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

Großer Beleg

Referee
Eric Brachmann

Submitted on: July 18, 2018
Defended on: Not yet

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 and less recovery time needed afterwards [1]. This kind of operation is more involved for the surgeon, as it is conducted only through a small hole in the otherwise closed abdominal wall. 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 during the surgery. Next to navigation cues and other vital information, the surgical tools could be rendered into the endoscopic video to compensate the missing depth information to some extend and facilitate the operational process.

The task of annotating images with 3D objects is called 6D pose estimation and is already used and applied in various fields, e.g. robotics, augmented reality, medical imaging, and many more. For well-textured clearly visible objects the task is considered solved. Methods like Lowe's [2] 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, depth-based pose estimation procedures were able to achieve good results on texture-less but unoccluded and non-deformed objects. Many of those methods match the image against a database of templates to recover the pose.

These techniques have in common that they are non-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 and object coordinates to achieve good results on the occlusion dataset [3]. The forests are trained to output the 3D coordinate on the object for every pixel. Using the 2D-3D correspondences, a robust estimate of the object's pose can be computed. With the breakthrough of neural networks in 2012 [4], a lot of research in the area of learning-based methods has been spawned. The so called *Convolutional Neural Networks (CNNs)* offer outstanding accuracy that beat most traditional computer vision algorithms [5]. Krull et al. [6] combine the idea of learning the 3D coordinates representation of an object with the power of CNNs. A major drawbacks of neural networks is the need for training data. In contrast to earlier methods, the data needs to be present upfront and annotated with the desired output.

The goal of this work is to analyze the process of manual 6D pose estimation and develop a tool based on the carved out requirements. The tool is supposed to enable annotating the medical images, similar to the ones provided with the project.

To support the user in the still tedious and time-consuming task of annotating images, a neural network is manufactured for estimating the objects' poses. The base architecture of the network is *ResNet* [7]. ResNet is a network presented in 2015, which allows for deeper networks by adding the input to intermediate layers and still achieves state-of-the-arts results. Similarily to Brachmann et al., the network predicts object coordinates. But since segmentation images are already provided by the medical data, the focus is only on coordinate prediction and not instance segmentation. The network works on RGB-only images because videos from endoscopes often lack depth information.

The remainder of the work is structured as follows: Chapter 2 explains the basic concepts of deep learning and pose estimation. Chapter 3 then introduces and discusses the latest research in the area of pose estimation, after giving a brief overview over earlier methods. The process of manually annotating images with 6D poses of 3D models with the aid of the developed tool, is laid out in chapter 4. Chapter 5 describes the workflow of annotating images with the aid of the neural network. The datasets used to train the network as well as the respective experiments are presented in chapter 6. Last but not least, chapter 7 summarizes this work, draws conclusions and gives a prospect into possible future research.

2. Background

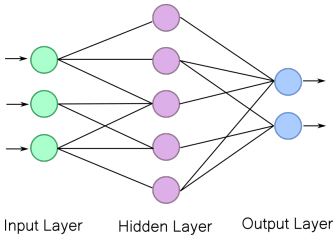


Figure 2.1.: Abstract structure of a feed-forward neural network. A real network will have many more layers and a lot more neurons per layer. Own figure.

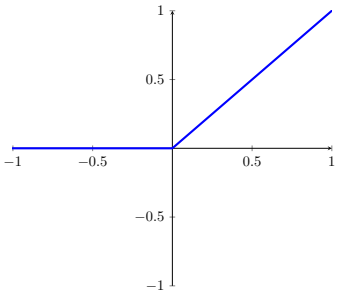


Figure 2.2.: Visualization of the ReLU activation function. Own figure.

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 unit of a neural network and is essentially what the network is made up of. It computes the linear function

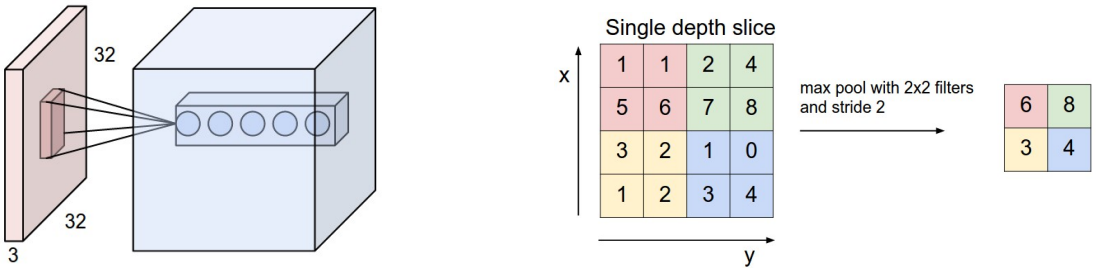
$$y = \sum_{i=0}^{k} w_i x_i + b \tag{2.1}$$

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 neurons of a 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, although 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 one called *Rectified Linear Unit (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 Network

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 output of the neural network depends on the design implied by the use case. It can be object coordinates like in our case, class 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 [7], thus the name deep neural networks and deep learning. All layers can have different numbers of neurons and connections to previous layers. A schematic overview over the structure of a neural network



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

(b) Example pooling layer with maximum as pooling operation.

