe et al. rely on detecting sparse features, for example keypoints, which are matched against a database that contains the corresponding pose. Unfortunately, these approaches only work for objects with a strong and also visible texture. After cheap depth sensors like the Xbox Kinect became available, pose estimation procedures relying on depth were able to achieve good results on texture-less but unoccluded and non-deformed objects.

Many of the mentioned techniques have in common that they are not learning-based. This means that there is no learning procedure which learns an object's appearance. Brachman et al. on the other hand used random forests to achieve very good results for occluded objects [3]. The forests are trained to output the 3D location on the object for every pixel. Using the 2D-3D correspondences, a robust estimate of the object's pose can be computed. Krull et al. [4] combine the idea of learning to predict the 3D coordinates with the power of so called *Convolutional Neural Networks (CNNs)*. CNNs became popular around 2012, after training deeper networks turned out to be achievable in a feasible time on graphics cards. Deep CNNs offer outstanding accuracy that beat most traditional computer vision algorithms [5]. A drawback is their need for training data. Images have to be annotated with the desired network output beforehand. To obtain a versatile network that maintains a high accuracy in various situations a lot of training data has to be provided.

The goal of this work is to create a system that allows users to annotate large datasets, effectively and efficiently. We will introduce a tool that allows to annotate images with the 6D poses of arbitrary objects. To further reduce the time needed to annotate a whole dataset, we also present a neural network to support the user in the annotation process. The base architecture of the network is *ResNet* [6]. ResNet is a network presented in 2015, which allows construction of deeper networks and outperformed all previous architectures this way. The network is tailored to the characteristics of the dataset of the *Endoscopic Vision Challenge* [7], this means that it uses images and the corresponding segmentation masks to predict object coordinates. We also examine multiple configurations to obtain the best network. This symbiotic system of the annotation tool and the neural network can serve to create large datasets for other 6D pose estimation networks.

The remainder of the work is structured as follows: Chapter 2 explains the basic concepts

of deep learning and pose estimation. Chapter 3 introduces and discusses the latest research in the area of pose estimation, after giving a brief overview over earlier methods. We present the developed tool in Chapter 4. Chapter 5 describes the workflow of annotating images with the aid of the neural network. In Chapter 6, we explain the datasets used for the experiments and discuss the latter in depth. In Chapter 7 we discuss the results of this work and draw conclusions and give a prospect into future works in Chapter 8.

2. BACKGROUND

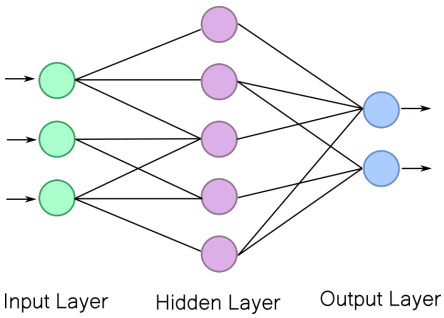


Figure 2.1.: Abstract structure of a feed-forward neural network. Most networks have more layers and more neurons per layer.

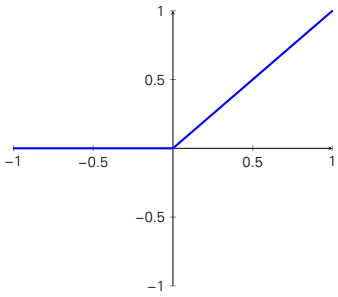


Figure 2.2.: Visualization of the ReLU activation function $f(x) = \max(0, x)$.

2.1. NEURAL NETWORKS

The following sections dissect the individual components of a neural network and explain concepts and procedures.

2.1.1. NEURON

A *Neuron* is the smallest atomic unit of a neural network. The layers of a network consist of neurons. A neuron computes the linear function

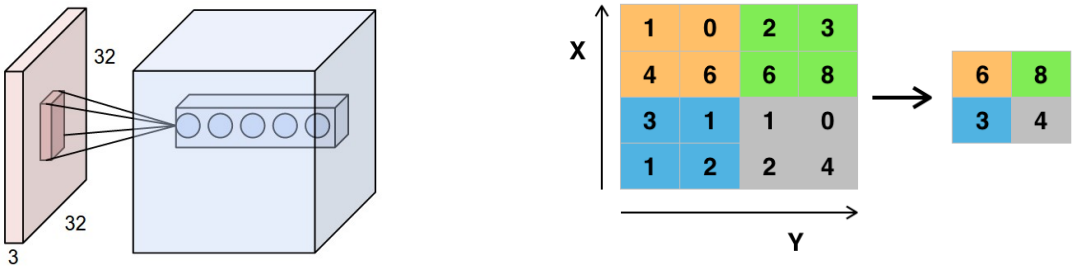
$$y = \sum_{i=0}^{k} w_i x_i + b \tag{2.1}$$

where w_i is the weight for the i-th input x_i and b is a bias. The weights and the bias are the values that can be learned during the training process of the network (see Section 2.1.4). The output of a layer of the network is the vector $(y_0, \cdots, y_t)$, y_j being the output of the j-th neuron. Those outputs then serve as inputs to the next layer, i.e. y_i becomes x_i. Not every output has to be used by every next neuron. To prevent the network from collapsing into a single linear classifier and thus to increase the expressivity of the network, the output of a neuron is put into a non-linear so-called activation function. Although many different activation functions exist, most popular is the *ReLU*, which was first presented by Jarrett et al. in [8]. The function is exemplarily visualized in Fig. 2.2 and is calculated as

$$f(x) = \max(0, x) \tag{2.2}$$

2.1.2. FEED-FORWARD NEURAL NETWORKS

A *Feed-Forward Neural Network* is a network consisting of layers of neurons. The first layer is called the input layer. Those are the neurons that receive the data that the network is supposed to process. The last layer is called the output layer. The form of the output of the neural network depends on the field of application. It can be object coordinates like in our case, class



(a) Example convolutional layer. The input image has dimensions 32x32 and 3 channels. The blue volume is the actual convolutional layer with multiple filters. Image from [9].

(b) Example pooling layer with maximum as pooling operation. Image from [10].

Figure 2.3.: The two main layer types of a convolutional neural network.

probabilities in classification tasks or any other real-valued output. The intermediate layers are called hidden layers. Their number can grow to over a hundred in modern networks [6], thus the name deep neural networks. All layers can have different numbers of neurons and connections to previous layers. A layer, in which each neuron is connected to every output of the previous layer using individual weights for each connection, is called a *Fully-Connected Layer*. A schematic overview over the structure of a neural network is visualized in Fig. 2.1. The green circles on the left are the neurons of the input layer. The purple neurons in the middle belong to intermediate (hidden) layers. The output layer consists of the blue neurons on the right. A neural network can generally be seen as the function $y = f(x, \theta)$, where θ are the weights and biases of the neurons and y is the output of the network.

2.1.3. CONVOLUTIONAL NEURAL NETWORKS

Depending on its depth and number of neurons, a densely connected feed-forward network with individual weights for each neuron needs a lot of memory and computation time to perform operations. A *Convolutional Neural Network (CNN)* reduces this need for memory and increases computational throughput by sharing the weights of the neurons. This type of network is often employed in computer vision tasks, like image classification and segmentation. A CNN introduces two new types of layers: *Convolutional Layers* and *Pooling Layers*. A convolution layer convolves the input using a kernel. This means that the kernel is moved along the input's dimensions and sums up the respective values multiplied by the kernel weights. For a position (i, j) in the input I and a kernel matrix K, this can be written as

$$O(i,j) = \sum_{w_0, w_1} I(i + w_0, j + w_1) K(i + w_0, j + w_1) \tag{2.3}$$

where O is the output. The ranges of w_0 and w_1 depend on the size of the convolution window. The kernel can be learned but stays the same for the whole convolution. Fig. 2.3a shows an abstraction of a convolutional layer. The red plane on the left is the input image, the blue cube on the right is the actual layer. The blue cube is broader than the red input because

it has multiple filters. Each filter consists of an independent kernel. This means that this layer has multiple outputs, each being a convolution of the input with a different kernel.

A pooling layer can be used to reduce the size of the data between convolution layers. This significantly speeds up computation and reduces memory consumption. It also reduces the number of parameters which helps to prevent the network from overfitting. An overfitted network performs well on the data it was trained on but does not generalize well, i.e. has a significantly lower accuracy on unseen data. A pooling layer alters equation 2.3 in the following way:

$$O(i, j) = \max_{w_0, w_1} I(i + w_0, j + w_1) \tag{2.4}$$

The size of the output of the pooling layer is reduced by a factor depending on the size of the pooling window.

2.1.4. NETWORK TRAINING

A network's weights and biases can be initialized to 0 or sampled randomly from a Gaussian distribution. To set the parameters to meaningful values that produce the desired output, we need to train the network. The following paragraphs describe techniques to train a network.

ERROR-BACK PROPAGATION

The predominant procedure to train a network is called *Error-Back-Propagation* or *Backpropagation*. To employ backpropagation, a differentiable but arbitrary loss function has to be defined. The loss function measures the error of the output of the network compared to the desired output, e.g. by summing up the squared differences of the output and the objective output. The term *loss* and *error* or *error rate* stand for the same thing. First, the network is applied to a training example in the so-called forward pass. Next, the loss function is derived with respect to each weight of the network. The derivative, the computed and the desired output together yield the delta to be applied to the weights. This delta is then also propagated backwards through the net and the deltas of the layers in between are calculated using the chain rule of calculus. After the deltas for all neurons have been computed, the weights are updated according to an update rule. We explain some of these rules in the next paragraph.

OPTIMIZATION

There exist many different patterns how to update the network weights after obtaining the deltas. The *Stochastic Gradient Descent (SGD)* multiplies the delta by a learning rate and subtracts it from the weight. To ensure convergence, the learning rate is often decreased over time. A more elaborate method is the *Adaptive Moment Optimization (Adam)* [11] which computes adaptive learning rates for each parameter. The algorithm computes and stores an exponentially decaying average over past gradients as well as an exponentially decaying average over the past squared gradients. This serves to increase the learning rate in case of small gradients

(a) The object coordinate representation of a human. Image from [14].

(b) An example object from the T-Less dataset. Image from [15].

(c) The corresponding rendered 3D object coordinates (scaled for visualization).

Figure 2.4.: Example 3D coordinate representations.

and decrease the learning rate in case of large gradients. Both averages are stored for the next iteration. There exist other optimization procedures that are not covered here.

REGULARIZATION

Regularization can mean various things for neural networks. In general, its purpose is to keep the network from overfitting. Ian Goodfellow defined it in [12] as any modification that reduces the generalization error but not the training error. Many network architectures regularize the network by adding a penalty term on the weights. Another possibility is to randomly deactivate a subset of the neurons to force the network to adapt to this loss of information. This is called *Dropout*. *Batch Normalization* helps the network generalize by subtracting the batch mean and batch standard deviation from the output of the previous layer. It was first introduced in [13]. A batch is a subset of the input data. This technique typically speeds up network training and allows running the network on a machine that can't fit the whole dataset in its memory. To keep SGD or other optimizers from reversing the covariance shift performed by batch normalization, the two parameters *beta* and *gamma* of such a layer are usually trainable. This way, a layer of neurons cannot fully counterpoise the shift.

2.2. 6D POSE ESTIMATION

6D pose estimation is a central task in the computer vision community. The goal is to retrieve the translation and rotation of an object relative to the camera, i.e. the 6D pose. 6D refers to the *6 Degrees of Freedom (6-DoF)*, i.e. the 6 free parameters of the 3D translation and 3D rotation. A *6D pose P* is the tuple (R, t), where R is the 3×3 rotation matrix and $t \in \mathbb{R}^3$ the translation vector which defines the position of the object. The field of application ranges from medical imaging, robotics, augmented reality and many more.

There are different possible ways to estimate the pose with learning-based approaches. The 6 parameters can be predicted directly or an intermediate representation can be used for pose computation. Predicting the parameters offers no possibility to verify or improve the pose afterwards. A possible intermediate representation are object coordinates. This means that the 3D locations on the object are predicted for the 2D pixels in the input image. Each subset

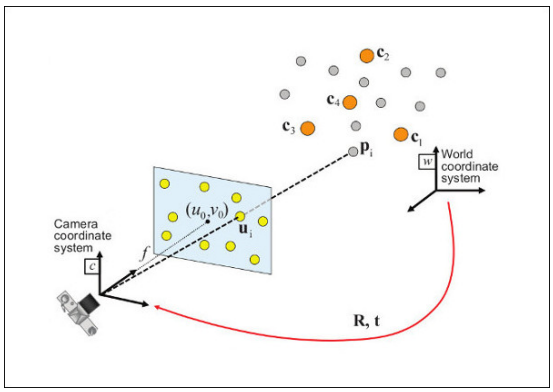


Figure 2.5.: The relationship between the camera, the 3D points and their projections on the screen defined by the rotation matrix R and the translation vector t. Image from [16].

of object coordinates leads to a certain pose. The next section will introduce this concept of object coordinates in depth.

2.2.1. OBJECT COORDINATE REGRESSION

Tayler et al. first used object coordinates in [14]. Instead of directly predicting the location of joints or limbs of the human body, they computed the corresponding 3D location on the person for each pixel (see Fig. 2.4a). This idea can be transfered to objects of any kind. Fig. 2.4b shows an example object from the T-Less dataset [15] and Fig. 2.4c shows the corresponding rendered 3D object coordinates.

The 2D-3D correspondences between object coordinates and pixels yield the *Perspective-n-Point (PnP) Problem*. For a given 2D location u in the image and the corresponding 3D point p, a pose consisting of the rotation matrix R and the translation vector t has to fulfill the equation

$$u = K\,[\,R \mid t\,]\,p \tag{2.5}$$

where K is the camera matrix

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

which projects the 3D point transformed into the camera coordinate system on the 2D image plane. The points u and p are both in homogeneous coordinates, i.e. $u = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ and $p = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$. x and y correspond to the column and row in the image respectively. The skew factor s is usually 0. The principal point (c_x, c_y) normally is the center of the image and f_x and f_y are the focal lengths in x and y direction, respectively. The values for those variables can vary, for example

when cropping the image. Fig. 2.5 visualizes the relation of pixels and object coordinates. If equation 2.5 is overdetermined because there are more correspondences than variables, it cannot be solved directly.

A method to solve the PnP problem for many correspondences is to use the *Random Sample Consensus (RANSAC) algorithm* [17]. RANSAC selects a model based on a subset of the dataset and evaluates it using an energy function. This way, the approximate best model is found iteratively. Algorithm 1 shows the outline of the RANSAC algorithm for pose estimation. The energy function can be arbitrary but should capture the quality of the pose. For example, the number of inliers can be counted. An inlier is a 3D point whose reprojection error is within a certain threshold.

Algorithm 1 RANSAC

Require: Set of 3D points
Require: Set of corresponding 2D points
Require: Number of iterations i
Require: An energy function to score the pose hypothesis E
  Set current energy $e = 0$
  Set current best pose hypothesis $H* = null$
  for $1...i$ **do**
    Select a subset of corresponding 2D and 3D points to compute pose H
    Compute energy e' of pose H: $e' = E(H)$
    if $e' > e$ **then**
      $H* = H$
      $e = e'$
    end if
  end for
  return Best pose $H*$

3. RELATED WORK

In the following chapter, we investigate the current state of research on 6D pose estimation. It starts with an excerpt of non-learning-based methods. We then present more recent, learning-based works with emphasis on active learning and fine-tuning of neural networks.

3.1. RESEARCH ON 6D POSE ESTIMATION

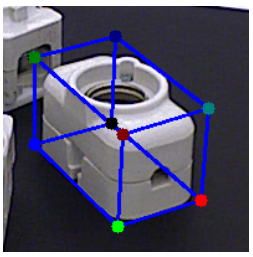
Pose estimation has become an increasingly interesting problem with the advent of robots in production and the area of virtual and augmented reality [18]. Early research focused on manual feature extraction, divided into the major groups of sparse, dense and template-based approaches. Nowadays scientists often employ learning-based techniques which offer higher accuracy.

3.1.1. NON-LEARNING-BASED

Before the major breakthrough of deep learning in 2012 [19], handcrafted feature-based approaches were common for 6D pose estimation [5]. A lot of research matches sparse characteristics detected in the image against a database that contains the related pose information. The works of [2, 20] use keypoints as the discriminative feature and offer good accuracy on well-textured objects. Unfortunately, precision declines for poor-textured or texture-less objects, because in those cases detectors are unable to find stable keypoints or any at all. The methods presented in [21, 22, 23] identify edges in the image and use different techniques to obtain an initial pose guess based on those edges. The resulting pose is retrieved iteratively refining the previous guess, by minimizing the distance of the projected and detected edges.

Template-based methods [24, 25, 26, 27] use generated views of discrete viewopints on the object at different angles which are matched against the given image to retrieve the pose. This approach relies mainly on the shape of objects and thus works well for poor-textured or textureless objects [28]. To cover many objects and a large pose range, the number of templates of an object has to be increased though, which slows down prediction. Hashing the views to speed up the pose estimator reduces accuracy [29]. Additionally, deformed or occluded objects decrease precision, as templates globally reason about the object's pose.

Some authors draw on multiple views to improve accuracy. Stereo cameras are used in [30]. A second image from another angle implies depth information to a certain extend but involes the additional task of stereo matching. The availability of cheap depth sensors, like the Kinect, gave rise to algorithms making use of RGB-D images. Multiple approaches are possible when incorporating depth to retrieve an object's pose. Voting schemes were employed in [31, 32] and achieved good precision. First, a point in the image, potentially on the surface of the object, is selected and paired with all other scene points. This so-called point pair feature then votes for a possible pose, if the combination of their distance and respective normals are contained in the global sparse model description. The pose with the most votes is deemed to be the optimal one.



(a) The corners of the object's bounding box.

(b) The segmentation preceding the pose estimation.

Figure 3.1.: Example images displaying the functioning of BB8. Images from [18].

3.1.2. LEARNING-BASED

Methods that learn information about objects - that is not explicitly modeled - started to outperform the previous techniques. There exist a vast literature for this task but we restrict our review to recent works that are most relevant to our approach.

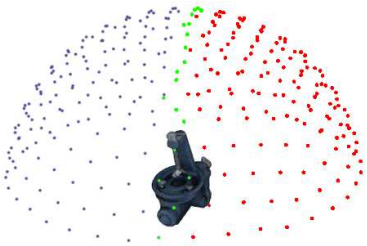
Lai et al. use decision tress in [33], which incorporate the semantic information of objects and their poses. Tomè et al. predict human poses from single RGB images in an end-to-end manner [34]. The system developed in [35] relies on multi-view images and depth information to retrieve an object's pose in a cluttered and occluded environment and scored the 3rd and 4th place in the Amazon Picking Challenge 2016 [36]. A prominent example of camera localization, which represents the problem of estimating the camera position in 3D space relative to the scene, is PoseNet [37]. The underlying architecture of the employed neural network is based on GoogleNet, which was introduced in [38]. The CNN estimates the camera position directly without regression from a single image and is fast. In some indoor cases, PoseNet has an error as high as 50 centimeters, but our application demands high accuracy.

BB8

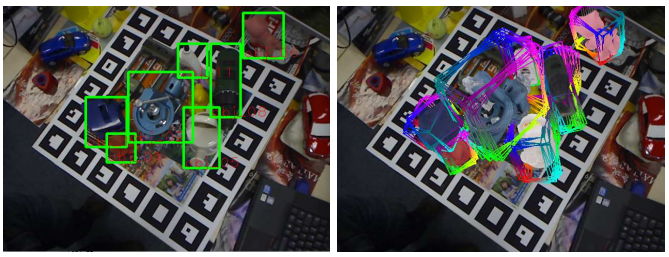
The recently developed BB8 [18], which is an abbreviation for the 8 corners of the bounding-boxes of objects, is the result of the work of Rad and Lepetit that operates on RGB-only images. Instead of regressing the pose of an object with object coordinates (see Section 2.2.1) like [3], they let a deep neural network estimate the object segmentations first (see Fig. 3.1b) and then predict the 2D locations of the 3D corners of the object's bounding box. An example of the corners can be seen in Fig. 3.1a.