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

can be seen in fig. 2.1. A neural network can be seen as the function $y = f(x, \theta)$, where θ are the weights and biases of the neurons and some other parameters an y is the output of the network.

2.1.3. Convolutional Neural Networks

A *Convolutional Neural Network (CNN)* describes a certain type of feed-forward neural network that consists of mainly two types of layers: *Convolutional Layers* and *Pooling Layers*. Normally, one or more convolutional layers are followed by a pooling layer. A convolutional layer convolves its input with a kernel which is moved along the input's dimensions. The kernel weights are shared, i.e. stay the same for the convolution of the whole input. Fig. 2.3a shows the schematic of a convolutional layer. Each filter of the layer extends the output in the third dimension and has its own kernel. To reduce the size of the data a pooling layer can be added after the convolutional layer. This significantly speeds up computation and reduces the memory needed. 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, thus does not perform well on unseen data. Some architectures draw on a *Fully-Connected Layer* as the last layer to perform the actual classification, etc. In such a layer every neuron is connected to every previous output. Convolutional neural networks are often used in computer vision tasks like image classification, instance segmentation and many more.

2.1.4. Network training

When a network is created, usually its weights and biases are initialized to 0 or sampled randomly from a Gaussian distribution. To set the parameters to meaningful values that produce the desired output, the network has to be trained. We will now have a look at different techniques how 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, e.g. by summing up the squared differences of the output and the objective output. The network is then applied to a training example in the so-called forward pass. Next, the loss function is derived with respect to each network weight using the chain rule of calculus. The derivative, the computed and the desired output together yield the change, also called delta, to be applied to the weights of the neuron. This delta is then propagated backwards through the net to calculate the deltas of

(a) The object coordinate representation of a human. Image from [11].

(b) An example object from the T-Less dataset. Image from [12].

(c) And the corresponding rendered 3D object coordinates (scaled for visualization). Own image.

Figure 2.4.: Example 3D coordinate representations.

the layers in between, again using the chain rule. After the deltas for all neurons have been computed, the weights are updated according to an update rule (see next paragraph).

Optimization

There exist many different patterns how to update the network weights after obtaining the detlas. The *Stochastic Gradient Descent (SGD)* multiplies the delta by a learning rate and subtracts it from the weight. To ensure convergence, often the learning rate is decreased over time. A more elaborate method is the *Adaptive Moment Optimization (Adam)* [9] which computes adaptive learning rates for each parameter. Here, an exponentially decaying average over past gradients as well as an exponentially decaying average over the past squared gradients serve to increase the learning rate in case of small gradients and decrease the learning rate in case of large gradients. Both averages are stored for the next iteration. Adam has greatly benefited the training process and proven to decrease computation time by finding local minima faster than SGD. Other optimization procedures exist as well but will not be 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 [10] as any modification that reduces the generalization error but not the training error. Many network architectures regularize the net by adding a penalty term on the weights. Another possibility is to randomly deactivate a portion of the neurons to force the network to compensate the missing information, which is called *Dropout*. *Batchnormalization* helps the network generalize by subtracting the batch mean and batch standard deviation from the output of the previous layer. A batch is a subset of the input data which the network is trained with. This technique is often employed when the underlying machine can't fit the whole dataset in memory and because the network typically trains faster this way. To keep SGD or other optimizers from reversing the covariance shift performed by batchnormalization the two parameters *beta* and *gamma* of such a layer are trainable.

2.2. 6D Pose Estimation

6D pose estimation is a central task in the computer vision community. The goal is to retrieve an object's translation and rotation from an image, relative to the camera. *6*D refers to the *6-Degrees-of-Freedom (6-DoF)*, i.e. the 6 free parameters of the *3*D translation and *3*D rotation. The field of application ranges from medical imaging, robotics, augmented reality, and many

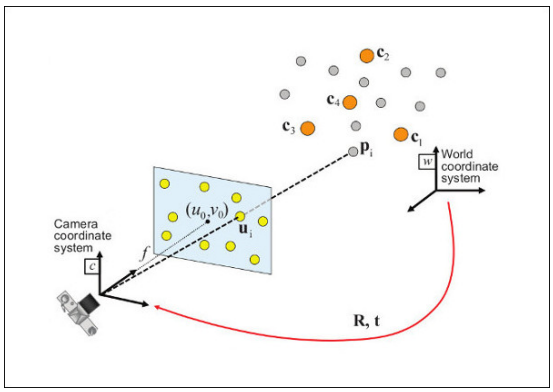


Figure 2.5.: The relationship between the camera, the 3D points and their projections on the screen using the rotation matrix R and the translation vector t. Image from [13].

others. Different systems provide different accuracy. Setups using two cameras for stereo vision or depth information in addition to colors achieve good results. As endoscopes usually do not provide depth information or stereo vision, this work focuses on pose estimation using RGB-only images.