Similar to [3], the object's position is not predicted directly either, but instead regressed by solving the perspective-n-point problem (PnP) of the correspondences of the corners of the object's bounding box and the projected locations of the corners in the image. The architecture of the first and second net is based on VGG [39] respectively but with the last layer cut off and replaced by a fully connected layer which is fine-tuned.

The first neural network, which segments the image, helps estimating the pose of the object in a way that the second network positions its window at the center of the segmentation and estimates the 2D locations of the 3D corners of the object's bounding box. The network reasons globally about the object by not moving the window during training and prediction. The authors argue that patch-based pose estimators are typically very noisy, and hence require a robust optimization scheme, like RANSAC.



(a) An exampe of a discrete distribution of viewpoints on an object.



(b) Left image: An example for bounding boxes found by the SSD-6D network. Right image: The most confident views and rotations for those boxes.

Figure 3.2.: Example images displaying the functioning of SSD-6D. Images from [40]

Unfortunately, BB8's take on pose estimation fails on symmetric objects, when implemented directly as described above. To address this problem, the authors first estimate the rotational angle of the object using a neural net, and mirror the image if necessary. This way, a CNN can be trained only on a certain range of the angle of the pose.

The proposed method offers good performance and can compete with and partly surpasses state-of-the-art research. Yet, object coordinates provide high accuracy too. Furthermore, we assume that the often merely partial visibility of the surgical instruments calls for a non-global reasoning design. Thus, BB8 is not followed any further in this work.

SSD-6D

The system presented in [40] is based on the Single-Shot Multibox Detector (SSD) [41]. Their take on 6D pose estimation is especially alluring, as the authors train the network exclusively on synthetic data. A positive outcome would imply that the lack of already annotated datasets for 6D pose estimation could be overcome by generating data for network training.

The SSD-inspired network architecture produces feature maps of the input images, that are convolved to predict the class and 2D bounding box (see Fig. 3.2b left) of objects in the scene. The network also estimates the probability of a discrete viewpoint on the object. Those viewpoints are sampled equidistantly alongside a predefined step size (see Fig. 3.2a). The pose is calculated taking into account the class, bounding box, viewpoint and a guess on the in-plane rotation of the object. The network is trained entirely on images from the MS COCO dataset [42]. The objects that are to be detected are rendered into them with arbitrary translations and rotations. For each transformation, the network is given the closest discrete viewpoint, in-plane rotation and the tightest bounding box as a regression target.

The author's reasoning, that their approach on pose estimation is a more natural one than pose regression. This stands to reason, as a human learns what an object looks like by viewing it from different angles. Nevertheless, the network produces rather inaccurate initial results. To remedy this drawback, an optimization scheme extracts 3D contour points from the rendered hypothesis and minimizes the distance to the detected 2 locations of the points on the image.

The neural network alone performs less well compared to [3] or [18]. The optimization process could also be applied to the two mentioned works. Hence it is not investigated further.

KURMANN ET AL.

The work of [43] estimates poses of surgical instruments in minimally invasive surgery. Similarly to object coordinate regression, Kurmann et al. develop a system that produces probabilities of the presence of an object but not in a dense way but instead only for the joints of the tools. Their design draws on a scene model which holds how many tools can be visible at most, what tools are currently visible and what parts of those tools.

The authors of that paper argue that the common two-stage pipeline for 6D pose estimation, that consists of object detection and then pose estimation, results in a more complicated design, and criticize that a sliding window approach might miss very small or very large instruments, i.e. advocate a global reasoning design.

The architecture of the employed network is based on the U-Net developed by the authors of [44], and trained by optimizing the cross-entropy derived from the scene model described earlier. The network architecture of U-Net is extended by a fully connected layer that is trained to predict the probabilities of the instruments.

The results of the design look promising and run time per image is around 100*ms*, enabling it to be deployed as a real-time solution. Conversely, mainly literature of the biomedical imaging was considered and compared, disregarding research work in other areas on pose estimation.

3.1.3. LEARNING-BASED: OBJECT COORDINATE REGRESSION

There exists interesting work achieving high accuracy by using object coordinate regresseion. The idea is based on [14] and [45]. The first used it to estimate the pose of a human body, the latter to regress the cameras position in a scene retrieved from a single RGB-D image.

Brachmann et al. achieved record-breaking results in [3] for texture-less objects and good performance in general. The authors based their work on forests instead of trees to regress the object's pose. The forests are trained to jointly predict the probability of object instances as well as the object coordinate probability for a given pixel. The output of the forest is processed in an energy function that imposes an energy minimization problem to regress the object's pose. A RANSAC-like scheme then iteratively refines the pose.

In [46], Brachmann et al. adjusted their pipeline from [3] to work with RGB-only images. To reduce uncertainty in the object instance and coordinate predictions they incorporate an auto-context framework and marginalize the object coordinates over the depth information to cope with the missing fourth channel. The presented system outperforms Brachmann et al. 's previous work but is still based on forests.

Krull et al. elaborated [3] by replacing the energy function by a CNN [4]. They were able to further improve the performance and transfer object coordinate regression to modern deep neural networks. The system called PoseAgent fuses regression forests and CNNs [47]. The regression forest outputs pose hypotheses that the CNN then refines. Krull et al. are able to achieve state-of-the-art results and improve resource utilization. Although [48] finds, that random forests partly offer a slightly superior performance, neural networks can compete and are therefore the focus of this work.

PERTSCH

Although [28] also works with RGB-D images, we present his work here as the author proposes an easy extension to adapt the entire process to RGB-only images and presents promising results. The work by Pertsch is based on [3], i.e. uses object coordinates (see section 2.2.1), but replaces the random forest with a CNN. The developed pose estimation pipeline relies on three steps. The first one segments the image, the second regresses the object coordinates, and the last one evaluates pose hypotheses.

We don't to go into detail into the first operation of the pipeline as our training and test data already includes segmentations and we can hence omit it completely. The second stage consists of a multilayer CNN-architecture to predict the object coordinates. The architecture consists of an encoder-decoder multilayer network. Inspired by [44], the author assimilates skip-layer connections. The network computes object coordinates for each pixel of the previously computed segmentation. Pertsch employs a RANSAC-scheme to improve the quality of the predicted final pose. Two different methods to retrieve the pose are presented in the work, one relying on the energy function introduced in [3] but in an altered version, and one procedure solely drawing on RGB information without the additional depth of a sensor. The latter, which is more relevant for us as we do not have any depth readings, is based on the earlier presented [46].

Instead of penalizing depth divergences of the rendered depth from the estimated pose and the sensor readings, the number of pixels whose reprojection error is greater than a certain threshold is minimized iteratively by the RANSAC algorithm. This allows for the RGB-only extension that the author suggests but does not elaborate in detail. Without depth the pose regression becomse a PnP problem, that can be solved by a PnP solver which takes at four 3D-2D point correspondences as input. We pursue a similar approach in our work which differs in architecture of the network, the steps of the pipeline and the input data. But the general ideas of [28] and especially [3] are adopted and further complemented with current research and active and incremental learning.

3.2. RESEARCH ON ACTIVE AND ONLINE LEARNING

Online learning considers how to incorporate new available data into a trained model or neural network. Active learning is the field of selecting data that a human should annotate manually, because the model does not perform well on it. The literature survey [49] gives an overview over methods developed before 2011. Wang and Shang, the authors of [50], might be among the first to incorporate active learning into deep learning though, according to [29]. In [51], Al Rahhal et al. apply a similar idea to hyperspectral image classification, a task that shares the foibles to be tedious and time-consuming with biomedical image annotation. The authors developed an active selection paradigm to electrocardiogram classification.

ZHOU ET AL.

In [29], Zhou et al. describe a novel process of actively demanding data to be annotated to improve the network quality and apply the newly available data in an incremental manner. They focus on this area because annotating biomedical images is still a time-consuming task that

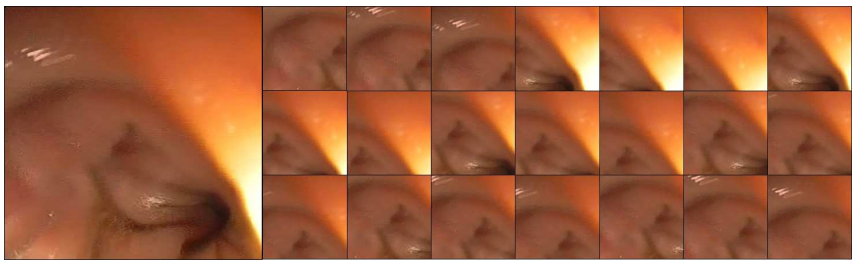


Figure 3.3.: A candidate (left) and the patches generated sharing the same label. Images from [29].

requires a lot of expertise and skill.

Opposed to retraining from scratch, the network is fine-tuned by an incremental tuning algorithm. According to the authors, researchers have shown that this offers superior performance. In contrast to our task, the authors want to achieve improvements on image classification and frame detection, but work on biomedical images nonetheless.

Computer-aided-diagnosis (CAD) systems usually provide a candidate generator, which can quickly produce candidates, including true and false positives, to train with. Through data augmentation the learner can be made more robust to unforeseen situations. For this, numerous patches sharing the same label are generated from the candidate, as can be seen in Fig. 3.3, an presented to the classifier.

The active selection process of candidates requires a measure of the worthiness of a candidate. To achieve this, the entropy and diversity of patches are calculated using the network's predictions for the patches of a candidate. Entropy is the negative log-likelihood of the network's prediction, whereas diversity captures how much patches of a candidate contradict, as they should all share the same label. Candidates with contradicting patches or low entropy can be selected for manual annotation.

The procedure quickly improves the neural network's accuracy. Learning from scratch or random selection of next candidates are quickly outperformed, as only around 20% of manually annotated candidates are necessary to reach the same error rate with the presented procedure.

In our case, data augmentation is possible, as we can render synthetic images, but the unnatural lightning and shadowing might decrease the accuracy of the design. The question of how to find a worthiness measure arises too, as we can't directly tell how sure a network is when predicting a pose. But it provides a good direction of how to approach active learning.

LIU & FERRARI.

In [52], Liu and Ferrari describe new strategy for active learning for human pose estimation, a time-consuming task to produce groundtruth annotations for. Their key contributions consist of an uncertainty estimator for the joint predictions produced by a convolutional pose machine (CPM), and an annotation interface that reduces the time needed by a human annotator to click joints. The predictions by the CPM are heatmaps, with high temperature resembling the most probable position of the joints (the input image can be seen Fig. 3.4(a2), the heatmap in 3.4(b2)).

The active learning scheme incorporates influence and uncertainty cues. Influence cues consider images that are similar to other unlabeled images could propagate information. The uncertainty cues are measured by the uncertainty estimator. The estimator focuses on uncertain

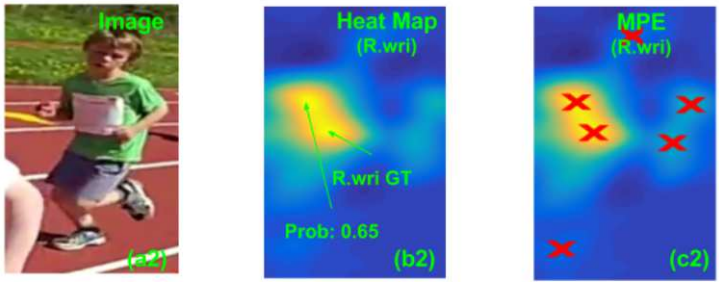


Figure 3.4.: An image of a human body, the corresponding heatmap as well as the result of the Multiple Peak Entropy (MPE). Images from [52].

predictions, i.e. predictions with multiple weak peaks (multiple peak entropy) (see Fig. 3.4 (c2)). The final selection process then takes both cues into account when requesting an image to be manually annotated.

In addition, an interface is proposed that reduces the annotation time significantly. The system predicts a pose and segments the image around joints. The user can then right click anywhere in the segmentation to accept the estimated joint location or manually select it.

The results of [52] look auspicious, as a reduction in annotation time can be achieved through the interface and the selection process. Regrettably, it cannot be directly applied to our problem, as we need a problem-specific certainty estimator. But the idea of requesting similar images to be annotated might yield a performance gain, and similarly to the presented interface we present the user with an initial probable pose, too, to make only small corrections necessary.

4. MANUAL ANNOTATION

Figure 4.1.: The logo of the pose annotation tool 6D-PAT.

The following chapter analyzes the manual 6D pose annotation process and its prerequisites. To this end, we define the necessary terminology and explain the workflow of recovering poses from images using the tool we developed.

4.1. TERMINOLOGY

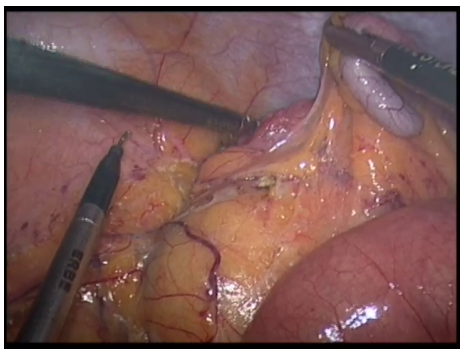
Image. An image I is a 2D matrix of pixels. A pixel is a 3D dimensional vector containing the RGB colors, which range from 0 to 255.

Object Model. An *object model* (or *3D model*) O is composed of a set of points $M \subseteq \mathbb{R}^3$, which are often called *vertices*, and a set of triangles $T \subseteq M^3$, also called a mesh. The type of object is not restricted. In the T-Less dataset [15], the objects are mostly screws and other hardware.

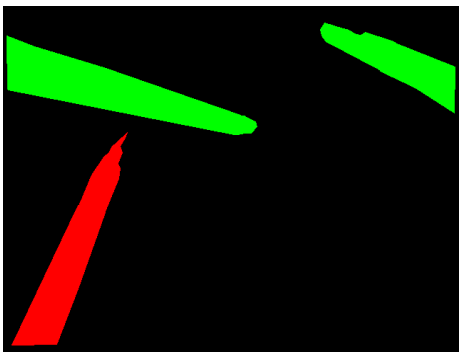
Correspondence. A *correspondence* C is the tuple (u, p), which captures the relation between a pixel u of an image I and a 3D point p on the surface of an object model O. The pixel u is the projection of p onto the image plane using the camera matrix K and a pose P. A pose can be recovered computationally if at least three correspondences are known (see Section 2.2.1).

Segmentation Mask. A *segmentation mask* (or *segmentation image*) S for an image I is a second image of the same size. Each position (i, j) of the mask encodes the class of the pixel at (i, j) in I. The set of classes can be defined arbitrarily. In the context of this work, each class represents a type of object model. The segmentation mask can be seen as the mapping $s(i, j) = q_k$ for a class set $Q = \{q_0, \cdots, q_n\}$. We say, a pixel at position (i, j) in an image I belongs to an object model O of type q iff $s(i, j) = q$ in the corresponding segmentation image S.

Ground-Truth. A *ground-truth pose* $\bar{P}$ is a 6D pose, which is always recovered by a human instead of a machine. The ground-truth pose is the best approximation of the rotation and translation of an object model O visible in an image I. It is an approximation because there can be a discrepancy between the real world object and its digital 3D representation. Image conditions like lightning, motion blur, etc. might make it unfeasible to recover the perfect pose. Camera distortions and other influences in the photographs of the objects might also not be modeled correctly or not accounted for at all. But it must apply that the translation and rotation error of a ground-truth pose $\bar{P}$ are within a certain threshold.



(a) An example image from the Endoscopic Vision Challenge dataset. Image from [7].



(b) The corresponding segmentation mask. The colors encode the tools' classes. Image from [7].

Figure 4.2.: An example image and its corresponding segmentation mask from the Endoscopic Vision Challenge dataset.

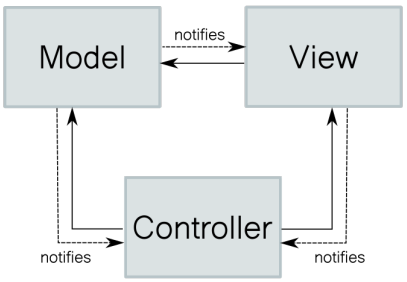
Depth Image. A *depth image* is an image D belonging to an image I of the same size that contains the distance d between the camera and the surface for each pixel u in I. Depth images can be obtained from stereo images or using special cameras and are often used in pose estimation and other computer vision tasks. A depth image can also be denoted by *RGB-D*.

4.2. IMAGES OF THE ENDOSCOPIC VISION CHALLENGE

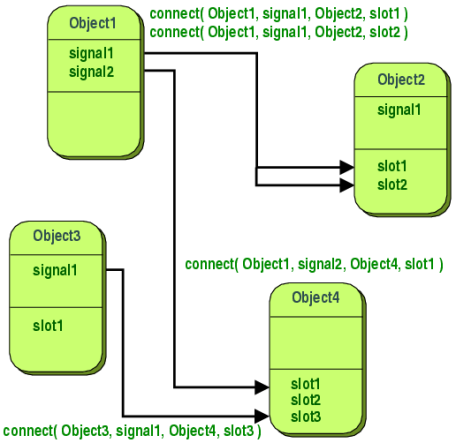
The goal of this work is to provide a system to successfully and efficiently annotate the images of the *Endoscopic Vision Challenge* [7]. The dataset includes segmentation masks but neither object models nor pose annotations or depth images. An example image together with the corresponding segmentation mask are given in Fig. 4.2. Occlusion and artifacts like motion blur can occur in the images. The issues with this dataset and the obstacles preventing its complete annotation are discussed Section ??.

4.3. 6D POSE ANNOTATION TOOL (6D-PAT)

The creation of sufficient training data for neural networks can be a time-consuming and tedious process. Using non-specialized tools designed for other purposes, like 3D modeling or CAD programs, require the person creating the annotations to get accustomed to complex user interfaces (User Interfaces (UIs)). The goal of the annotation tool is to provide a system that allows easy and efficient annotation of images - images of the Endoscopic Vision Challenge in particular. The ground-truth poses recovered using the program can then be used to train a neural network. The program is written mainly in the language C++ and is dubbed *6D Pose Annotation Tool (6D-PAT)*. Its logo can be seen in Fig. 4.1. The next sections present the carved out requirements, details on the used frameworks, the program's architecture, as well as the designed pose recovery process.



(a) The Model-View-Controller architecture. The solid lines stand for a direct connection either because the target is owned or known by reference. The dashed line is an indirect connection realized using the observer pattern or the Qt Signals and Slots mechanism.



(b) The Signals and Slots mechanism of Qt. A class can define signals which can be emitted. Slots are functions that can be connected to signals. When a signal is emitted, all connected slots will be called. Image from [54].

Figure 4.3.: Two basic architectural concepts used in 6D-PAT: the model view controller pattern and the signal and slots pattern.

4.3.1. REQUIREMENTS

Fulfilling the goal of providing a tool for annotating large datasets implies some requirements. Datasets can consist of thousands of images and many object models. To guarantee a fluid workflow, we need to provide the user with a browsable overview over the dataset. This prevents a disruptive annotation process, as the user doesn't need to open the next image or a different object model individually after recovering a pose. The program has to offer the possibility to select and view images and object models. The most essential part of such an annotation tool is the incorporation of functionality to recover poses and to edit misaligned poses. Different manual pose recovery mechanisms are possible. Due to the limited time frame of this work, we implemented only one method, which we expected to be the most efficient one. The annotation process is described in detail in Section 4.3.4.

4.3.2. FRAMEWORKS & THIRD-PARTY LIBRARIES

All necessary dependencies of the tool are listed below. The user needs to compile or install the dependencies before using the annotation tool. But since all frameworks and libraries are platform-independent, the program can be compiled and run on different systems.

Qt. *Qt* [53] is a powerful framework for C++ that offers a vast selection of user interface components but also general functionality that exceeds the capabilities of the standard C++ library. Qt was also chosen as the main framework because it ensures portability of C++ applications by encapsulating system calls of all kind.

OpenGL. *OpenGL* [55] is a widespread open-source 3D graphics library specification. Implementations of the specification exist for many different operating systems, which makes appli-

cations using OpenGL portable.

OpenCV. *OpenCV* [56] is a C++ library created for various computer vision tasks. OpenCV provides functions for object tracking, object detection, image segmentation and many more. We use its *solvePnPRansac* method in this work.

Assimp. *Assimp* [57] is a C++ library designed to import 3D models. The library was incorporated into the tool to ensure a broad support of 3D model formats.

4.3.3. ARCHITECTURE & CODE DESIGN

6D-PAT is primarily a UI program, i.e. its purpose is to visually present data and enable easy interaction for the user. Thus, we chose the *Model-View-Controller (MVC)* pattern as the underlying architecture. MVC separates the concerns of data management (Model), displaying data (View) and high level logic (Controller). The schematic of the MVC architecture is shown in Fig. 4.3a. To ensure extensibility and exchangeability, the classes of the model do not know view or controller classes directly. The model only informs its observers (views that display data, for example) through indirect connections. The indirect connections are realized via Qt's signals and slots mechanism, which is visualized in Fig. 4.3b. The view classes only display data and handle simple user interactions that do not involve complex logic. The latter is the responsibility of the controller classes. Classes do not know implementation details of classes of other groups. To speed up interface creation, we used *Qt Designer* to layout the views. Qt Designer is a graphical development tool that allows placement of UI components and linking of signals and slots without directly writing code. It is part of the standard Qt framework.

The most important classes of the program are displayed in Fig. 4.4. The diagram is simplified for easier understanding. The large rectangles group the classes by their affiliation in the MVC pattern. The bold arrows between the groups denote which classes can use or know which other classes. This does not necessarily mean, that all classes of the source rectangle use all classes of the target rectangle.

CONTROLLER CLASSES

The controller classes mostly consist of the *Main Controller*, which owns the *Pose Recoverer* and the *Neural Network Manager*. The code of these classes has not been incorporated into the main controller to further separate the classes by concern and facilitate future modifications. The pose recoverer handles the clicked 2D-3D correspondences and offers functionality to compute the pose. The neural network manager handles all network related tasks. To keep the UI responsive during time-consuming network operations, the network manager uses the *Neural Network Runnable* to run tasks in a separate thread. This design allows easy substitution of how the network is run, if necessary. The main controller also creates the *Model Manager* and the *Load And Store Strategy*. The *Main Window* is created by the main controller, as well, and receives the model manager from it.

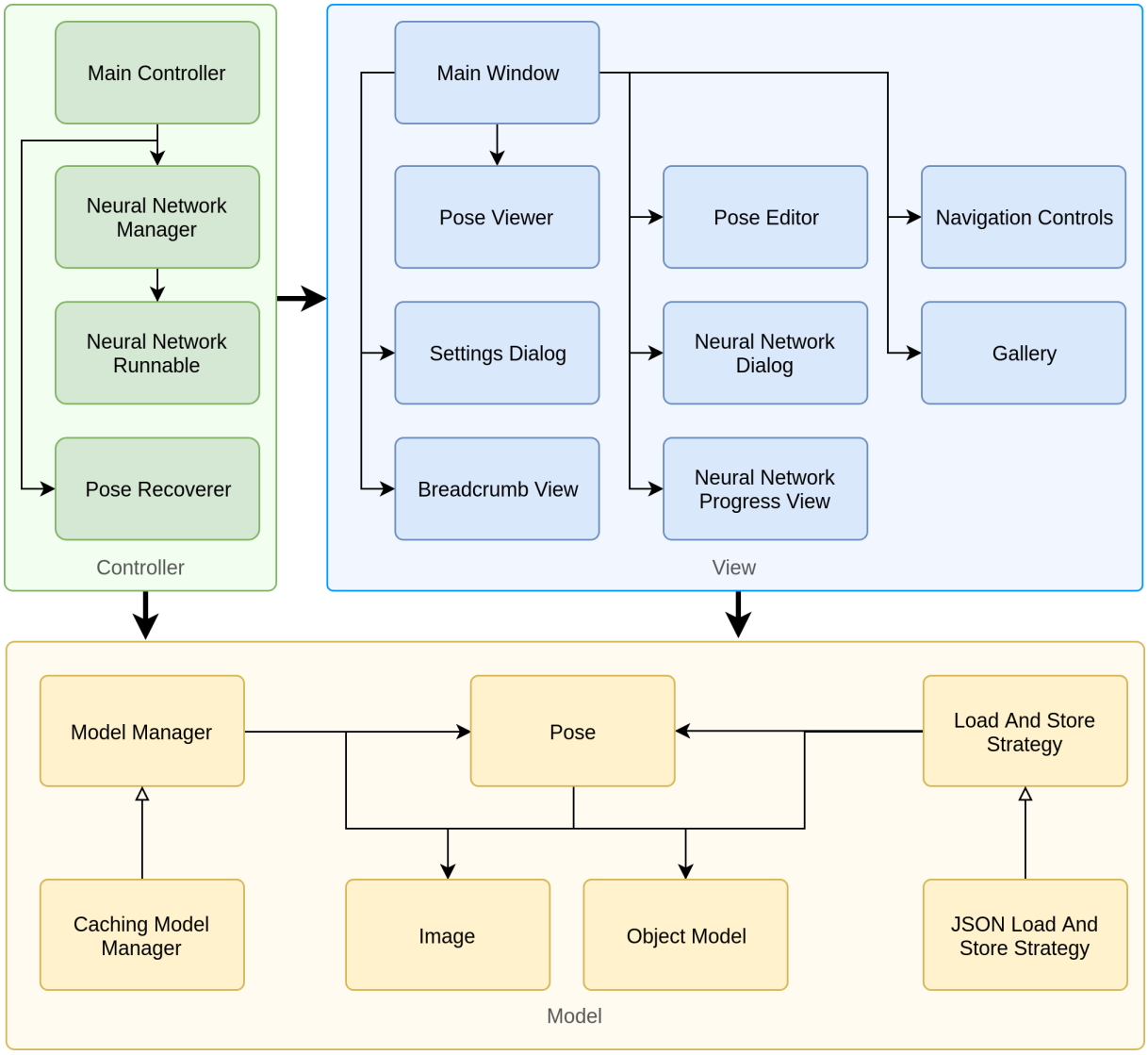


Figure 4.4.: An abstract high-level class diagram of a subset of the classes of 6D-PAT. The large rectangles show the affiliation of a contained class in the MVC pattern. Arrows between those rectangles imply which class can use which target class, although not all classes of the source rectangle necessarily use all classes of the target rectangle. Small filled-out arrows imply a use relationship, while the not filled-out arrow heads indicate inheritance.

VIEW CLASSES

The main window holds all views of the UI. It is the main controller's counterpart on the view side and delegates all alterations requested by the controller to the other views. All signal and slots that are necessary between view classes get connected by the main window, if not defined in the Qt Designer. Most of the signals and slots communication takes place between the *Pose Viewer*, the *Pose Editor* and the galleries. Whenever the user clicks an image or an object model in the gallery, the gallery notifies the viewer and editor to display the respective entity. The view classes know the model classes, especially the model manager, by reference. They receive the reference from the main window. The pose viewer and editor communicate to show the alterations made to a pose before saving the changes. Simple operations, like saving an edited pose, are often directly processed by the view classes to reduce the amount of signals and slots.

MODEL CLASSES

The model manager and the load and store strategy are the key interfaces to access data. The actual code of managing the data was separated into the *Caching Model Manager*, which caches the list of images, object models and poses, and the *JSON Load And Store Strategy*, which uses JSON files to store the poses. It expects the camera matrices file of the images to be in JSON format, as well. Both classes can be replaced to support other storing techniques, like databases, in the future. The classes that contain the actual data are the *Image*, the *Object Model* and the *Pose* class. The image class references the path to an image and stores the corresponding camera matrix. The object model class stores the path to an object model. This allows loading the entities when need. The pose class holds a reference to the participating image and object model, as well as the respective rotation matrix and translation vector.

MISCELLANEOUS CLASSES

Some classes are not displayed in Fig. 4.4, as they were deemed not important to understand the architecture of the program.

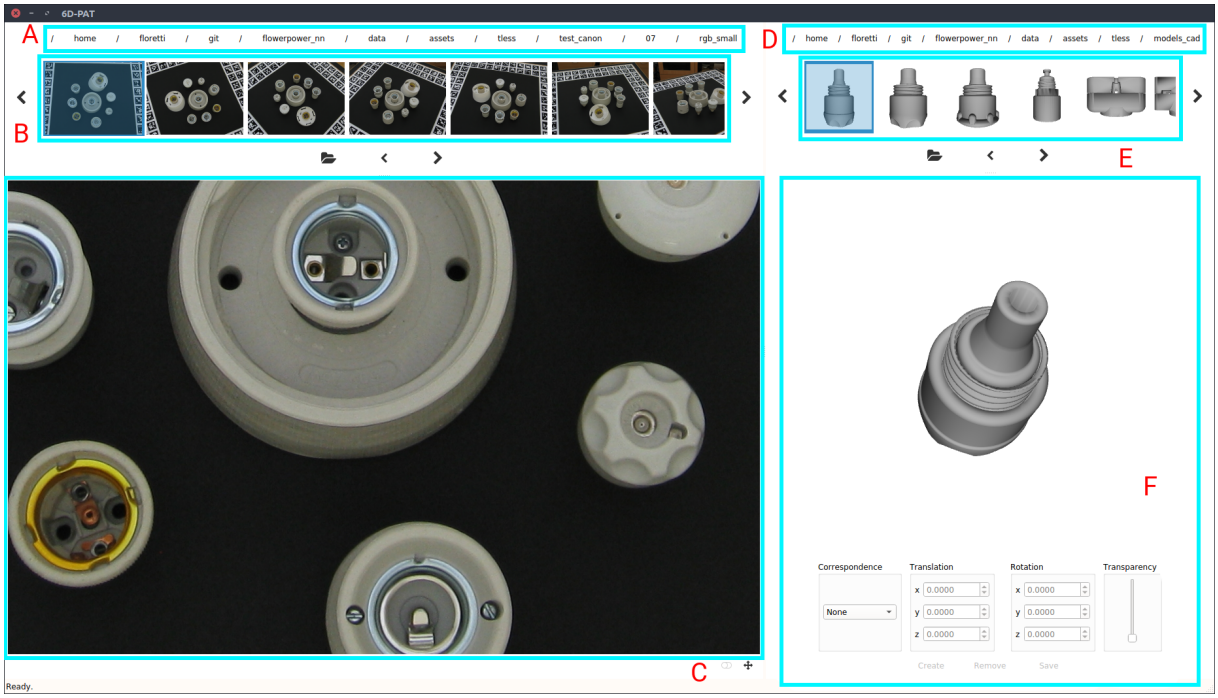
4.3.4. MANUAL ANNOTATION

This section describes the user interface of 6D-PAT and the steps required to annotate images with 6D poses in detail.

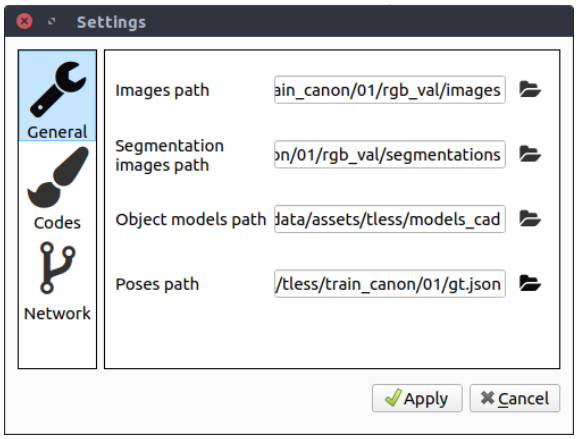
PREPARATION

The first step after starting the program is to open the settings (see Fig. 4.5b) and to set the path to the images that are to be annotated, as well as the path to the folder that contains the object models that are visible in the images.

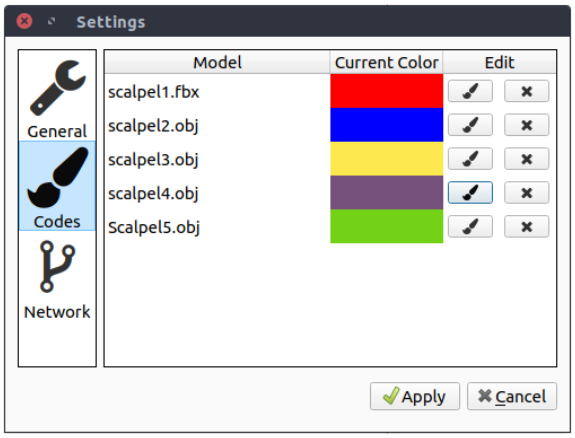
The folder of the images has to contain a JSON file that holds the camera matrix K for each individual image. If no camera info file exists, a Python script that is distributed with the neural network can be used to create approximate camera matrices. The path to the segmentation images, if any, has to be set as well. The program loads the images and the segmentation



(a) The user interface of the annotation tool 6D-PAT. The displayed images and object models are from the T-Less dataset. The following components are marked with a turquoise box: **A**: The full path of the currently selected folder to load images from. **B**: The *Images Gallery* showing the images loaded from the selected path. **C**: The *Pose Viewer* shows the image selected in gallery B. The user can click on the image to define the 2D starting point of a correspondence. **D**: The full path of the currently selected folder to load object models from. **E**: The *Objects Gallery* of rendered 3D previews of the object models loaded from the selected path. **F**: The *Pose Editor* shows the object model selected in gallery E. The user can click on the object model to complete the correspondence with the 3D point. The controls at the bottom can be used to edit existing poses.



(b) The settings dialog of 6D-PAT. The dialog allows editing of the paths where the program loads images, object models and poses from.

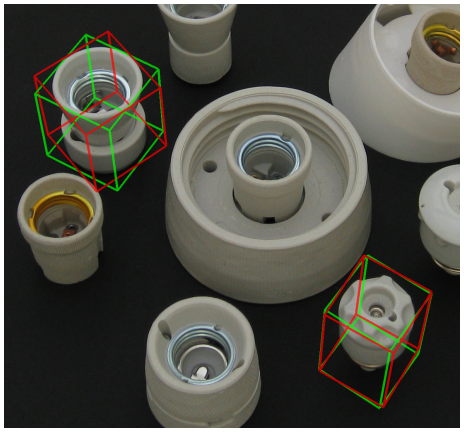


(c) The tab of the settings dialog that can be used to assign colors to the object models.

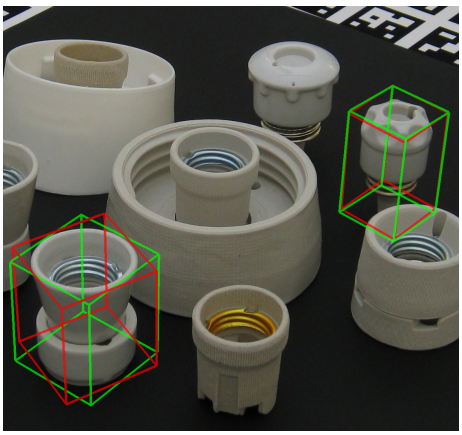
Figure 4.5.: The UI of 6D-PAT.

# Object \ Image	0000.jpg	0150.jpg	0250.jpg	0350.jpg	0500.jpg
01	**2:09** (0:47)	**4:17** (1:05)	**1:31** (0:52)	**1:24** (0:55)	**1:15** (0:52)
15	**2:31** (0:56)	**2:11** (1:15)	**2:27** (0:44)	**1:30** (0:52)	**2:23** (1:06)

Table 4.1.: The table shows the time in minutes the author needed to recover a single pose in an image using 6D-PAT. The bold numbers are the overall time needed to recover the pose, the numbers in parentheses are the times needed to click the correspondences. Images were taken from test scene 7 of the T-Less dataset. The measurements shall give an impression of the tool's efficiency, without making the claim to resemble an empirical study.



(a) Image 0150.jpg shows a nearly top-down view.



(b) Image 0350.jpg shows mix of a top-down and sideways view.

Figure 4.6.: Two example frames from test scene 7 of the T-Less dataset annotated using 6D-PAT. The green bounding box was derived from the ground-truth pose of the dataset, the red box from our recovered pose.

images and sorts them by the numbers in their filenames and matches image I at index j with the segmentation image S at index j.

If segmentation images are present, they are linked to the respective image and can be viewed by activating the toggle at the bottom right corner of the pose viewer. The program displaying a segmentation image can be seen in Fig. 4.2b.

Lastly, the user needs to specify the location of the JSON file where the program is supposed to load existing ground-truth poses from and write new ones to. If no such file exists, an empty one can be created and selected.

If required, the user can assign colors to the object models using the settings dialog depicted in Fig. 4.5c. The colors should correspond to the color used in the segmentation mask. The object models gallery will automatically display only the object models whose color is present in the segmentation image of the currently viewed image. If no segmentation images exist, the gallery shows all object models.

CREATION OF CORRESPONDENCES AND POSE RECOVERY

The components corresponding to the letters that we use in the following paragraph can be taken from Fig. 4.5a. To annotate a new pose, the user has to select an image I from the

images gallery, first (see *B* in Fig. 4.5a). The pose viewer (see *C* in Fig. 4.5a) then displays the image. The object models gallery (see *E* in Fig. 4.5a) can be used to select the object model *O* that the image is to be annoated with. The object model is displayed by the pose editor (see *F* in Fig. 4.5a) after selection. The user can rotate the object to view otherwise hidden areas. Using the arrow keys, the user can move the object along the *x* and *y* axis and also, by pressing the shift key, along the *z* axis.

When the object is in an appropriate position, the user can begin to create a correspondence *C* by clicking on *I*. This defines the 2D location *u* of the correspondence on the image. To complete the correspondence, the object model *O* has to be clicked at the respective 3D point *p*. This procedure has to be repeated until enough correspondences $C_1, \cdots, C_n$ have been defined to recover the new pose *P*. The minimum number of correspondences is 4.

The creation process is shown in Fig. 4.7. More correspondences can make the initial pose more accurate. Clicking the *Create* button at the bottom of the pose viewer recovers the pose *P* by solving the PnP problem implied by the correspondences C_i with OpenCV's *solvePnPRansac* method. The newly recovered pose can be refined using the controls of the pose viewer. After pose refinement, it is necessary to click the *Save* button.

Another intuitive way of recovering poses is dragging the object model from the pose editor onto the pose viewer at the approximately correct position. We assumed that the chosen procedure is faster, as clicking correspondences is a swift action which, when executed diligently, can result in immediately accurate poses. It also resembles the way the neural network pipeline recovers poses (see Chapter 5).

The slider labeled *Transparency* can be used to reduce the object's opacity on the image. After all poses have been annotated successfully, the next image can be selected from the image gallery. This operation has to be repeated until the dataset is fully annotated, although intermediate states can already be used to train the network (see Section 5.5.1).

It is possible to use the neural network to predict poses. This requires a proper setup and training of the network beforehand. The user can then start the prediction process by clicking the *Predict* button in the lower right corner of the pose editor.

Example times of the manual annotation process (without aid of the network) are given in Table 4.1. The times in parentheses show how long it took to click the correspondences, the bold times are the overall times needed. The measured times were achieved by the author of this work trying to recover a single pose of the specified object model. We accepted a pose as correct, when the visual result was subjectively satisfactory. This means that the measurements are not to be seen as an empirical study but as an indicator of the efficiency of the program. The table shows that the time of clicking correspondences does not linearly scale with the overall length of the annotation process. Recovering the pose for object 01 in image 0250.jpg took less overall time than recovering the pose of object 15 in the same image, but clicking the correspondences took longer.