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 directly can be problematic, as the output of the neural network underlies noise and there is no possibility to verify or improve the pose afterwards. This is the reason why this work employs a network with a dense output, i.e. one output for each pixel. The output for a pixel are the 3D coordinates on the object. Each subset of object coordinates leads to a certain pose. Some coordinates may contradict a pose and therefore the pose with the least contradictions is chosen as the best one. The next section will introduce this concept of so-called object coordinates in depth.

2.2.1. Object Coordinate Regression

Tayler et al. first used object coordinates in [11]. Instead of directly predicting the location of joints or limbs of the human body they computed for each pixel the corresponding 3D location on the person (see fig. 2.4a). The idea can be transfered to objects of any kind. Fig. 2.4b shows an example object from the T-Less dataset [12] and fig. 2.4c shows the corresponding rendered 3D object coordinates.

TODO: this is not entirely correct The 3D coordinates and the respective 2D pixel locations on the image yield the *Perspective-n-Point (PnP) Problem*. That is, for a given pixel p on the image and the corresponding 3D point x (both in homogeneous coordinates), a pose consisting of the rotation matrix R and the translation vector t has to fulfill the following equation

$$p = K\ [\ R \mid t\ ]\ x \tag{2.3}$$

where K is the camera matrix

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

which projects the 3D point transformed into the camera coordinate system on the 2D image plane. s is a skew factor which is usually 0, (c_x, c_y) is the principal point, usually the center of the image and f_x and f_y is the focal lengths in x and y direction respectively. The values for those

variables can vary when for example cropping the image. Fig. 2.5 visualizes the relation of pixels and object coordinates. If equation 2.3 is overdetermined because there are more correspondences than variables it cannot be solved directly.

A common method to solve the PnP problem for many correspondences is to use the *RANSAC algorithm* [14]. RANSAC selects a model based on a subset of the dataset and evaluates it against the remaining data. 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 varied. It can for example be the number of inliers. Inliers are 3D points whose reprojected 2D location is within a certain area of the actual 2D location. Since RANSAC is well known and employed in many different applications to estimate a model that fits a dataset best, we use it in this work to compute the pose. The implementation that is used is the one incorporated into the *solvePnPRansac* method of the *OpenCV* [15] framework.

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 to *null*
  for 1...i **do**
    Select a subset of corresponding points to compute pose H
    Compute energy of the pose $e' = E(H)$
    if $e' > e$ **then**
      Store pose H as current best in h
      Set $e = e'$
    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 [16]. 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 [17], 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, 18] 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 [19, 20, 21] 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.

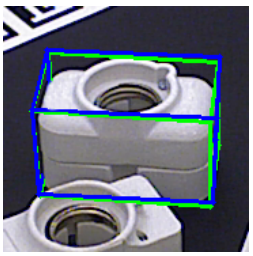
Template-based methods [22, 23, 24, 25] use generated views of discrete viewopints on the object at different angles which are matched against the given image to retrieve the pose. This approach relies mainly on the shape of objects and thus works well for poor-textured or textureless objects [26]. To 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 [27]. 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 [28]. 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 [29, 30] 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.

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 [31], 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 [32]. The system developed in [33] 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 [34]. A prominent example of camera localization, which represents the problem of estimating the camera position in 3D space relative to the scene, is PoseNet [35]. The underlying architecture of the employed neural network is based on GoogleNet, which was introduced in [36]. The CNN estimates the camera position directly



(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 [16].

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 [16], 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 Figure 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 Figure 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 [37] 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.

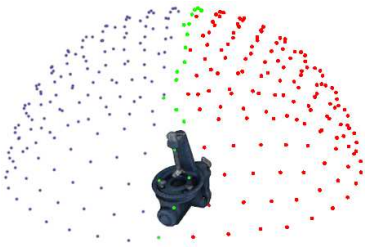
Unfortunately, BB8's take on pose estimation fails on symmetric objects, when implemented directly as described above. To address this problem, the authors first estimate the rotational angle of the object using a neural net, and mirror the image if necessary. This way, a CNN can be trained only on a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

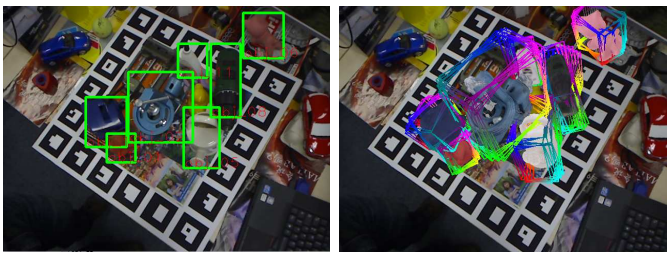
SSD-6D

The system presented in [38] is based on the Single-Shot Multibox Detector (SSD) [39]. 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 Figure 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 Figure 3.2a). The pose



(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 [38]

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 [40]. 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 [16]. The optimization process could also be applied to the two mentioned works. Hence it is not investigated further.

Kurmann et al.

The work of [41] 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 [42], 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 $100ms$, 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 [11] and [43]. 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 [44], 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 [6]. 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 [45]. 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 [46] finds, that random forests partly offer a slightly superior performance, neural networks can compete and are therefore the focus of this work.

Pertsch

Although [26] 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 [42], 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 [44].

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 [26] 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 [47] gives an overview over

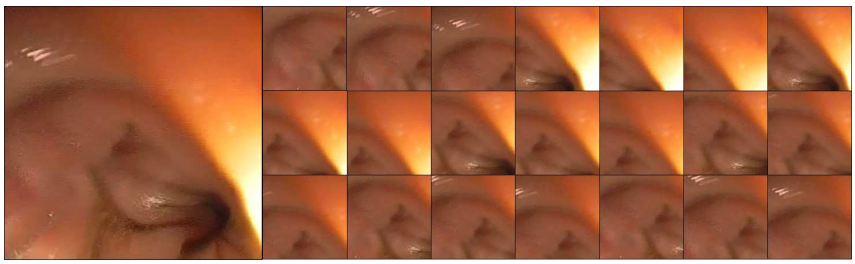


Figure 3.3.: A candidate (left) and the patches generated sharing the same label. Images from [27].

methods developed before 2011. Wang and Shang, the authors of [48], might be among the first to incorporate active learning into deep learning though, according to [27]. In [49], 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 [27], 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 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 Figure 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 [50], 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

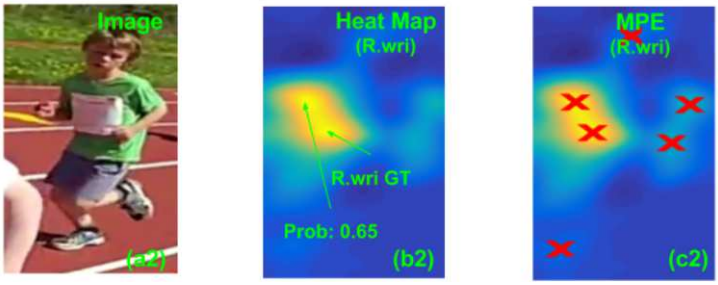


Figure 3.4.: An image of a human body, the corresponding heatmap as well as the result of the Multiple Peak Entropy (MPE). Images from [50].

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 Figure 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 predictions, i.e. predictions with multiple weak peaks (multiple peak entropy) (see Figure 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 [50] 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. Own image.

The following chapter analyzes manual 6D pose annotation process and its prerequisites. To this end, the necessary terminology is defined and explained and the workflow to annotate poses using the developed annotation tool is described. The assessment of the requirements of the annotation procedure and the creation of the tool are conducted based on the medical image dataset, which is described further below.

4.1. Terminology

Image. An image I is a 2D matrix of pixels. The pixel u at position (i, j) is referenced by the tuple (x, y), where $x = j$ and $y = i$. The inverted notation is chosen over the common row-major matrix indexing to account for the universal

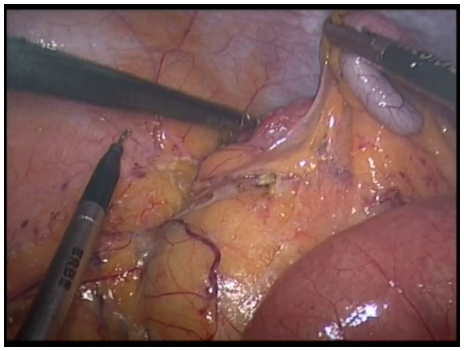
Object Model. An *object model*, or *3D model*, O is composed of a set of points $M \subseteq \mathbb{R}^3$ and a set of triangles $T = \{(m_1, m_2, m_3) \in M^3\}$, also called mesh. The real-world entity that the object model resembles is not restricted. In case of the T-Less dataset [12] the objects are mainly hardware like screws and power sockets.

6D Pose. A *6D pose* P is the tuple (R, t), where R is the 3x3 rotation matrix and t the translation vector used to transform the respective object model into camera coordinates (see section 2.2.1 for details).