The quality of the initial pose is crucial for the efficiency, as an accurate initial pose needs less corrections. Fig. 4.6 shows two of the example images used for annotation. One of the reasons why the times of recovering poses vary significantly is the angle between the camera and the object. The nearly top-down view in image 0150.jpg prevents clicking correspondences along the object's *z*-axis, which is the largest dimension of the object. The mix of the top-down and sideways view in image 0350.jpg allows to click many correspondences along all axes which

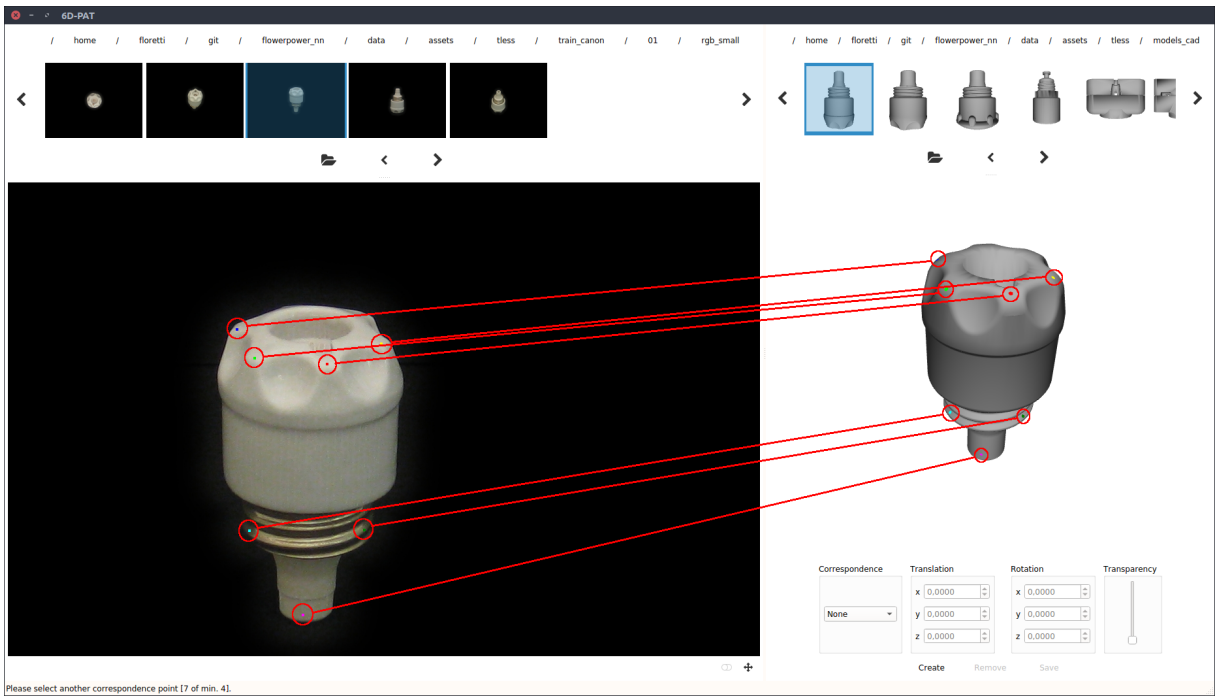


Figure 4.7.: The pose creation process. The user has to click the image first and then the corresponding 3D location on the object model. The red circles and red lines were added afterwards to emphasize the colored dots drawn by the program.

makes the initial pose more accurate.

The rotational discrepancy of the object on the left in the two images is a result of missing characteristics of the 3D model which are part of the real-world counterpart. This made it difficult to recover the pose properly. We neglected this error because the rotation is offset by a similar amount in all annotated images (see Appendix A). The object model on the right has a notch at the top which prevents this rotation error. This issue is discussed further in Section 7.1.

4.4. IMPLEMENTATION DIFFICULTIES

During the implementation of the program, multiple problems arose and complicated the development process. We summarize the most important ones and explain issues which impacted the final design.

One of the major factors that prolonged the implementation process was that we used *Qt3D* to realize all graphical processing at first. Qt3D is part of the main Qt framework. Qt3D's purpose is to encapsulate graphics programming to increase portability of applications including 3D graphics. The incomplete documentation and unintuitive concepts make it difficult to use without consultation. After complications still emerged in the almost completed annotation tool, the Qt3D framework was deemed unusable and omitted in favor of a native OpenGL implementation.

A first try to use the official Python bindings did not work to our complete satisfaction. Including the bindings in the program required complex alterations. Running Python scripts is also possible by starting a new process. Qt natively offers this functionality. Starting a new process

was thus favored over including Python directly in the C++ code.

The Python binding is not complete, yet. Training is not possible and a deep learning expert has to setup the network and its parameters to enable its usage in the program. The current state of the incorporation of the network is to be seen as a proof of concept that requires further development.

5. SEMI-AUTOMATIC ANNOTATION

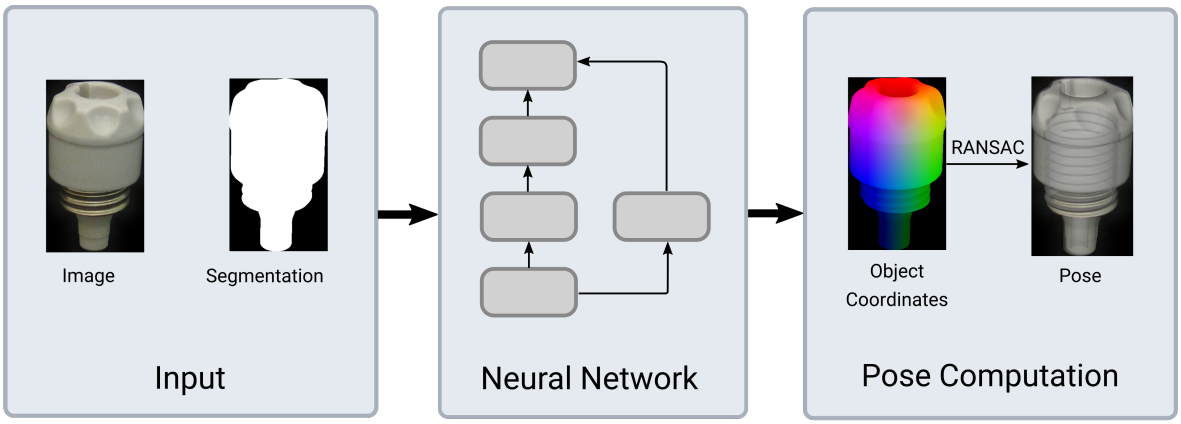


Figure 5.1.: The pipeline of the neural network. The two inputs of the network are the image and the corresponding segmentation image. The neural network then uses the segmentation to retrieve the pixels to predict object coordinates for. In the final pose computation stage the object coordinates and their 2D locations are used to retrieve the best pose using RANSAC.

To assist in the process of manual image annotation presented in Chapter 4, we developed a neural network for the task of 6D pose estimation tailored to the characteristics of the images of the Endoscopic Vision Challenge. This chapter describes the network architecture and its variations, as well as different approaches to train the network and the resulting modified annotation procedure.

5.1. TERMINOLOGY

Residual Connection. A *residual connection* (or *skip connection*) is a part of a neural network that skips some layers of the network and adds the input of a layer to the output of a layer that is situated deeper in the network. They can prevent the problem of the increasing training error in deeper networks [6]. Fig. 5.2a visualizes such a connection.

Training Set. A *training set* is a subset of the dataset intended to train a neural network with. The network uses the training set to adjust its weights as opposed to the validation set.

Training Example. A *training example* is an element from the training set fed to the neural network during training.

Training Batch. A *training batch* consists of a subset of training examples. During training the network updates its weights using all examples in the batch.

Validation Set. A *validation set* is a subset of the dataset intended to train a neural network with. The validation set's purpose is to validate the network's new configuration which emerged from a run within the training.

L1 Loss. The *L1 loss* is a loss function that sums up the results of the function g applied to

the k elements in a training batch, i.e. $L_1(x, y) = \frac{1}{k}\sum_k |g(x_k, y_k)|$. We set g to be the euclidean distance: $g(x_k, y_k) = \sqrt{\sum_i (x_{ki} - y_{ki})^2}$, x_{ki} and y_{ki} being the i-th entries of the vectors x_k and y_k.

L2 Loss. The *L2 loss* is a loss function defined similarly to the L1 loss but sums up the squared instead of the absolute results of g, i.e. $L_2(x, y) = \frac{1}{k}\sum_k g(x_k, y_k)^2$.

L2 Regularization. *L2 regularization* penalizes the weights by a factor λ in the following way: $\lambda \sum_i w_i^2$. The regularization term is added to the weight update to keep the network from overfitting.

Error. The term *error* denotes the value computed using the loss function. It is essentially the discrepancy between the desired output and the output of the network. It is also called *error rate* or *loss* or, when the loss is computed only on the training data, *training error* or *validation error*, when computed on the validation set.

Performance. The term *performance* denotes the network's accuracy in this work.

Dropout Percentage. *Dropout percentage* is the percentage of neurons to deactivate in a layer during training.

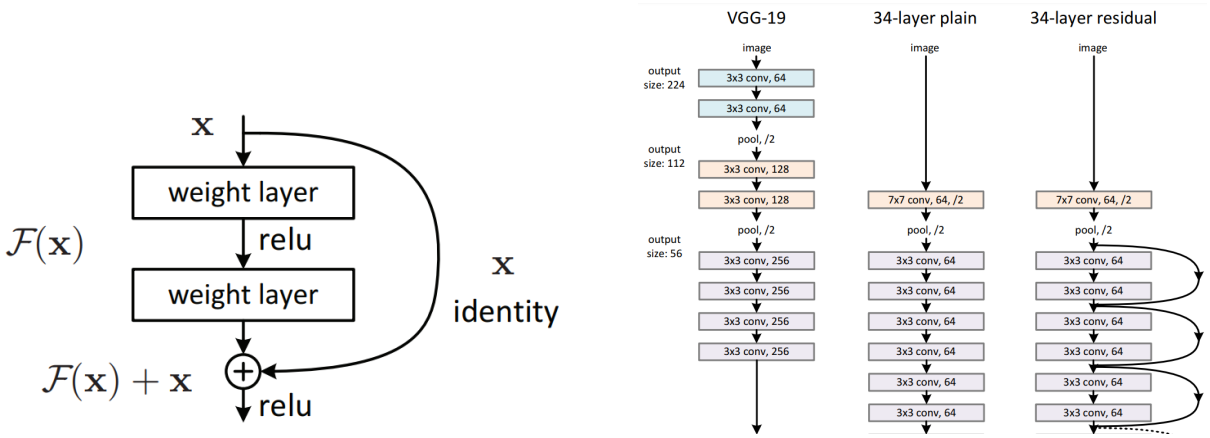
Receptive Field-Size. The *receptive field-size* of a network denotes how many input pixels an output value of the network has taken into account. This factor is determined by the number of operations in the network and their parameters. A convolution operation can skip pixels, which results in a larger receptive field-size. Increasing the kernel size has the same effect. Depending on the field of application a field-size should be larger or smaller.

5.2. FRAMEWORKS

The neural network and all of its functionality is implemented in Python using Tensorflow [58] and Keras [59]. Tensorflow is a deep-learning framework that uses C++ to perform the actual calculations. It provides many different types of network layers, as well as tools to modify images, automatic differentiation of the loss function and optimization algorithms. Keras is a framework that makes using Tensorflow easier and allows to create neural networks in a few lines of code. Most of the operations of Tensorflow and Keras are performed in C++ but both can be easily used from Python. Both fromeworks offer functionality to transfer the computations to the graphics cards. This allows training deep networks in a feasible time [5].

5.3. NETWORK ARCHITECTURE

The architecture of the network is adjusted to the problem domain of the images of the Endoscopic Vision Challenge. There is no mechanism of inferring the masks and the network expects RGB-only images without depth information. This significantly reduces the complexity



(a) A detailed view on a residual connection. The residual connections add the input of earlier layers of the network to deeper ones.

(b) Excerpt of a comparison between the ResNet architecture and VGG. ResNet allows the creation of deeper network, as the problem of vanishing gradients can be targeted this way. The image has been cropped.

Figure 5.2.: A key component of ResNet: the residual connection. The left image shows the connection in detail and the right image shows a comparison with the VGG architecture. Images from [6].

of predicting the position of the object. Many contradicting predicted object coordinates can still lead to positional errors, though. Fig. 5.1 shows the processing pipeline of the network which takes the image and the segmentation image as inputs. The network processes each pixel which is part of the object according to the segmentation image and outputs the 3D coordinates. As explained in Chapter 2, we then compute the optimal pose using RANSAC and the 2D-3D correspondences.

The basic architecture that we chose for the network is called ResNet. He et al. first presented ResNet in 2015, which enabled researchers to create deeper networks than before [6]. Two major problems arise in very deep neural networks. One is the vanishing gradients problem, which means that gradients gradually become 0 in deeper layers. A 0-gradient implies that the weights don't change anymore and the layer can't take in the information of the training examples. This problem can be targeted with batch normalization [6]. The second problem is that making a network deeper without further adjustments can lead to an increasing training error. Adding layers to a network to increase expressivity does not directly lead to a higher accuracy. Even stacking identity layers, i.e. layers that learn the mathmatical identity function $f(x) = x$, on top of the network increases the training error. This phenomenom can be compensated with residual connections [6]. An example of a residual connection can be seen in Fig. 5.2a. Fig. 5.2b shows a comparison of ResNet with the architecture *VGG* [39], which is another well known neural network design. The residual connections are clearly visible as the arcs of the rightmost architecture.

We chose ResNet over other architectures due to its easy scalability and its still superior performance in terms of accuracy. *Mask RCNN* is a prominent example that makes use of ResNet's structure [60]. The Facebook AI Research team developed Mask RCNN. The network is still ranking within the top 5 networks of the MS COCO challenge, as of July 2018. The MS COCO challenge is a challenge on image segmentation and object detection [42].

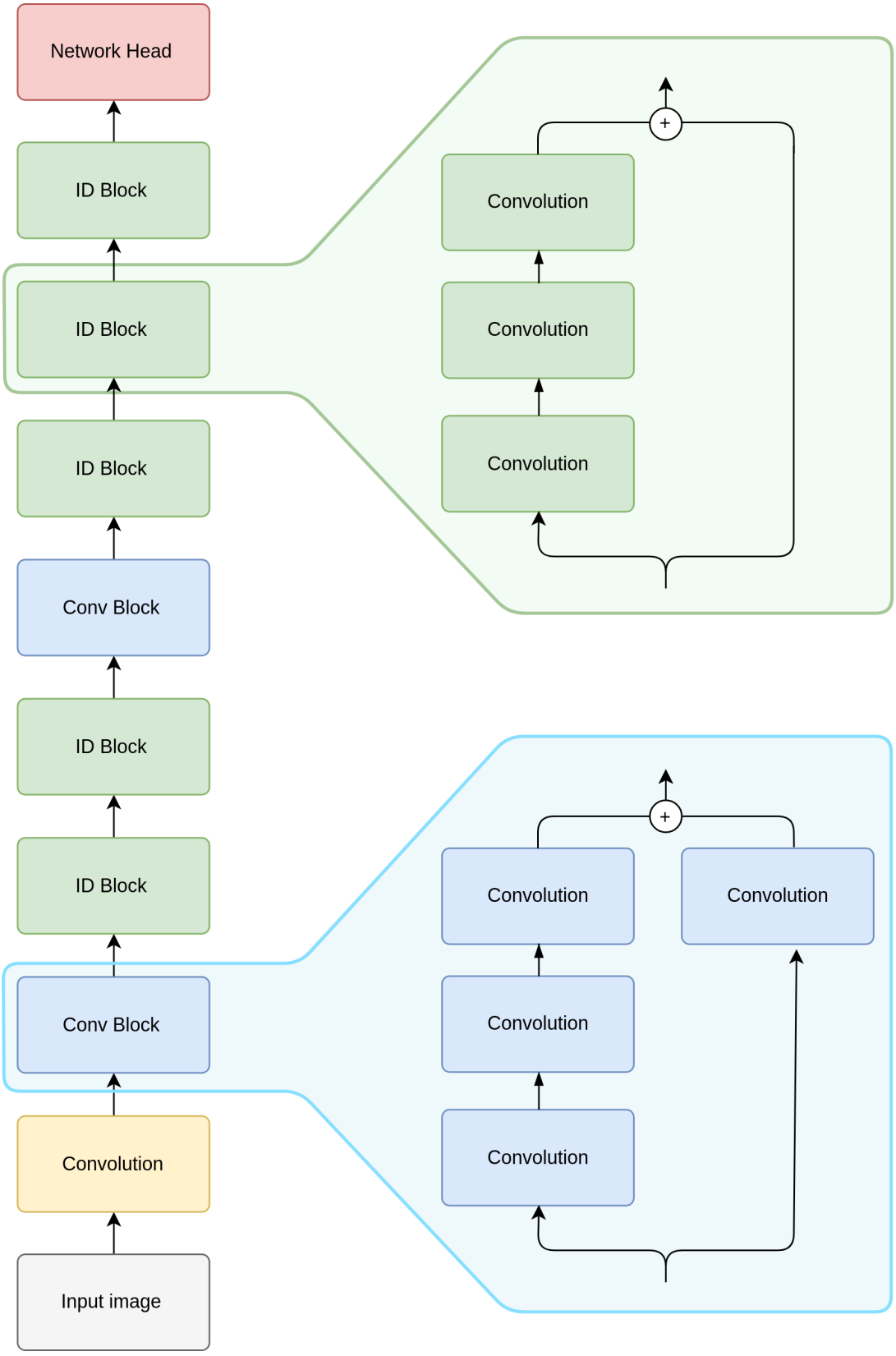


Figure 5.3.: Architecture 1 with a total of 23 (convolutional) layers. The components of the **Conv Block** and **ID Block** elements are visualized on the right. The characteristic of the Conv Block is that it applies a convolution on the shortcut connection while the ID Block doesn't. The yellow **Convolution** rectangle is a plain convolution layer. The **Network Head** consists of another three convolution layers.

The structure of the network is visualized in Fig. 5.3. The depicted architecture has 23 convolutional layers (see Section 5.3.2 for a detailed explanation on used architecture variations). The building blocks of the network are the *Conv Block* and the *ID Block*. Both blocks are shown in detail on the right. A Conv Block applies another convolution to the shortcut connection on the right and then combines the result with the output of the left path. The ID Block simply adds the input to the result of the convolutions on the left. The figure doesn't show the batch normalization and *activation layers* to keep the illustration clear. The actual network has a batch normalization layer followed by an *activation layer* after each convolution layer. Activation layers are layers in Keras that apply a specified activation function to their input. This way, non-linear neurons are realized in the framework. All layers employ the ReLU activation function.

5.3.1. LOSS FUNCTION & OPTIMIZER

Since the network outputs the 3D coordinates for each pixel in the input image a straight-forward approach is to penalize the euclidean distance with respect to the ground-truth coordinates. The loss function sums up these distances of the individual XYZ components of the predicted object coordinate and the ground-truth, but only at the relevant positions according to the segmentation mask. This resembles the L1 loss (see Section 5.1). The L2 loss potentially penalizes outliers more. But outliers get eliminated in the RANSAC algorithm during pose computation and thus might not have to be taken into account by a great extend. The L2 loss might strive for a worse compromise between outliers and accurate coordinate predictions. To verify our beliefs we compared both loss functions in Chapter 6. To find the best optimizers, we also compared SGD and Adam (same chapter).

5.3.2. VARIATIONS

The problem formulation of this work demands exploration of multiple architecture designs. There are multiple reasons why. The Endoscopic Vision Challenge dataset does not provide any pose annotations at all. This means that the user likely trains the neural network with few images at first.

A very deep network with many layers might not be able to incorporate the little information optimally. Conversely, a shallower architecture might not reach the same performance as a deeper network.

A characteristic of the Endoscopic Vision Challenge dataset is that the tools have writings on them which are often not or only partly visible. A network with a large receptive field-size could tend to predict 3D coordinates based on this very distinct feature, even for pixels that are further away. This way, less prominent characteristics, that could still be used to accurately identify the object coordinate belonging to a pixel, might be disregarded and the pose correctness might drop when the letters are not visible.

Employing batch normalization can reduce the training time needed for a network, as well as increase its accuracy [13]. To exploit the full potential of batch normalization, a batch has to consist of enough training examples so that the normalization represents a good approximation of the statistics of the dataset. Due to the limited memory resources on graphics cards, a large batch size is not always possible for deep networks because Tensorflow transfers the

Architecture No.	1	2	3	4	5	6
# Layers	**23**	**35**	**23**	**50**	**50**	**50**
Receptive field-size	99	67	51	59	59	59
# Parameters	3,95	19,69	6,32	24,63	24,63	24,63
Data normalization	bn	bn	bn	do	bn	do

Table 5.1.: Overview over the differences between the network architectures. The number of layers corresponds to the number of convolutional layers. Batch normalization layers are not counted. The abbreviations *bn* and *do* stand for batch normalization and dropout, respectively. The number of parameters is given in millions. The number of parameters resembles the number of trainable variables in the network.

training examples to the GPU memory for optimal speed. This is the reason why we believed it necessary to compare batch normalization and dropout. These challenges gave rise to the architectures described in the following paragraphs.