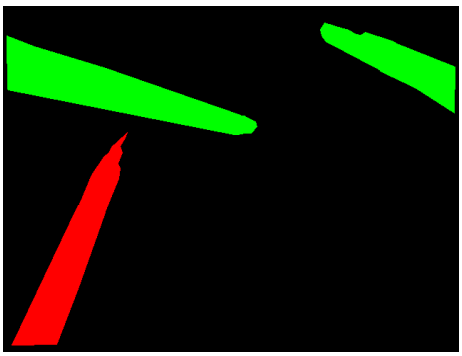
Correspondence. A *correspondence* C is the tuple (u, p), which captures the relation between a pixel u of an image I and an object model O. The pixel u is the projection of the 3D point p onto the image plane using the camera matrix K and a pose P. If the pose is unknown, a set of at least three correspondences can be used to retrieve the pose computationally (see section 2.2.1 for details).

Segmentation Mask. A *segmentation mask* (or *segmentation image*) S for an image I is a second image of the same size. Each position (x, y) of the mask encodes the class of the pixel at (x, y) 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 $s(x, y) = q_i$ for a class set $Q = \{q_0, \cdots, q_n\}$.

Pose Creation. The process of creating correspondences can be performed in various ways. In this work the key operation is to create a set of correspondences and solve the implied perspective-n-point problem.



(a) An example image from the medical dataset. Image from TODO: cite.



(b) The corresponding segmentation mask to the image in fig 4.2a. The colors encode the tools' classes. Taken from TODO: cite.

Figure 4.2.: An example image and its corresponding segmentation mask from the medical dataset.

Ground-Truth. A *ground-truth pose* $\bar{P}$ is a 6D pose. A ground-truth pose is always created 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 on an image I of said object. It is an approximation because there can be a discrepancy between the real world object and its digital 3D representation. Distortions by the camera used to photograph the object and other influences might also not be modeled correctly or not accounted for at all. The rotation and translation error of a ground truth pose are bearable and thus are the objective for neural networks.

Bindings. *Bindings* is the term for a set of code that allows code written in one language being used in a different one.

Makefile. A *makefile* is the directive file used by the program *make* to compile programs of any kind. Often more complex programs imply dependencies which have to be compiled in a certain order. This is an example what make takes care of and what is defined in a makefile.

Training. *Training* stands for the process of training the neural network using gradient-descent (see chapter 2).

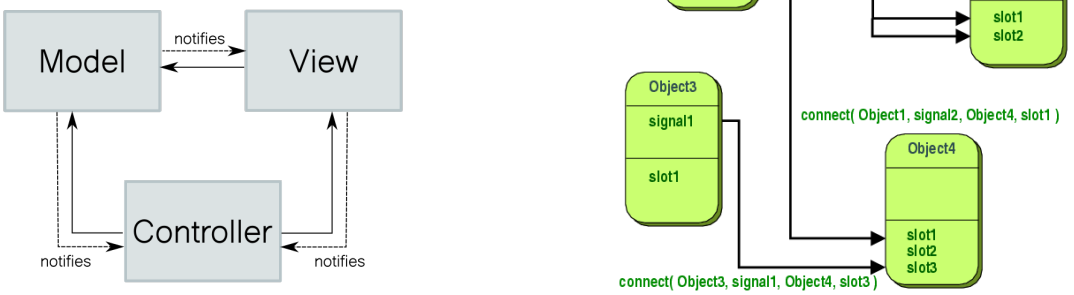
Inference. After training a neural network it can be used to predict poses on unseen images. This is called *inference*.

4.2. Medical Images

The goal of this work is to provide a system to successfully and efficiently annotate the medical images that were provided upfront. The dataset includes segmentation masks but no object models or existing pose annotations. An example image together with the corresponding segmentation mask are given in fig 4.2. The dataset resembles the characteristics and quality of images from laparoscopic videos, i.e. there is no depth information present and occlusion and artifacts like motion blur can occur. The issues with this dataset are discussed in section 4.3.4.

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 programs,



(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 using for example the observer pattern or the Qt Signals and Slots mechanism visible in fig. 4.3b. Own image.

(b) The Signals and Slots mechanism of Qt. A class can define signals it can emit, independently of any listeners waiting for that signal. Slots are the functions that, when connected a signal, are called and can thus react to the signal. 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.

require the annotation personnel to get accustomed to complex 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 medical dataset in particular. The annotated poses, the so-called *Ground-Truth Poses* (see section 4.1), can later be used to train a neural network. The program is written in the language C++ and named *6D - Pose Annotation Tool (6D-PAT)*. Its logo can be seen in fig 4.1. Although the program was developed using the *Linux*-based operating system *Ubuntu*, it is compilable on other systems as well.

4.3.1. Frameworks & Third-Party Libraries

The listed frameworks are all necessary dependencies of the annotation tool. There are no dependencies other than the mentioned ones. They are not part of the repository but have to be compiled individually instead. Since all three frameworks/libraries are platform-independent the program can be compiled and run on different systems.

Qt. *Qt* [51] 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 is also chosen as the main framework because it ensures portability of C++ applications by encapsulating system calls of all kind. The program *Qt Creator* is the *Integrated Development Environment (IDE)* that is included in the Qt distribution was used to write and execute the code of 6D-PAT. An IDE combines an editor to edit code an some environment to execute it.

OpenGL. *OpenGL* [52] is a widespread open-source 3D graphics library specification. Implementations for many operating systems exist, thus making most OpenGL using applications compilable in different environments without further modifications. *Mesa 3D*

OpenCV. *OpenCV* [15] is a C++ library created for various computer vision tasks, hence the name. OpenCV provides implementations for tracking, object detection, segmentation and many more. In this work its *solvePnPRansac* is used.

Assimp. *Assimp* [53] 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.

4.3.2. Architecture & Code Design

6D-PAT is primarily a *Graphical User Interface (GUI)* program, i.e. its purpose is to display a window and enable optical interaction like clicking. Thus, the chosen underlying architecture is *Model-View-Controller (MVC)*, which separates the concerns of data management (Model), displaying data (View) and high level logic (Controller). The schematic of the MVC architecture is given in fig. 4.3a. The indirect connections are realized via the Qt Signals and Slots mechanism, which is visualized in fig. 4.3b. To speed up interface creation, *Qt Designer* was used to layout the views. Qt Designer is a graphical tool that allows placement of UI components and linking of signals and slots.

The most important C++ classes of the program are displayed in fig. 4.4. The diagram is simplified for easier understanding. The arrows from the large rectangles imply usage of the classes in the target rectangle. This does not necessarily mean, that all classes of the source use all classes of the target rectangle. The *Pose Editor* uses the *Model Manager* but the *Breadcrumb View* does not, for example. The general structure of the program is the following: the main controller holds references to the *Neural Network Manager*, the *Pose Creator* and the *Main Window*. The neural network manager uses the *Neural Network Runnable* the execute training and inference tasks with the neural network. The pose creator manages the currently clicked corresponding points and computes the new pose when requested and enough corresponding points were clicked. The main window consists of the two main widgets *Pose Editor* and *Pose Viewer*. The pose editor can display an object model O selected from the objects gallery and allows editing all poses P of an image I selected from the images gallery. Both *Gallerys* are owned by the main window, as well. The pose viewer in turn renders all poses P of an image I onto the image. Pose viewer and editor can be used to create new poses (see section 4.3.3). The main window also dispalys the paths selected to load images and object models from using the *Breadcrumb Views*, and allows navigation through the gallerys via the *Navigation Controls*. The settings dialog can be opened from the to edit the preferences of the program. To run the network inference ,the *Neural Network Dialog*, owned by the main window, can be openend to select the images to run prediction on. The main window then displays the *Neural Network Progress View*. Most view classes use the model manager, the *Pose*, the *Object Model* and the *Image* class. Their use should be clear. The main controller sets the *Load And Store Strategy* on the model manager. This class is responsible for reading imags and object models from the hard-drive, as well as persiting poses. The *JSON Load And Store Strategy* uses the JSON format to store poses and read camera matrices. The *Caching Model Manager* is a simple implementation of the model manager interface that chaches images, object models and poses instead of delegating all loading calls to the strategy.

The program expects the camera information and the ground-truth poses file to be in *JSON format*. JSON is a format that is relatively easy for humans to read and is used by many deep learning applications. Transforming JSON formats can be done quickly and therefore provides the possibility to read data from other projects.

The program offers functionality to start the network inference. We realized this feature by starting a new process instead of implementing a Python bridge because the network writes the inferred poses back to the ground-truth file and we do not need to retrieve Python objects as return values.