We created the initial design of the network with 23 convolutional layers and a receptive field-size of 99. The second architecture consists of 35 layers, while decreasing the receptive field-size to 67.

The third architecture has the same number of layers like the first architecture but a reduced receptive field-size of 51. Architectures 4 to 6 all have a receptive field-size of 59 and 50 convolutional layers. While architecture 5 uses batch normalization for regularization, architectures 4 and 5 use dropout instead. The difference between those two is the dropout percentage. An overview over the architectures can be seen in Table 5.1.

The number of parameters does not scale linearly with the number of layers, as deeper layers have more filters. The reason why architecture 3 has more trainable parameters than architecture 1 is not entirely clear. The difference between the two networks is that architecture 3 uses a reduced kernel size on deeper layers to obtain a smaller receptive field-size. But this should not cause the number of parameters to grow in a fashion that is visible in the table.

The different depths of the network architectures are visualized in Appendix B. The architectures that omit the batch normalization layers and employ dropout layers instead are not listed separately, because the number of the convolutional layers is the same as in architecture 5.

5.4. SETUP & IMPLEMENTATION DETAILS

Before the user can run the network to train on a dataset, some steps to setup the network have to be performed. Existing ground-truth files of a dataset have to be converted to the format expected by the network. An example conversion function is provided, which converts the T-Less format to the JSON format that the network is able to read.

The T-Less dataset does not provide segmentation masks, nor object coordinates. The mask and coordinate images can be rendered using the images, the ground-truth poses file and the utility scripts of the network. After obtaining all data necessary for training, the user needs to write a JSON configuration file, which holds the paths to the data and the parameters of the network. The network architecture that is to be trained can be defined directly in the file.

The network consists of convolutional, batch normalization and pooling layers only. This means, it can be applied to arbitrary image sizes without having to retrain the network, as

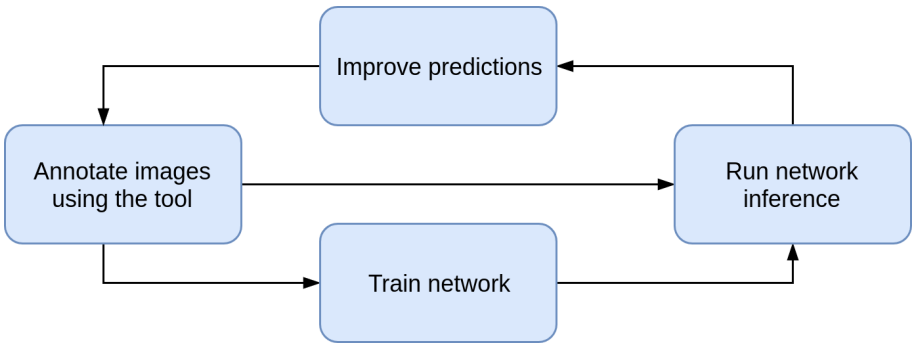


Figure 5.4.: The new workflow of the annotation procedure including the neural network. First, the user has to annotate enough images from the dataset to be able to train the network. After successfully training the network it can be used to predict poses on a subset of the dataset. The user can then improve the poses and return to annotating more images or run the network inference on more images.

in contrast to networks employing fully-connected layers. If the image size changes, a fully-connected layer has to be retrained but a convolutional layer not. The maximum image size needs to be specified in the configuration file, nevertheless. This is necessary because the network is trained in batches instead of single training examples and Keras expects all samples to have the same dimension. The training procedure automatically scales up images smaller than the configured size as much as possible and, for non-square images, pads the remaining space with 0s. It is important to note that during training no cropping of the input data to the relevant area takes place because, to save disk space, we assume the user always crops the training data beforehand. In contrast, the network crops images to the mask area before an inference run. Thus, the data does not need to be altered in any way when only predicting poses.

5.5. MODES OF OPERATION

The goal of the neural network is to make the annotation process presented in Chapter 4 faster and more efficient. To achieve this, we strove for a symbiosis between the annotation tool and the neural network. The new workflow is visualized in Fig. 5.4. In addition to manually annotating images, the user can train the network with the annotated data. After training, the neural network can be used to predict poses in previously unseen images. As the network will likely make errors, the poses can be corrected afterwards using 6D-PAT. The improved pose predictions can serve as input to further train the network. As the dataset of the Endoscopic Vision Challenge does not provide any pose annotations, the question how to optimally train the network with a reduced set of images arises. The following sections describe different strategies which are evaluated in Chapter 6.

5.5.1. INCREMENTAL TRAINING

During the annotation process, the user can train the network using the new data. Given that at the beginning likely no poses exist and the user first has to annotate a couple of images manually, we have to use the little available data as optimally as possible. One option is to fully retrain the network with all annotated images. This procedure is called *training from scratch*.

It is also possible to train the network incrementally, instead. *Incremental training* means that the user trains the network using the annotations that were created after the last training run. In this approach, the weights of the network are not initialized to 0 or randomly. Instead, the weights of the last training run are loaded and trained further using the new data. This method is likely to converge faster and generally needs less time to complete the training compared to training from scratch. We compare the network performance of both approaches in the next chapter.

5.5.2. ACTIVE LEARNING

An intuitive approach to optimally train the network with only a few images is to propose to the user to annotate images which the network does not perform well on. This gives rise to the question how to obtain an indication of the network's accuracy without any ground-truth poses. As the Endoscopic Vision Challenge dataset provides segmentation masks, a simple but direct approach is the compute the intersection over union between the given segmentation mask and a rendered mask using the inferred poses. The intersection over union is defined in the following way

$$\text{IoU} = \frac{\sum_{(i,j)} \mathbb{1}[s_g(i,j) = q] \cdot \mathbb{1}[s_p(i,j) = q]}{\sum_{(i,j)} \frac{\mathbb{1}[s_g(i,j)=q] + \mathbb{1}[s_p(i,j)=q]}{2}}$$

where s_g and s_p are the segmentation mappings of the ground-truth segmentation mask and the rendered segmentation mask, respectively (see Section 4.1 for details on segmentation masks). q is the type of the object model. $\mathbb{1}$ is a function that becomes 1, if the statement inside is true, 0 otherwise. Therefore, the formula sums up all pixels that both segmentation masks assign the same type to. It then divides this sum by the whole area covered by this type in both masks. Note that, by dividing the whole area by 2, the IoU becomes 1 if the masks match exactly. The training process can suggest images to the user that need correction of the recovered poses based on this measure. The next chapter explores this method experimentally.

6. EXPERIMENTS

(a) An example frame from the training dataset of T-Less.

(b) An example frame from the test dataset of T-Less.

Figure 6.1.: Example frames from the T-Less dataset [15].

The following section describes the setups of the experiments conducted with the neural network presented in Chapter 5. Due to the lack of existing annotated medical data, the focus of the experiments is to ascertain a network design and mode of operation that compensates this as best as possible. First, we test the different network architectures that were described in section 5.3.2 using the same parameters. We then choose the best architecture for further experiments which we use to evaluate the two different training schemes introduced in section 5.5: training from scratch and incremental training.

6.1. TERMINOLOGY

Hyper-Parameter. *Hyper-parameters* are the tunable parameters of a network. The number and kind of parameters vary with the used type of network architecture, optimizer, etc.

6.2. DATASETS

The initial intention to completely annotate the medical images of the Endoscopic Vision Challenge dataset turned out to be not feasible (see Chapter 4). Instead, we chose the T-Less dataset to conduct the experiments with. Its objects are mostly texture-less and of rather small size, making them similar to surgical tools. Next to many human pose estimation datasets, there exist some datasets of objects, too, like [61], [62] and [63]. But the objects of T-Less resemble surgical tools more than the objects of those datasets.

The T-Less dataset was released in 2017 by Hodaň et al. [15]. It contains the object models of 30 industry-relevant real world objects. Two versions exist: one mesh of the object that was manually reconstructed using CAD software and one that was produced from RGB-D images. The 3D coordinates of the objects reflect their size in millimeters. For example, the minimum and maximum of any component of the coordinates of object model 01 are about -30 and 30. This means that the model is about 60 mm long. The objects were captured using a Primsense CARMINE 1.09, a Microsoft Kinect v2 and a Canon IXUS 950 IS. We used the photos of the Canon camera due to their quality and since we do not need depth information, which is also provided by the CARMINE sensor and the Kinect. There are 1296 images of each object in the training set of the dataset, sampled in 10 degree steps in elevation and 5 degree azimuth. 20

test scenes, consisting of 504 images each, exist as well. Those are photographs of cluttered scenes of different complexity, with sometimes more than 15 objects visible. All training and test images are annotated with the ground-truth poses of the visible objects. The authors also provide a tool set to render the objects at a given pose. Fig. 6.1a shows an example frame from the training set and Fig. . 6.1b an example frame from the test set. The training images of the other objects look similar to the depicted image. The dataset provides depth images, as well but this is irrelevant to our setting.

6.3. DATA PREPARATION

The T-Less dataset does neither provide 3D object coordinates nor segmentation masks. This is the reason why we rendered both ourselves. The rendering script is part of the network package. Because the 16-bit TIFF object coordinate ground-truth files used up too much disk space when kept in their original size, they were cropped to the relevant area by determining the smallest box around the segmentation pixels. The images and segmentation masks were cropped to the same area and the camera matrices adjusted accordingly. We used 16-bit TIFF files instead of 32-bit because we deemed the accuracy of 16-bit to be high enough.

6.4. EVALUATION METRIC

We developed a set of evaluation metrics to capture different aspects of the prediction quality of a network. We assumed that multiple metrics provide better insights of the strengths and weaknesses of an architecture. To this end, we provide functionality that directly assesses the quality of the predicted object coordinates and also functionality that provides details on the accuracy of the resulting recovered pose.

The *Object Coordinate Error* e_{coord} computes the euclidean distances between the predicted and ground-truth object coordinates:

$$e_{\text{coord}} = \frac{1}{|S_O|} \sum_{(i,j)} \sqrt{(g_{ij1} - p_{ij1})^2 + (g_{ij2} - p_{ij2})^2 + (g_{ij3} - p_{ij3})^2}$$

where (i, j) are all 2D locations belonging to the object model O according to the segmentation mask. $|S_O|$ denotes the cardinality of the set of these locations. g_{ijk} and p_{ijk} are the k-th components of the object coordinate at (i, j) in the ground-truth and predicted object coordinates image, respectively.

The number of *Inliers* n_{inlier} captures the number of object coordinates whose euclidean distance to the ground-truth object coordinate is below a certain threshold:

$$e_{\text{inlier}} = \left| \left\{ (i,j) \;\middle|\; \sqrt{(g_{ij1} - p_{ij1})^2 + (g_{ij2} - p_{ij2})^2 + (g_{ij3} - p_{ij3})^2} \leq \sigma_{\text{thresh}} \right\} \right|$$

where σ_{thresh} is the chosen threshold and (i, j) are all locations belonging to the object model. We set the threshold to 2.

Both e_{coord} and n_{inlier} assess the accuracy of the predicted object coordinates. The metrics described next provide a measure of the quality of the recovered pose.

The *Distance Error* e_{dist} is the euclidean distance between the ground-truth position of the object and the position computed by RANSAC. e_{dist} is computed as:

$$e_{\text{dist}} = \sqrt{(t_{g1} - t_{p1})^2 + (t_{g2} - t_{p2})^2 + (t_{g3} - t_{p3})^2}$$

where t_{gk} denotes the k-th component of the ground-truth translation vector, and t_{pk} the k-th component of the translation vector recovered from the coordinate predictions.

The *Angle Error* e_{angle} is the rotational error in degrees between the ground-truth and the calculated rotation matrix:

$$e_{\text{angle}} = \frac{cos^{-1}(\frac{\text{Tr}(R_g^T R_p)-1}{2}) \cdot 180°}{\pi}$$

where R_g and R_p denote the ground-truth and predicted rotation matrix, respectively and $Tr()$ denotes the trace of a matrix.

Finally, we implemented the metric presented by Hinterstoisser et al. in [25], which computes the average distance between each vertex of the object model with the ground-truth and recovered pose applied to it. This metric e_{pose} is defined in the following way:

$$e_{\text{pose}} = \frac{1}{|M|} \sum_{X \in M} ||(\bar{R}X + \bar{t}) - (RX + t)||$$

where M is the set of points of an object model O. $\bar{R}$ and $\bar{t}$ denote the rotation matrix and translation vector of the ground-truth pose $\bar{P}$, while R and t are the respective components of the recovered pose. A pose is an *inlier*, if

$$e_{\text{pose}} \leq khs \cdot d$$

where d is the diameter of the object and khs is a constant which we set to 0.1 for the rest of this work. The percentage of inliers is denoted as *inlier %* in the tables of the experiments.

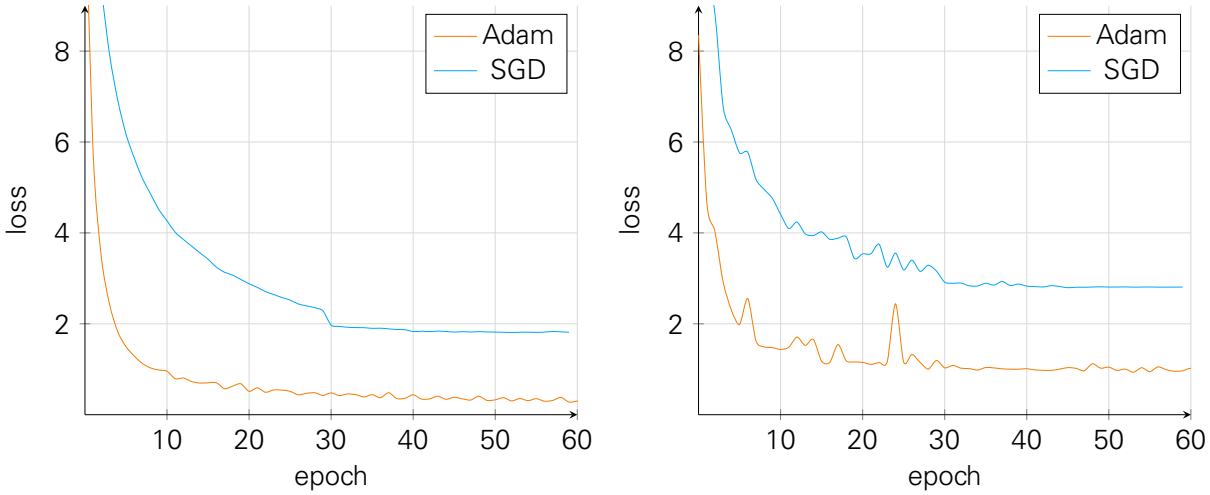
The scripts that we provide compute various partitions of the metrics. Next to the mean for all five metrics, the median at 25, 50 and 75% is calculated for each metric individually, as well. This makes it easier to assess whether a network produces inaccurate predictions for only a part of the dataset. We only provide the mean values in this chapter though because using the median for comparison requires to compare at least all three medians computed by the script. Otherwise anomalies can occur, for example because the pose inlier rate can only be computed as the average over all images which in some rare cases contradicts the result of comparing median at 50 % metrics of different networks.

6.5. TRAINING EXPERIMENTS

This section describes the configuration of the different training experiments. First, we compare the SGD and Adam optimizer in 6.5.1. Then, we evaluate the different architectures in 6.5.3. Finally, we asses training from scratch and incremental training in Section 6.5.4. We conducted all experiments on the object model 01 of the T-Less dataset. The image dimension was set to 500 pixels (for width and height), which is larger than the largest area covering an

Run	1	2	3	4	5
Epochs	30	45	55	65	70
Learning Rate	0.001	0.0001	0.00001	0.000001	0.0000001

Table 6.1.: The configuration of the SGD optimizer used in the experiment to compare SGD and Adam.



(a) Training losses.

(b) Validation losses.

Figure 6.2.: Training and validation losses of Adam and SGD used to train architecture 1. The plot is cropped on the *y*-axis to enhance the differences.

object in any image. All experiments were run using the training data of the object model 01. For some experiments we only used a subset of the available images (see Sections 6.5.4 and 6.5.5) but we always split the data into 70% training images and 30% validation images. The images were assigned to one of the sets at random. If not stated otherwise, we used the same training and validation set throughout the experiments. The axis labeled *epochs* in the loss graphs denotes the number of epochs the training was run for. Each epoch consisted of 1000 iterations in every experiment.

6.5.1. OPTIMIZERS

To obtain the best optimizer for further experiments, we compared SGD and Adam using the first architecture we constructed. This architecture consists of 23 layers and has a receptive field-size of 99. The hyper-parameters β_1 and β_2 of the Adam optimizer were left at the default values set by Keras, which are 0.9 and 0.999, respectively. The more complex configuration of the SGD optimizer is given in Table 6.1. Fig. 6.2 shows the loss of both experiments during training. The SGD optimizer profits from the reduction of the learning rate around epoch 30, visible as the small step in the training loss. This could imply that further tuning of the training parameters of SGD could lead to better results. But since the Adam optimizer does not need manual fine-tuning of its hyper-parameters and performs visibly better than SGD, we chose Adam as the optimizer for all following experiments. We based our conclusion that Adam is the superior optimizer for our network solely on the loss values. Since we trained the same

Loss	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
L1	**1.6494**	**569.4318**	**1.6416**	**5.4331**	**5.4627**	**78.92**
L2	1.5429	546.5629	2.0507	6.2401	6.2713	70.17

Table 6.2.: The metrics on the **validation set** of the experiments comparing the loss functions.

(a) Image 0045.jpg with inferred poses.

(b) Image 0438.jpg with inferred poses.

Figure 6.3.: Example images from test scene 7 of the T-Less dataset with poses inferred by architecture 1 trained on all images.

network architecture twice and only varied the optimizers, better error rates mean better overall results.

6.5.2. LOSS FUNCTIONS

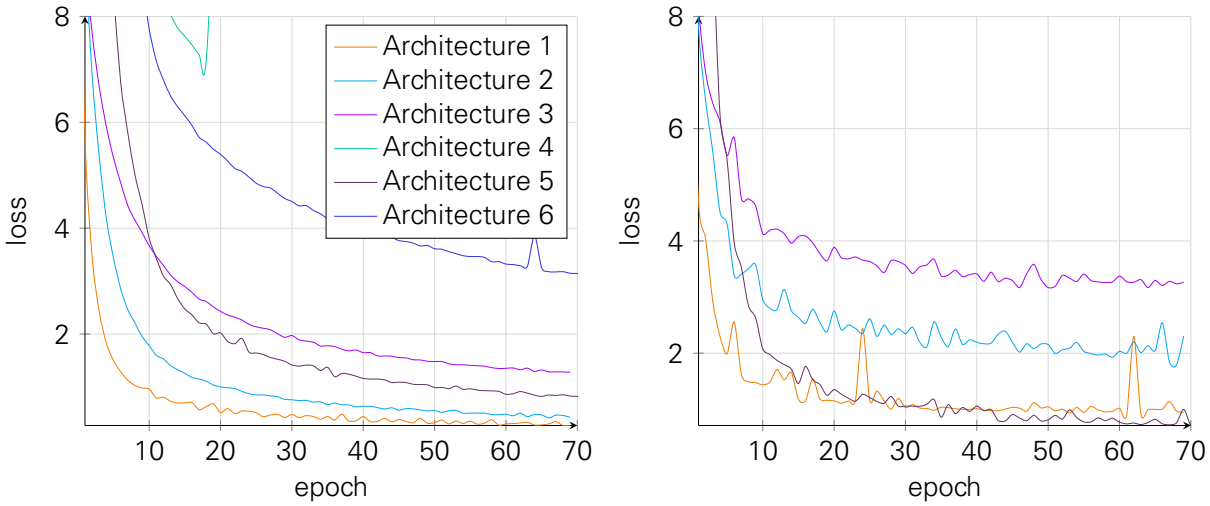
To prove our statement that the L1 loss is more suitable than the L2 loss for our application, we conducted another training run using the L2 loss and compared it to the run using the L1 loss of the previous experiment. The loss function comparisons are depicted in Appendix C.1 but are omitted here, due to their limited comparability. To obtain the L2 loss, we removed taking the square root from the loss function which naturally results in higher error rates. Table 6.2 shows the metrics of the experiments with the different loss functions. The L1 loss offers superior accuracy and is used in all further experiments.

6.5.3. ARCHITECTURES

The experiments run on the different architectures were all performed using the Adam optimizer with the parameters mentioned in Section 6.5.1 and the L1 loss. Therefore, the only difference is the used architecture. The losses of the experiments are displayed in Fig. 6.4 and two example images from test scene 7 of the T-Less dataset with rendered bounding boxes of the poses inferred by architecture 1 are shown in Fig. 6.3. The same images with poses

Architecture	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
1	**1.6494**	**569.4318**	1.6416	5.4331	5.4627	78.92
2	3.0175	482.0385	6.2531	14.1186	14.3991	77.3778
3	4.5000	322.1516	5.2395	11.0992	11.2473	59.8971
5	2.3125	510.7506	**1.1989**	**2.4177**	**2.4640**	**97.6863**

Table 6.3.: The evaluation metrics on the **validation set** of architectures 1 - 3 and 5, which were trained using all available images.



(a) Training losses.

(b) Validation losses.

Figure 6.4.: Training and validation losses of the different architectures. The *y* axis is cropped to enhance the more delicate differences at the lower end. The validation losses of architectures 4 and 6 are too high to be displayed here.

inferred by architectures 2, 3 and 5 are provided in Appendix C.2. The largest gap of the losses is between architectures 4 and 6 and the four remaining architectures. This gap is especially large in the validation loss but also observable in the training loss.

Architectures 4 and 6 employ dropout, as opposed to architectures 1 - 3 and 5, which use batch normalization (with batch sizes of 14, 10, 14 and 3, respectively). The authors of [13] propose that batch normalization reaches better optima than dropout, which could explain the training loss difference. The validation losses of the two dropout architectures actually increase, which is not visible in the graph. This indicates that the networks overfit to the data. The constantly increasing validation loss is the reason why we did not keep the training processes of architectures 4 and 6 running, despite their training errors still decreasing. Proper exploration of hyper-parameters, like frequency of the dropout layers, as well as the dropout frequency itself, could improve the performance of architectures 4 and 6. But since batch normalization achieves a good accuracy without further fine-tuning, we did not investigate these architectures any further.

The metrics of architectures 1 - 3 and 5 are provided in Table 6.3. Architecture 5 performs best, followed by architecture 1. The table shows that multiple different metrics can give a deeper grasp of a network's performance, as architecture 1 has a significantly lower object coordinate error (e_{coord}) than architecture 5 but their pose errors (e_{pose}) are of similar quality.

Architecture	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
1	13.9062	60.7162	67.7789	158.3841	160.7716	5.7539
2	14.6875	55.8055	68.1237	183.78932	186.3252	4.7619
3	15.4296	31.7361	85.2801	262.3253	265.8469	0.7936
5	**9.2421**	**161.6011**	**49.4737**	**119.0649**	**120.8825**	**7.7380**

Table 6.4.: The evaluation metrics on the **test set** of architectures 1 - 3 and 5, which were trained using all available images.

Architecture 2 offers a better object coordinate error than architecture 5 but achieves a worse pose error.

The results of the training experiments of the different architectures lead us to choose architectures 1 and 5 for further experiments, i.e. a shallower architecture with larger receptive field-size and a deeper architecture with reduced receptive field-size. Table 6.4 shows the evaluation metrics for the networks on test scene 7 of the T-Less dataset. Architectures 1 and 5 achieve the best accuracy which supports our decision to use them in the following experiments which investigate how these two different designs behave when trained with smaller datasets.

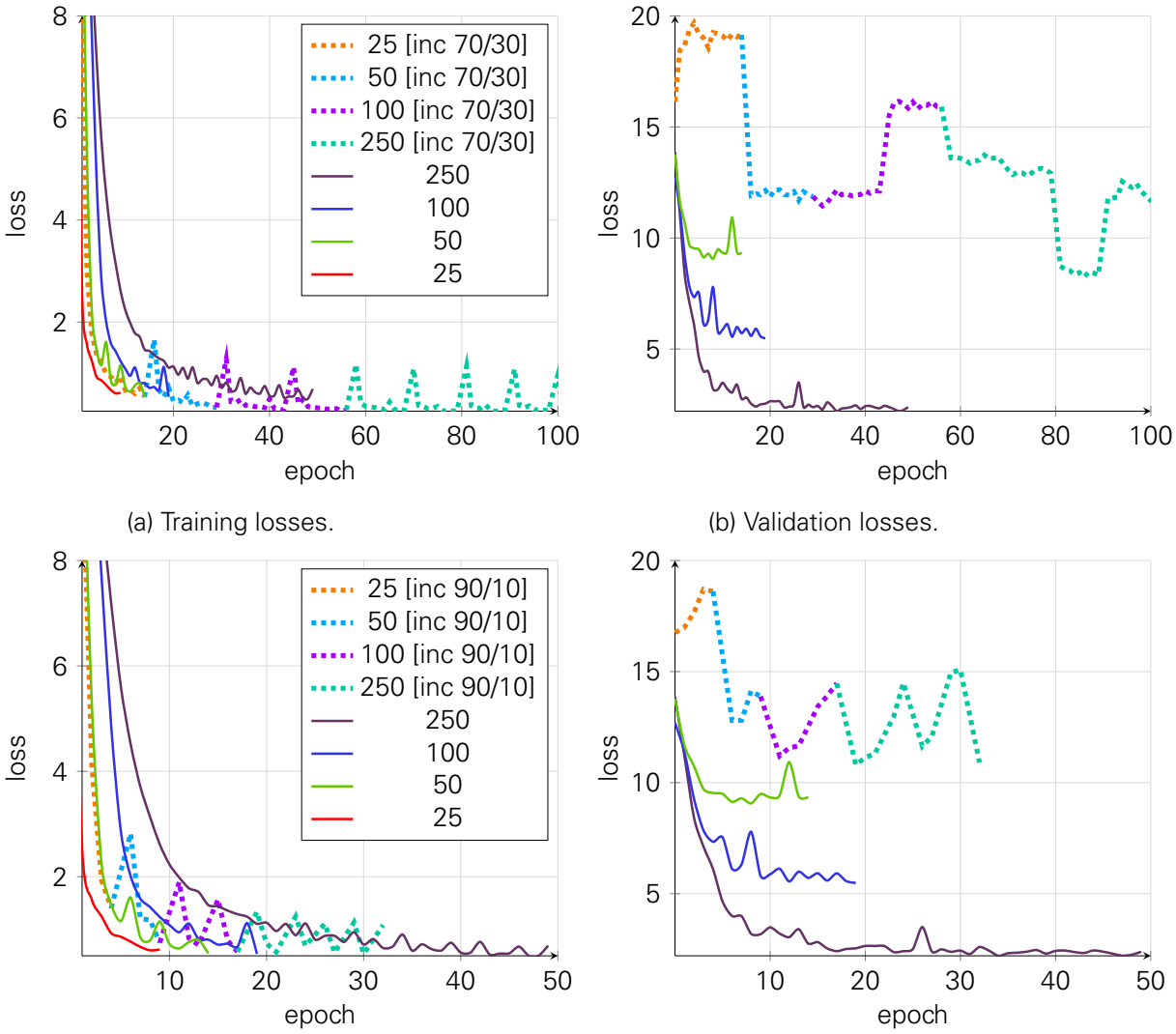
6.5.4. INCREMENTAL TRAINING

To propose a training strategy that makes optimal use of the little data available at the beginning of manually annotating an unannotated dataset, we investigate here whether the network should be trained incrementally or from scratch. To this end, we trained architectures 1 and 5 incrementally with datasets consisting of 25 images. Thus, we trained the network using 25 images and then continued to train it with the next set of 25 images. We repeated this process until we reached the desired number of 250 images. As we assumed that the proportion of the split of training and validation set becomes more relevant for such a small dataset, we ran one set of experiments using 70% of the images for training (i.e. 18 images) and 30% for validation (i.e. 7 images) and another set using a 90/10 split (22 images for training and 3 images for validation). The experiments utilizing the 70/30 split used the validation images of previous experiments for validation again, while the experiments of the 90/10 split did not. None of the training runs used training images of the previous experiments again.

The reason why we created datasets of 25 images is that we deemed this a number that is quickly reachable by manual annotation. We also trained the network from scratch using 50, 100 and 250 images using a 70/30 split and compared it with the performance of the network trained incrementally with the same number of images. The smaller datasets were created by extracting images at regular intervals from the whole set. The validation images were then

Method	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
from scratch (25, 70/30)	11.4687	53.5200	83.6374	13.4300	23.2540	5.3333
from scratch (50, 70/30)	7.8828	98.8333	51.6051	25.5187	29.1855	9.3333
from scratch (100, 70/30)	4.3750	219.6066	15.8770	21.2478	21.9831	37.3333
from scratch (250, 70/30)	**2.0234**	**475.93333**	**2.1194**	**4.4683**	**4.5293**	**78.6666**
incremental (50, 70/30)	11.9609	10.8750	93.0100	39.4105	45.0582	0.0000
incremental (100, 70/30)	10.5156	29.3750	92.7353	49.9825	56.1458	0.0000
incremental (250, 70/30)	8.2500	63.7750	57.3850	47.6536	51.02346	3.7500
incremental (50, 90/10)	10.3906	7.3333	90.4543	170.1558	174.4324	0.0000
incremental (100, 90/10)	10.5625	15.6666	63.2905	154.6329	156.7749	0.0000
incremental (250, 90/10)	5.1484	152.3333	23.7320	12.8070	14.6370	66.6666

Table 6.5.: The evaluation metrics of the experiments using architecture **1** on the respective **validation sets**. The numbers in parentheses are the number of images, followed by the training - validation set split, if specified.



(a) Training losses.

(b) Validation losses.

(c) Training losses.

(d) Validation losses.

Figure 6.5.: Training and validation losses of the experiments of training architecture **1** from scratch and incrementally. The keys are the number of images used in training. *inc* stands for incremental training and the numbers afterwards indicate the split of the training and validation dataset.

taken from this set at regular intervals, to ensure an optimal distribution. Fig. 6.5 shows the losses of the incremental training runs using architecture 1 in comparison to the runs from scratch. Due to the similarity to the loss graphs of architecture 1, the loss graphs of the experiments of architecture 5 are listed in the appendix in Fig. C.3.

Although the training losses reach a similar low in all experiments, the validation losses diverge by a large margin. No incremental training run managed to reduce the validation loss below the competing losses - not even to the same level. An observation that can be made is that incremental training does not need as many epochs as shown in Fig. 6.5b, since the graphs in Fig. 6.5d reach similar error rates after fewer epochs.

The results of the evaluation metrics of the different experiments are given in Table 6.5. The metrics were created using the validation set of the respective experiment. The same table is provided for architecture 5 in the appendix (Table C.1). The table shows that splitting the

Method	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
from scratch (250, 70/30)	13.0781	**21**	**47.2472**	**69.0290**	**70.5109**	**1.7854**
incremental (250, 70/30)	14.0937	9	81.8304	120.5959	121.1930	0.3968
incremental (250, 90/10)	**12.9375**	13	59.70111	91.1354	93.2678	0.4501

Table 6.6.: The evaluation metrics on **test scene 7** of the incremental training experiments of architecture **1**.

images into 90% training data and 10% validation data results in a network that has a higher accuracy but only for architecture 1. Architecture 5 seems to profit from the larger validation set, according to the table in the appendix. But neither approach can compete with training from scratch which, depending on the metric, outperforms incremental training by a factor of 3 to 15 for both architectures. The numbers of Table 6.5 are also reflected in Table 6.6, which provides the results of the metrics of test scene 7. Training from scratch achieves the best results, incremental training using a 90/10 split the second best closely followed by incremental training using a 70/30 split.

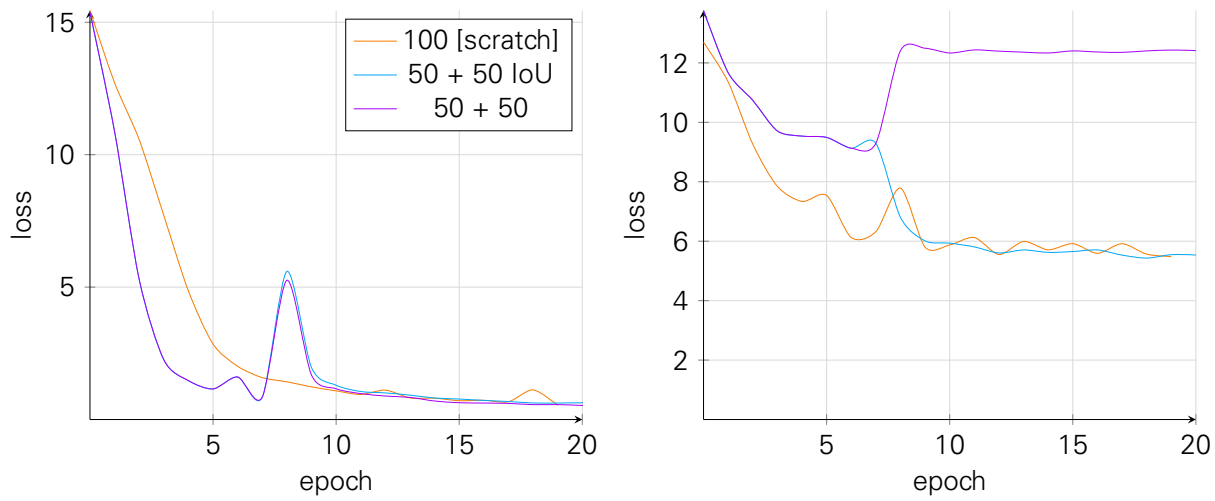
6.5.5. ACTIVE LEARNING

Section 5.5.2 described the possibility of using the intersection over union between segmentation masks resulting from predicted poses and the masks provided with the dataset as a measure of the accuracy of the network. We used the training experiment from the previous section, in which we trained the network from scratch using 50 images. Afterwards we ran network inference for all remaining, i.e. on 1246, images. For these images we rendered the segmentation masks using the predicted poses and computed the IoU with the ground-truth segmentation masks. In the case of the Endoscopic Vision Challenge, we are provided with these masks, in the case of the T-Less dataset we rendered them ourselves.

We selected the 50 images with the lowest IoU and used them for further training. We used 50 instead of 25 images to incrementally train the network to verify simultaneously if the problems described in the previous section can be compensated this way. The losses are depicted in Fig. 6.6. In contrast to training with the 50 images having the smallest IoU, training the network that was trained on 50 images already with another 50 images (these are exactly the images the 100 [scratch] training was performed on) lead to overfitting of the network. This is visible in the purple loss curve in Fig. 6.6b. Table 6.7 shows the metrics on the respective validation sets of the experiments, while Tabel 6.8 shows the metrics on test scene 7. Using

Method	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
100 [scratch]	**4.3750**	**219.6066**	**15.8770**	**21.2478**	**21.9831**	**37.3333**
50 + 50 IoU	4.7070	292.0625	21.7829	21.1833	22.8974	12.5000
50 + 50	9.0390	28.1333	53.1163	29.03213	33.7456	6.6666

Table 6.7.: Evaluation metrics of the active learning experiments on the respective **validation sets**. 50 + 50 IoU denotes the experiment of training the network incrementally with 50 images having the smallest IoU with the ground-truth segmentation masks. 50 + 50 stands for the experiment of incrementally training the network with another 50 images.



(a) Training losses.

(b) Validation losses.

Figure 6.6.: Training and validation losses of the active training method compared to training from scratch. The spike of the blue and purple error curves in the left graph are located where the training using the new images begins.

Method	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inlier %
100 [scratch]	**13.9687**	**24.9087**	**76.2622**	**203.8421**	**206.4120**	0.0099
50 + 50 IoU	14.1640	15.77976	93.2536	249.6441	253.2160	0.0000
50 + 50	14.6718	11.1448	91.4197	246.9521	250.6450	**0.3968**

Table 6.8.: The corresponding evaluation metrics on the **test scene 7** of the T-Less dataset.

the 50 images with the smallest IoU does yield a better trained network, although not in a scale we expected from the loss graphs. We assume that the inlier rate is not as meaningful as in the other experiments, due to the low values.

6.5.6. RUNTIME ANALYSIS

The approximate times of running the different architectures in inference mode on a single image can be seen in Table 6.9. Loading the weights is amortized in these times already, i.e. inference was run on many images, otherwise it takes a lot longer to process a single image. The actual pose recovery computation took 26 ms per image. With a minimum total of 41 ms (architecture 1) and a maximum total of 105 ms (architecture 5), this system can be utilized for near-realtime pose estimation. All benchmarks were achieved by a Intel Core i7-4770 CPU running at 3.40GHz and a Nvidia Titan X grahpics card. The authors of [18] report a similar runtime of 150 ms (for pose estimation and refinement). Brachmann et al. state in [3] that their

Architecture	1	2	3	4	5	6
Inference Runtime	15 ms	47 ms	21 ms	64 ms	79 ms	66 ms

Table 6.9.: Inference runtimes of the network architectures on a single image. These times can only be achieved when loading the weights has been amortized by running inference on many images.

pose estimation pipeline runs for about 550 ms but their approach uses depth in contrast to our work. Pertsch's RGB-only version needs 770 ms to recover a pose, according to [28].

7. DISCUSSION

# Object \ Image	0000.jpg	0150.jpg	0250.jpg	0350.jpg	0500.jpg
01	0:53	2:29	0:20	0:03	0:06

Table 7.1.: The table shows the **absolute difference** between in minutes between correcting a pose using the controls of 6D-PAT and the new rotation feature. The comparison was only performed exemplary for object model 01 due to the limited time-frame of this work.

Before discussing the outcomes of this work, we want to state a general remark. Our goal of training the network on the Endoscopic Vision Challenge dataset was not achievable, as we were not provided with the correct 3D object models. It is difficult to tell, how the networks would have performed on this dataset, as it appears to be more challenging in terms of image quality, occlusion, etc.

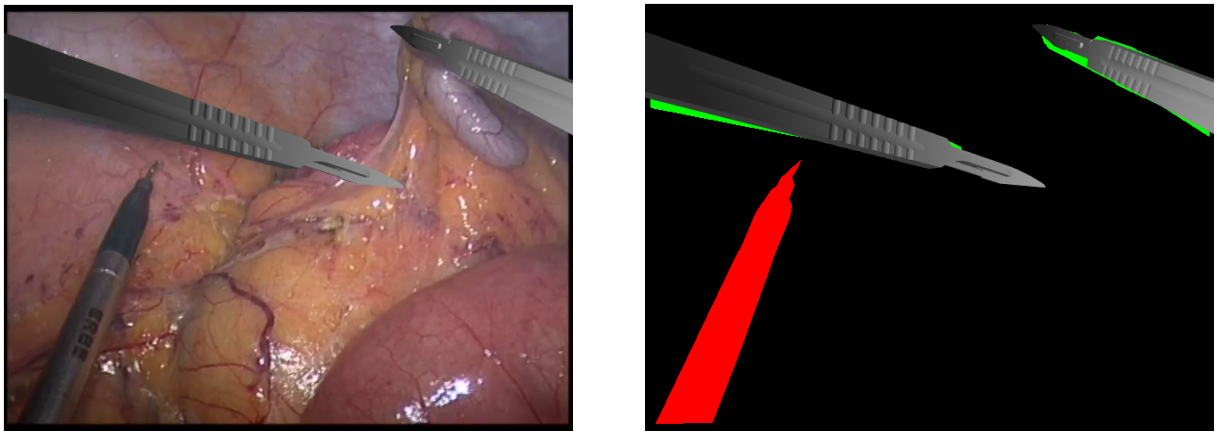
7.1. MANUAL ANNOTATION PROCESS

The process of manually recovering poses using 6D-PAT comes along with some peculiarities, which we discuss here. By creating some annotations ourselves, we figured out that one of the bottlenecks of the program is the correction of poses after the initial position and rotation has been recovered from the clicked correspondences. The reason is that adjusting the values of the rotation and position by hand using the provided controls is slow and a try-and-error process. By trying to enter a higher rotational degree, the correct value was often surpassed.