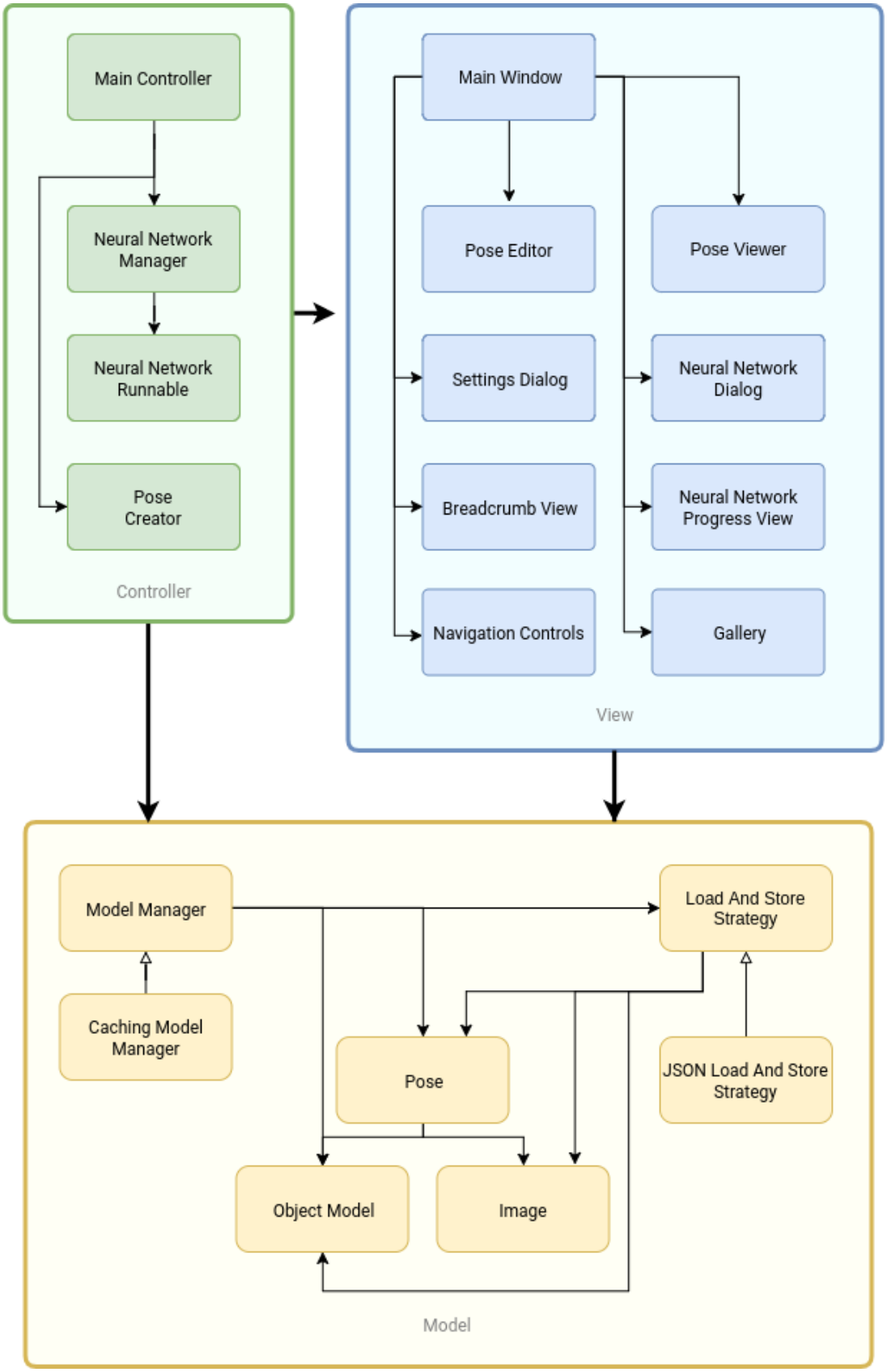


Figure 4.4.: An abstract high-level class diagram of the a subset of the classes of 6D-PAT. Own image.

4.3.3. Manual Annotation

This sections describe the user interface of 6D-PAT and the required steps to annotate images with 6D poses. The process needs some actions performed before the actual annotation can be begin.

Preparation

The first step after starting the program is to open the settings (see fig. 4.5b) and 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 should be used for annotation.

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

Lastly, the location of the JSON file where the ground-truth poses are to be written to has to be specified. If no such file exists an empty one can be created and selected. 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).

If required, the colors of the object models in the segmentation images can be set using the settings dialog as shown in fig. 4.5c. This can reduce the number of displayed tools if many different tools are needed for annotation.