Despite the limited time of this work, we decided finally to improve the correction procedure by implementing functionality that allows easy rotation of the object model of a pose using the mouse. To make use of this new feature, the user has to select a pose to edit from the drop down list of poses and rotate the object model that is displayed by the pose editor (not pose viewer). This concept allowed the author to achieve the difference in times presented in Table 7.1. The discrepancy was computed only for the time needed to correct a pose, i.e. the time needed for clicking correspondences was not considered. This is not an empirical study and the superiority of the rotation method compared to manually editing the numbers needs to be proven for many images and datasets. But the provided times can serve as an indicator that directly rotating the object can be faster.

Next to 6D-PAT, we also faced some issues regarding the datasets themselves. Initially, a desired feature was to present another unannotated image after the user finishes annotating all poses in the current image. This is not feasible for multiple reasons. First of all, a user might still not be content with the poses. Selecting the next image programmatically could happen too early and thereby disrupt the annotation process of the user. The datasets can also have very distinct characteristics, which require the user to choose the next image manually. For a dataset like T-Less, many images have to be skipped due to their similarity. A network trained on one view on an object only, will likely predict inaccurate poses for a different view showing other characteristics of that object. The dataset of the Endoscopic Vision Challenge has less similar consecutive images.

The diversity of available datasets (see Section 6.2 for a selection of datasets) is also the



(a) An image from the medical images dataset annotated using 6D-PAT. The correct rotation and translation are difficult to estimate without the correct 3D models.

(b) The corresponding segmentation image. The discrepancy between the segmentation masks and the object model is the clearly visible green area.

Figure 7.1.: Example annotations of images of the Endoscopic Vision Challenge dataset. The object model is taken from [64].

reason why the program does not provide functionality to initialize the next poses based on the ones in the last image. This feature would imply too many assumptions on the dataset and, in the worst case, cause more work for the user.

While trying to annotate the images of the Endoscopic Vision Challenge dataset, it became clear that proper 3D models are crucial for successful annotation. To temporarily annotate the medical images, 3D surgical tools were downloaded from the internet as a replacement for the missing object models. But having a different shape and also missing the distinctive features of the real objects makes it very difficult to estimate the ground-truth pose. The influence of contradicting annotated poses during training of a network is not clear. The author of this work strongly recommends to obtain the correct 3D models before annotating the Endoscopic Vision Challenge images. The issue of the object models not fitting the segmentation mask can be seen in Fig. 7.1b. Fig. 7.1a shows the actual image and the discrepancy between the object models and image pixels.

A general problem of any dataset can be seen in Appendix A in form of the rotational error of manually annotated images. If the offset is similar in all recovered poses, this might not cause any trouble. If not, a network might be trained with contradicting object coordinates. There is no simple solution to this problem. Distinct characteristics of the object models could reduce this effect significantly, like the letters on the surgical tools in the Endoscopic Vision Challenge dataset or the notch of object model 01 of the T-Less dataset.

We think that 6D-PAT already resembles a program that can be used to annotate pose estimation datasets, efficiently and effectively already. Nevertheless, we provide improvement suggestions in Chapter 8.

7.2. SEMI-AUTOMATIC ANNOTATION PROCESS

This section discusses the results of the experiments that we conducted using the network architectures of Chapter 5. To this end, we analyze the outcomes in the order they appeared in Chapter 6. First, we want to make clear that some of the ways how we inferred conclusions underly some restrictions and are owed to the limited time of this work. An incremental training approach might have worked better for architectures 2 and 3, for example. We do not assume that this impacted the experiment results by a great extend but it has to be kept in mind when using the results for further research. We now state our interpretations of the training outcomes.

We conclude that the reduction of the receptive field-size without increasing the network's depth leads to a higher training and validation error. This can be seen in the discrepancy between the validation loss curves of architecture 1 and 3. Architecture 3 is as deep as architecture 1 (both 23 layers) but a reduced receptive field-size and a validation loss four times as high. Architecture 2 compensates the increased error related to the field-size by deeper design (35 layers). Architecture 5 achieves an error as low as architecture 1. Although its receptive field-size is smaller than that of architecture 2 (59 opposed to 67), we increased the number of layers to 50. Intuitively, by seeing less data (smaller receptive field-size) the network has to compensate this by being able to store more information somehow, thus the need for more layers.

It appears like all architectures memorize the training data quite well and achieve high pose inlier rates. But this does not hold for the metrics on the test set, although the test scenes are similar in terms of lightning conditions and image quality. We are unsure how the large drop of the performance of the networks on the test scene images compared to the training images came to exist.

Rad et al. report an average pose inlier rate of all objects visible in the test scene of 53.5 in [18]. To the best of our knowledge, this is the only work providing an inlier rate on the T-Less dataset without using depth. This number is significantly higher than the accuracy we achieved. We expected even better results from our networks because the location of the object is already defined by the segmentation mask. Inspecting the metrics implies that the pose error is closely related to the error that captures the euclidean distance between the ground-truth and inferred translation vector. Since we predict poses on RGB-only images without depth information this is not too surprising. But this needs further investigation whether the z-coordinate is the one that causes the large distance errors.

Our initial assumption was that the networks are too deep and overfitted towards the data they were trained on. Despite the similar lightning conditions of the test images, we thought that the networks had taken in too many details of the training images. Since the validation images were taken from the pool of available training images, overfitting to the conditions of the training images can not be entirely prevented by such a validation set. But architecture 5 achieves the best pose inlier rate on the test set. Thus, it might be that the architectures are actually too shallow. Unfortunately, we were not able to explore this in depth in favor of examining the different training strategies.

The overall appearance of the validation losses of the incremental approaches is difficult to explain. The network seems to overfit but even if the training runs had been stopped in the

valleys visible in Fig. 6.5d, there would be a large gap compared to the training from scratch, still.

We cannot give a final statement why the incremental approach results in such an inferior validation loss. We assume that there is a way to improve this method so that it reaches the same level like the training from scratch. A reason for the inconsistent loss could be the trainable batch normalization. Training the network with 50 images first and then fixing the batch normalization weights could help decrease the validation loss when further training the network with datasets of 25 images. Although we expelled the SGD optimizer at the beginning of the experiments it could be that it is able to provide better results than Adam. We conclude this from the loss graphs of the incremental experiments, as the spiky curves could mean that Adam is not able to properly compute the learning rate. A very small rate set for SGD could yield different results. But for now, we have to recommend to train the network from scratch every time the user has manually annotated another batch of images.

The active training scheme on the other hand did provide better validation losses, i.e. selecting the images with the smallest IoU with the ground-truth segmentation masks reduced the validation loss significantly. This way, the validation loss of the experiment of training the network with 100 images from scratch was reached. Training with 50 of these images first and then incrementally training the network with the remaining 50 images resulted in a high validation loss. The results on the test set imply that this does not reliably mean that the network performs better in general. This is a sever issue and needs further investigation before drawing any conclusions.

We still suggest using a network together with 6D-PAT. We think that an unannotated dataset like Endoscopic Vision Challenge (EVC) can still profit from the approximate locations predicted by a neural network. Although poses on the test sets seem to be inaccurate, the poses can be accurate when the network is trained on very similar images. For example, training architecture 1 with 250 images yields 78% inlier on the validation set. Since we extracted validation images at regular intervals which ensures an even distribution, we conclude that the precision of the network is similar or only slightly below for the rest of the images. This means that predicted poses on images that the network was not trained with probably only need slight corrections. The fully annotated dataset can then be used to train a new network design that performs better on the test images.

8. FUTURE WORKS

To provide an outlook how the manual annotation process can be improved in the future, we provide some suggestions here. An important feature that should be implemented in 6D-PAT is allowing to move the object models around on the displayed image by dragging and them using the mouse. The initial poses using the clicking procedure are already sufficiently accurate in many cases. But when the camera is looking along an axis of the object, the position can be far off. The user can correct these poses with the provided controls. But the implemented rotation feature indicates that editing a pose by mouse can be faster which probably also holds for the position of an object.

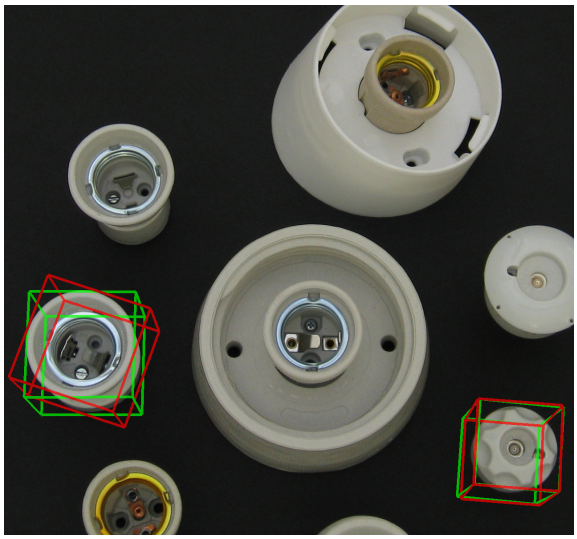
The binding of the neural network to the program does not allow to load the weights of the net and persist this state in the memory of the computer. In case that the user wants to annotate a large dataset of just one object, they could profit from a feature allowing to keep the weights in memory instead of loading them each time the inference process is started. This way, predicting the pose for only one image could be achieved in significantly less time. Running the network to predict the pose of one object in a large amount of images amortizes the loading of the weights.

Another suggested step is to fully incorporate the network in the annotation tool. This way, inexperienced users can profit from the network without the need for an expert to run the network inference. There is probably no way to automatize the setup process, as this requires knowledge neural networks. In an ideal setting, this interference can be reduced to a minimum and users are introduced to the most relevant parameters only, which they can set from within the program.

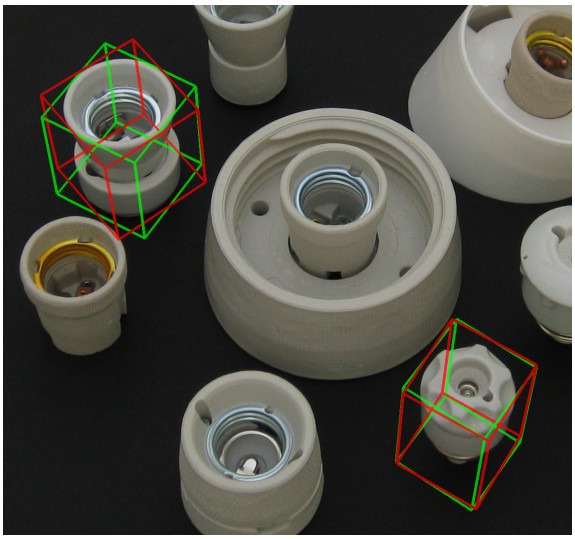
Because the neural network is trained for one object only, the time-intensive step (proportional to the overall time needed for inference on one image) of loading the trained weights has to be performed before predicting poses for different objects. The frame of this work was to explore neural networks trained for one object only, it is therefore not known whether this overhead of loading the weights can be reduced but this could be subject to future research.

Appendices

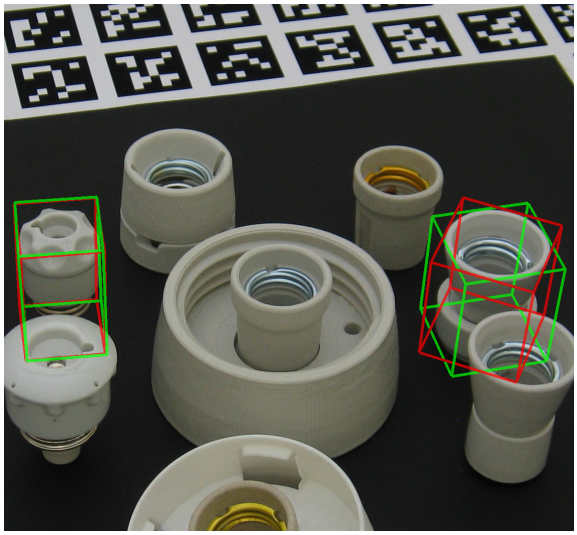
A. EXAMPLE MANUAL ANNOTATIONS



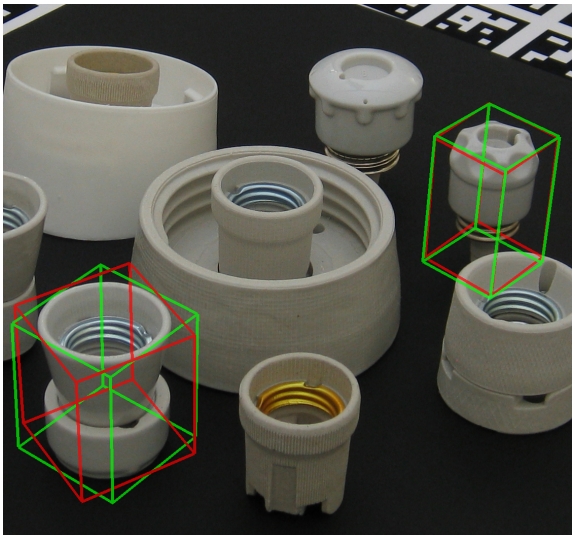
(a) Image 0000.jpg.



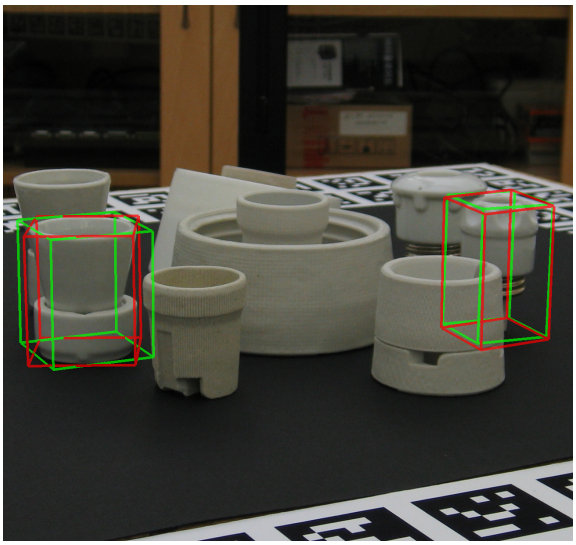
(b) Image 0150.jpg.



(c) Image 0250.jpg.



(d) Image 0350.jpg.



(e) Image 0500.jpg.

Figure A.1.: Example images from test scene 7 of the T-Less dataset with the poses recovered using 6D-PAT. The images are cropped to the relevant area.

B. NETWORK ARCHITECTURES

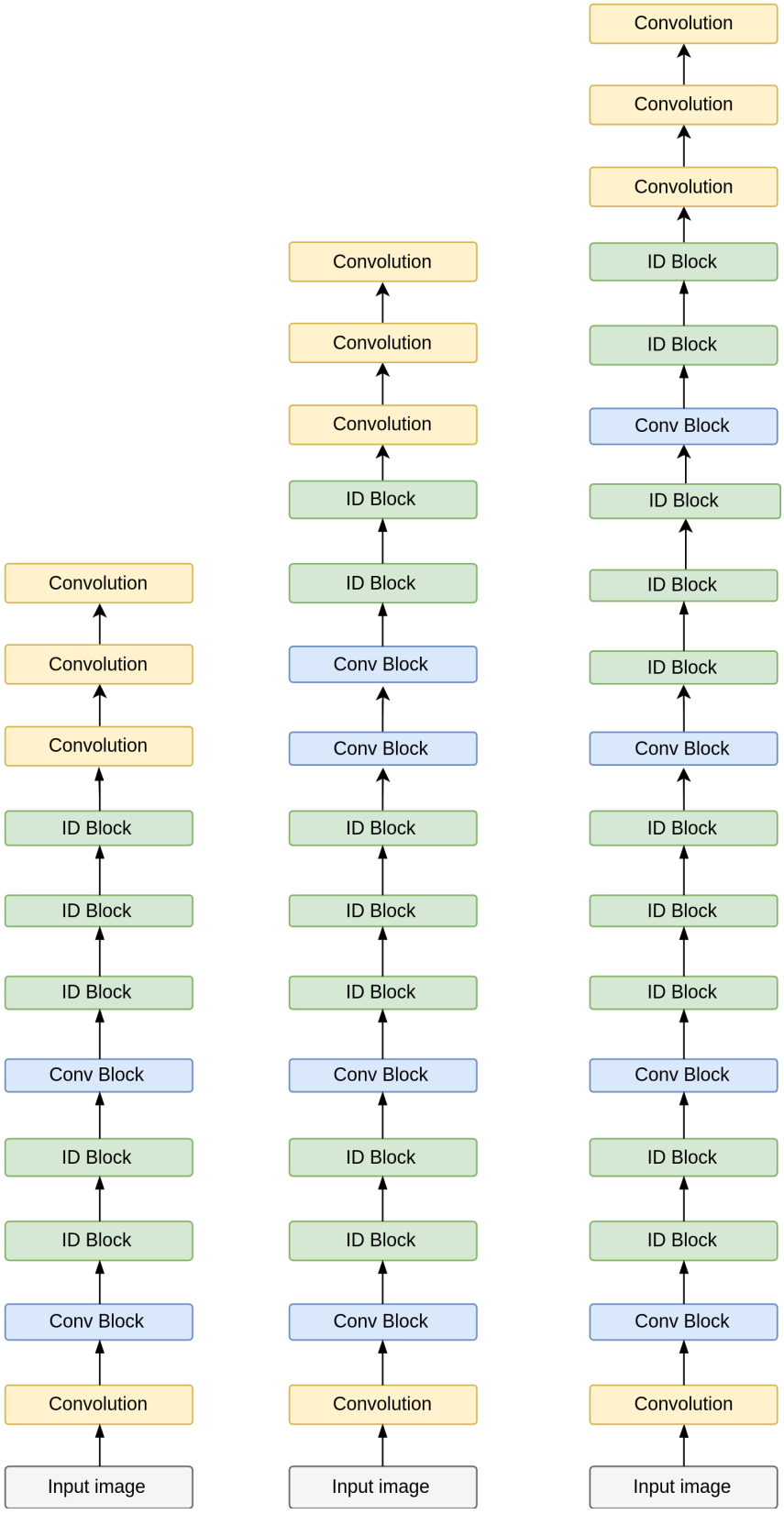
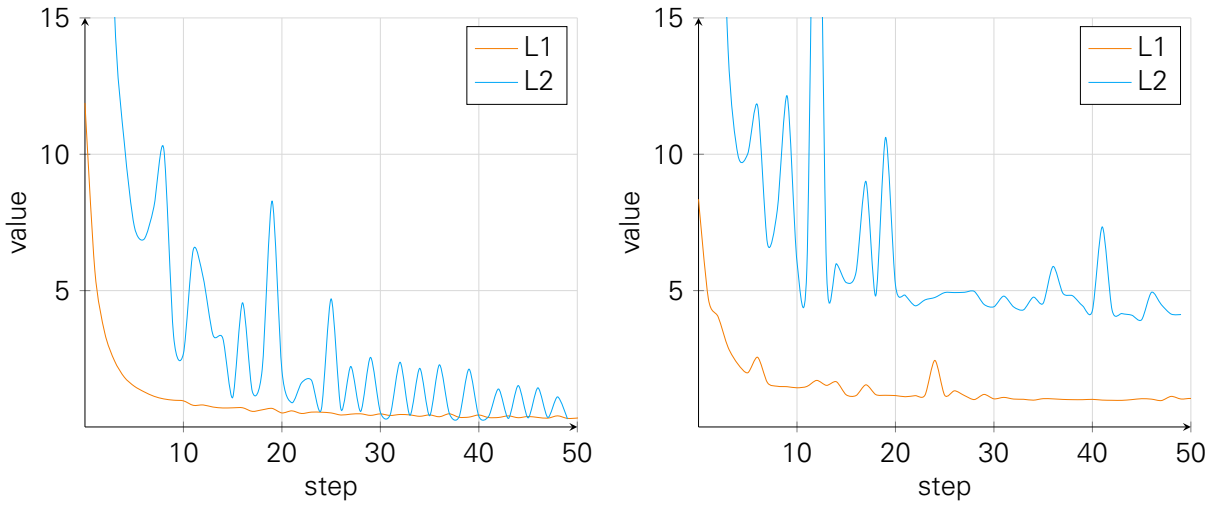


Figure B.1.: The different architectures used in this work. Architectures 1 and 3 use the leftmost structure with a total of 23 convolutional layers. We constructed architecture 2 using the layout in the middle with 35 layers. Architectures 4, 5 and 6 use the 50 layers structured as shown on the right, although architectures 4 and 6 employ dropout layers with different frequencies after each block and omit the batchnormalization layers. We omitted the batchnormalization layers in the other to structures for displaying purposes.

C. EXPERIMENTS



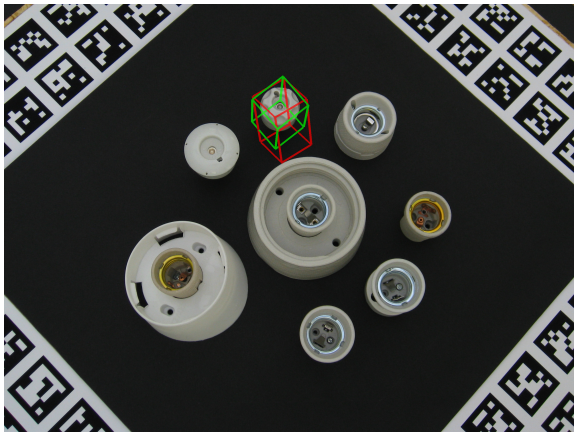
(a) Training losses.

(b) Validation losses.

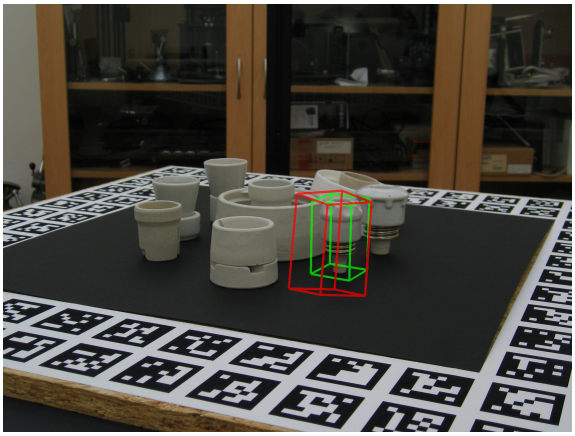
Figure C.1.: Training and validation losses of the L1 and L2 loss of training architecture 1. The plot is cropped on the *y*-axis to enhance the differences.

Method	e_{coord}	n_{inlier}	e_{angle}	e_{dist}	e_{pose}	inliers %
from scratch (25, 70/30)	11.5468	57.8933	84.3733	13.70721	23.0772	1.33333
from scratch (50, 70/30)	9.2500	87.6600	51.2956	18.8137	23.4164	14.0000
from scratch (100, 70/30)	6.5312	151.2266	26.1370	40.6087	41.934	23.3333
from scratch (250, 70/30)	**4.1992**	**332.2933**	**4.7410**	**9.2082**	**9.4344**	**62.0000**
incremental (50, 70/30)	12.6562	10.6250	97.5586	48.7668	56.2073	6.2500
incremental (100, 70/30)	11.3671	27.2500	90.8267	32.6782	38.9909	3.1250
incremental (250, 70/30)	9.3281	68.9375	68.3927	34.8104	39.7651	3.7500
incremental (50, 90/20)	11.5625	5.3333	97.9677	131.0969	135.8001	0.0000
incremental (100, 90/10)	11.9843	3.3333	116.4722	38.5254	46.8544	0.0000
incremental (250, 90/10)	12.7500	15.3333	101.6921	107.6326	113.3697	0.0000

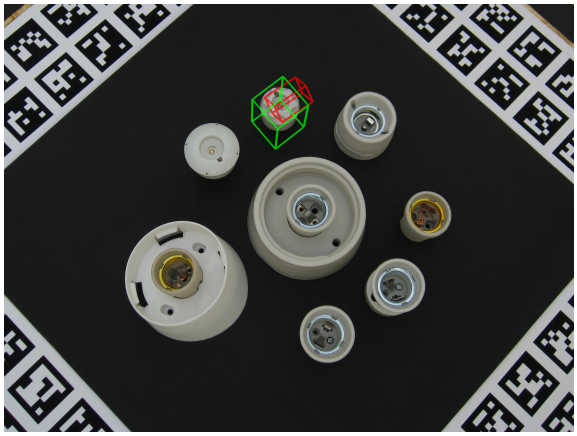
Table C.1.: The evaluation metrics set of the experiments using architecture **5** on the validation set. The numbers in parentheses are the number of images, followed by the training - validation set split, if specified.



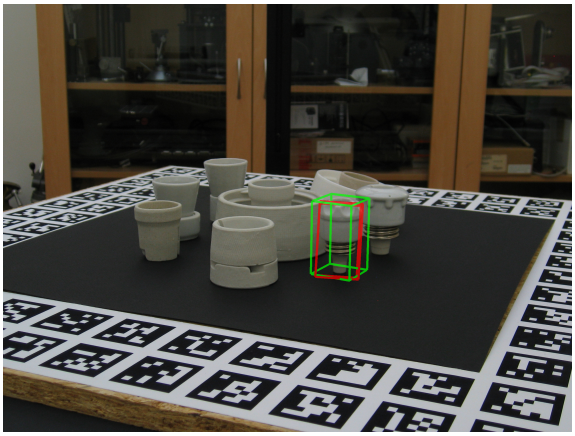
(a) Image 0045.jpg, pose recovered using architecture 2.



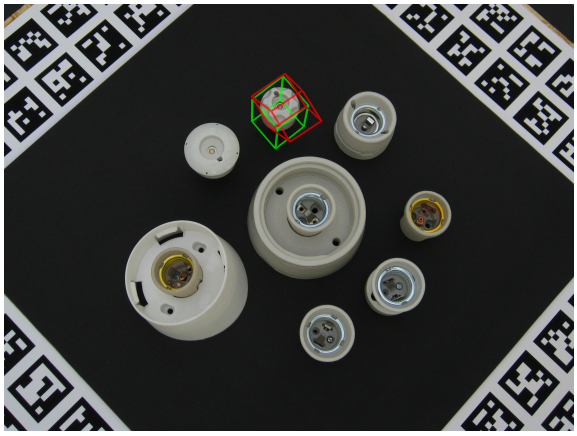
(b) Image 0438.jpg, pose recovered using architecture 2.



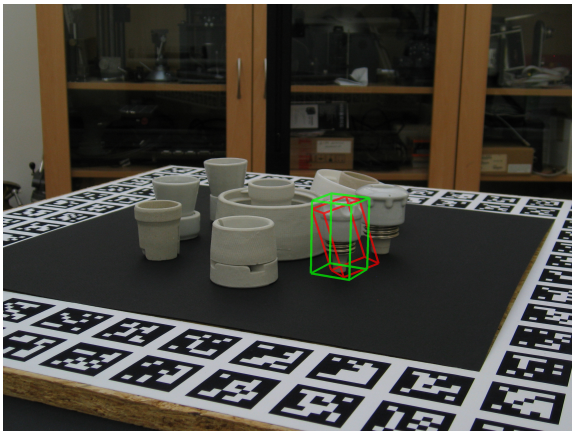
(c) Image 0045.jpg, pose recovered using architecture 3.



(d) Image 0438.jpg, pose recovered using architecture 3.

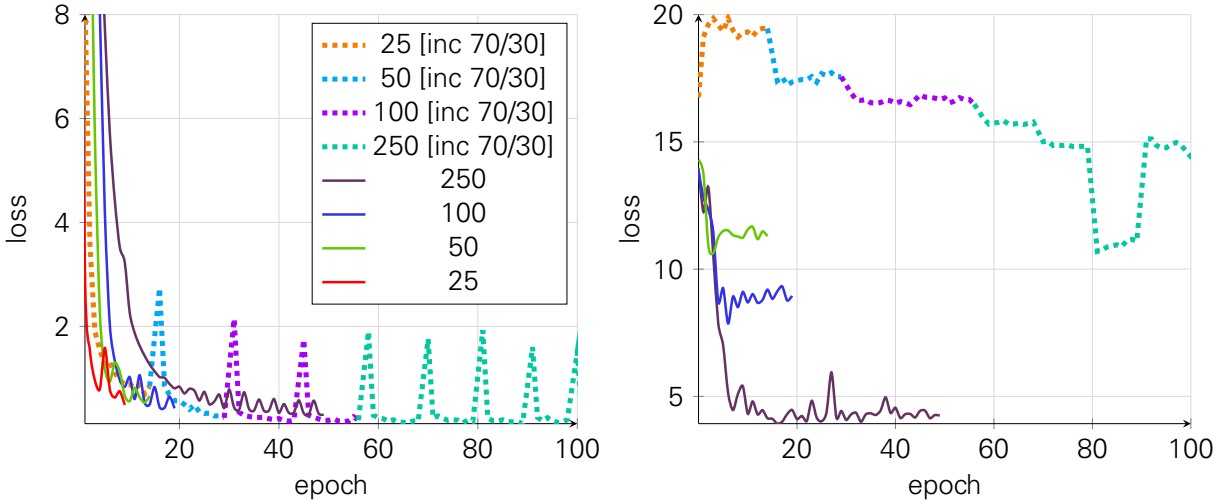


(e) Image 0045.jpg, pose recovered using architecture 5.



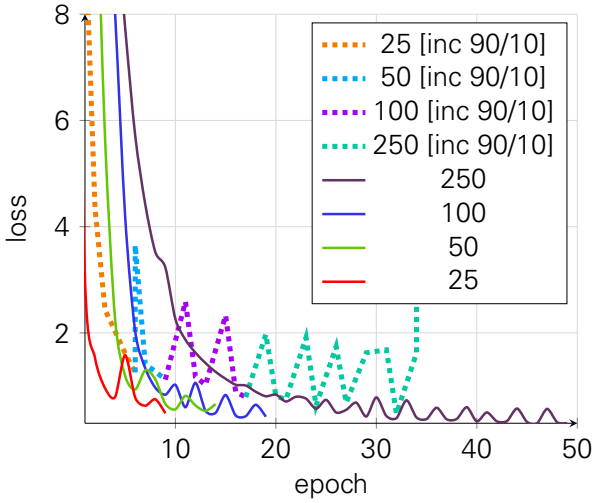
(f) Image 0438.jpg, pose recovered using architecture 5.

Figure C.2.: Example images from test scene 7 with rendered poses recovered by the architecture mentioned in the image caption.

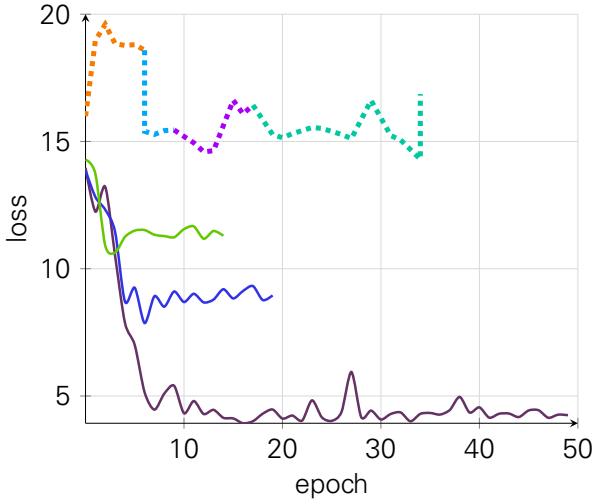


(a) Training losses.

(b) Validation losses.



(c) Training losses.



(d) Validation losses.

Figure C.3.: Training and validation losses of the experiments of training architecture **5**. The keys are the number of images used in training. *inc* stands for incremental training and the numbers afterwards indicate the split of the training and validation dataset.

ACRONYMS

6-DoF 6 Degrees of Freedom

CNN Convolutional Neural Network

EVC Endoscopic Vision Challenge

MVC Model-View-Controller

PnP Perspective-n-Point

RANSAC Random Sample Consensus

SGD Stochastic Gradient Descent

UI User Interface

LIST OF FIGURES

LIST OF TABLES

9. BIBLIOGRAPHY

[1] J. P. M. Rosen, "Minimally invasive surgery," vol. 33, no. 4, pp. 358–366, 2001. [Online]. Available: https://www.thieme-connect.com/products/ejournals/html/10.1055/s-2001-13689#N66875

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[3] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, *Learning 6D Object Pose Estimation Using 3D Object Coordinates*. Cham: Springer International Publishing, 2014, pp. 536–551. [Online]. Available: https://doi.org/10.1007/978-3-319-10605-2_35

[4] A. Krull, E. Brachmann, F. Michel, M. Y. Yang, S. Gumhold, and C. Rother, "Learning analysis-by-synthesis for 6d pose estimation in RGB-D images," *CoRR*, vol. abs/1508.04546, 2015. [Online]. Available: http://arxiv.org/abs/1508.04546

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, insight. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[7] Consortium for Open Medical Image Computing, "Endoscopic vision challenge: Instrument segmentation and tracking," 2015.

[8] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision*, Sept 2009, pp. 2146–2153.

[9] W. Commons. (2015) Conv layer. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Conv_layer.png

[10] ——. (2015) Max pooling. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Max_pooling.png

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[14] T. Sharp, "The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 103–110. [Online]. Available: http://dl.acm.org/citation.cfm?id=2354409.2354668

[15] T. Hodaň, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[16] G. Bradski, "Opencv: Real time pose estimation of a textured object," Jul 2018. [Online]. Available: https://docs.opencv.org/trunk/dc/d2c/tutorial_real_time_pose.html

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[18] M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," *CoRR*, vol. abs/1703.10896, 2017. [Online]. Available: http://arxiv.org/abs/1703.10896

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[20] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," in *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Sept 2008, pp. 125–134.

[21] G. Klein and D. W. Murray, "Full-3d edge tracking with a particle filter," in *BMVC*, 2006.

[22] D. G. Lowe, "Fitting parameterized three-dimensional models to images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 5, pp. 441–450, May 1991.

[23] C. Harris and C. Stennett, "Rapid - a video rate object tracker," in *Proceedings of the British Machine Vision Conference*. BMVA Press, 1990, pp. 15.1–15.6, doi:10.5244/C.4.15.

[24] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit, "Gradient response maps for real-time detection of textureless objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 5, pp. 876–888, May 2012.

[25] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," in *Proceedings of the 11th Asian Conference on Computer Vision - Volume Part I*, ser. ACCV'12. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 548–562. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37331-2_42

[26] R. Rios-Cabrera and T. Tuytelaars, "Discriminatively trained templates for 3d object detection: A real time scalable approach," in *2013 IEEE International Conference on Computer Vision*, Dec 2013, pp. 2048–2055.

[27] C. Steger, *Similarity Measures for Occlusion, Clutter, and Illumination Invariant Object Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 148–154. [Online]. Available: https://doi.org/10.1007/3-540-45404-7_20

[28] K. Pertsch, "Improving 6d pose estimation by predicting accurate instance segmentation masks," 2017.

[29] Z. Zhou, J. Shin, L. Zhang, S. Gurudu, M. Gotway, and J. Liang, "Fine-tuning convolutional neural networks for biomedical image analysis: Actively and incrementally," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[30] K. Pauwels, L. Rubio, J. Díaz, and E. Ros, "Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, June 2013, pp. 2347–2354.

[31] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3d object recognition," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 998–1005.

[32] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects." in *CVPR*. IEEE Computer Society, 2013, pp. 1352–1359. [Online]. Available: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2013.html#Salas-MorenoNSKD13

[33] K. Lai, L. Bo, X. Ren, and D. Fox, "A scalable tree-based approach for joint object and pose recognition," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11. AAAI Press, 2011, pp. 1474–1480. [Online]. Available: http://dl.acm.org/citation.cfm?id=2900423.2900656

[34] D. Tomè, C. Russell, and L. Agapito, "Lifting from the deep: Convolutional 3d pose estimation from a single image," *CoRR*, vol. abs/1701.00295, 2017. [Online]. Available: http://arxiv.org/abs/1701.00295

[35] A. Zeng, K. Yu, S. Song, D. Suo, E. W. Jr., A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge," *CoRR*, vol. abs/1609.09475, 2016. [Online]. Available: http://arxiv.org/abs/1609.09475

[36] Amazon.com, "Amazon picking challenge," 2016. [Online]. Available: https://www.amazonrobotics.com/#/roboticschallenge

[37] A. Kendall, M. Grimes, and R. Cipolla, "Convolutional networks for real-time 6-dof camera relocalization," *CoRR*, vol. abs/1505.07427, 2015. [Online]. Available: http://arxiv.org/abs/1505.07427

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[40] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[41] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[42] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[43] T. Kurmann, P. Marquez Neila, X. Du, P. Fua, D. Stoyanov, S. Wolf, and R. Sznitman, *Simultaneous Recognition and Pose Estimation of Instruments in Minimally Invasive Surgery*. Cham: Springer International Publishing, 2017, pp. 505–513. [Online]. Available: https://doi.org/10.1007/978-3-319-66185-8_57

[44] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[45] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, "Scene coordinate regression forests for camera relocalization in rgb-d images," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, June 2013, pp. 2930–2937.

[46] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3364–3372.

[47] A. Krull, E. Brachmann, S. Nowozin, F. Michel, J. Shotton, and C. Rother, "Poseagent: Budget-constrained 6d object pose estimation via reinforcement learning," *CoRR*, vol. abs/1612.03779, 2016. [Online]. Available: http://arxiv.org/abs/1612.03779

[48] D. Massiceti, A. Krull, E. Brachmann, C. Rother, and P. H. S. Torr, "Random forests versus neural networks - what's best for camera relocalization?" *CoRR*, vol. abs/1609.05797, 2016. [Online]. Available: http://arxiv.org/abs/1609.05797

[49] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009. [Online]. Available: http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf

[50] D. Wang and Y. Shang, "A new active labeling method for deep learning," in *2014 International Joint Conference on Neural Networks (IJCNN)*, July 2014, pp. 112–119.

[51] J. Li, "Active learning for hyperspectral image classification with a stacked autoencoders based neural network," in *2015 7th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, June 2015, pp. 1–4.

[52] B. Liu and V. Ferrari, "Active learning for human pose estimation," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[53] The Qt Company, "Qt." [Online]. Available: https://www.qt.io/

[54] ——, "The qt signals and slots mechanism," 2018, [Online; accessed July 8, 2018]. [Online]. Available: http://doc.qt.io/qt-5/images/abstract-connections.png

[55] P. Brian et al., "Mesa 3d grahpics library." [Online]. Available: https://mesa3d.org/

[56] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[57] A. Gessler, T. Schulze, and K. Kulling, "Assimp." [Online]. Available: http://www.assimp.org/

[58] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[59] F. Chollet *et al.* , "Keras," https://keras.io, 2015.

[60] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[61] A. Doumanoglou, R. Kouskouridas, S. Malassiotis, and T. Kim, "6d object detection and next-best-view prediction in the crowd," *CoRR*, vol. abs/1512.07506, 2015. [Online]. Available: http://arxiv.org/abs/1512.07506

[62] C. Rennie, R. Shome, K. E. Bekris, and A. Ferreira De Souza, "A dataset for improved rgbd-based object detection and pose estimation for warehouse pick-and-place," *IEEE Robotics and Automation Letters (RA-L) [Also accepted to appear at the 2016 IEEE International Conference on Robotics and Automation (ICRA)]*, vol. 1, pp. 1179 – 1185, 02/2016 2016. [Online]. Available: http://www.cs.rutgers.edu/~kb572/pubs/icra16_pose_estimation.pdf

[63] K. Pauwels, L. Rubio, J. Diaz Alonso, and E. Ros, "Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Portland, 2013, pp. 2347–2354.

[64] S. Schwecke, "Scalpel," https://grabcad.com/library/35682, 2012, [Online; accessed July 25, 2018].

Statement of authorship

I hereby certify that I have authored this Großer Beleg entitled *Annotation of Laparoscopic Surgery Images with Online Proposal Generation* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, August 14, 2018

Florian Blume