Correspondence and Pose Creation

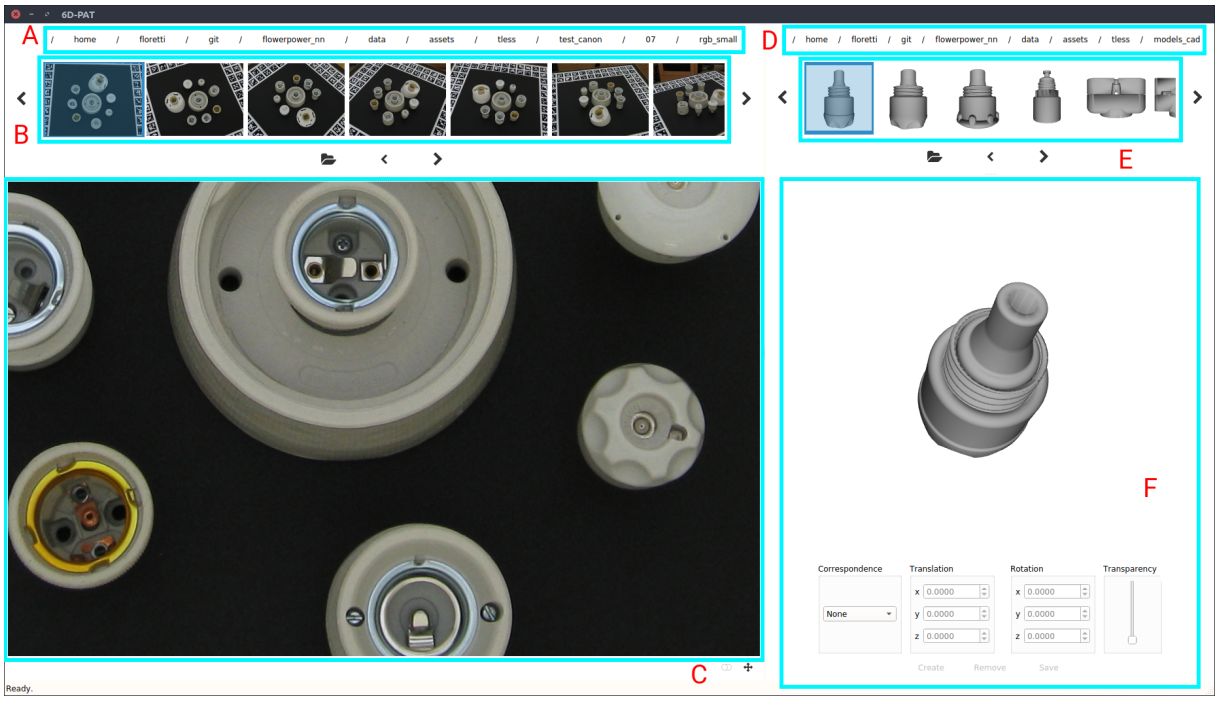
To create a pose an image I has to be selected from gallery B, first. The pose viewer C then displays the image. The model O that the image is to be annotated with can be selected from the gallery E. The pose editor F shows the selected object model. The user can rotate the object the reach otherwise hidden areas of it. The arrow keys can be used to move the object along the x and y and also, if the shift key is pressed, 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 and this way defining the 2D point u of the correspondence. To complete the correspondence, the object model O has to be clicked at the respective position p. This procedure has to be repeated until enough correspondences have been defined to create the new pose P. The minimum is 4 correspondences.

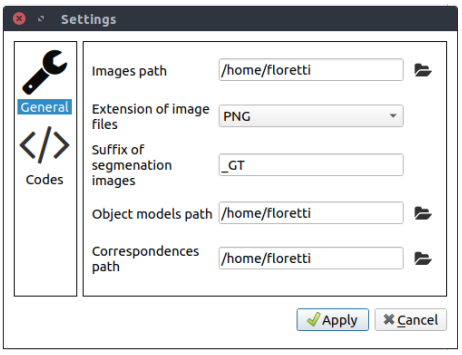
The creation process is shown in fig. 4.6. More correspondences can make the initial pose more accurate. Clicking the "Create" button at the bottom of the pose viewer creates the pose P using the correspondences C_i and OpenCV's *solvePnPRansac* method. The newly created pose is directly selected for editing and can be refined using the controls of the pose viewer. After pose refinement it is necessary to click the "Save" button.

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 gallery B. This operation has to be repeated until the dataset is fully annotated, although intermediate states can be used to train the network already (see chapter 5).

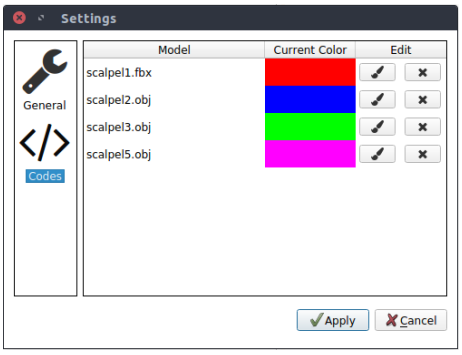
It is possible to use the neural network to predict poses. This requires a proper setup and training of the network beforehand. The prediction process can be started by clicking the "Predict" button in the lower right corner of the pose editor.



(a) The user interface of the annotation tool 6D-PAT. The data displayed are images and object models from the T-Less dataset. The components marked with a turquoise box are the following ones: **A**: The full path of the currently selected folder to load images from. **B**: The 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 point of a new pose. **D**: The full path of the currently selected folder to load object models from. **E**: The rendered 3D preview 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. Own image.



(b) The settings dialog of 6D-PAT. From here the paths to the images and object models can be set, as well as the segmentation image suffix the image type and the file to write the poses to. Own image.



(c) The settings dialog while editing the colors of the object models in the segmentation images. Own image.

Figure 4.5.: The main view of 6D-PAT as well as the two settings views.

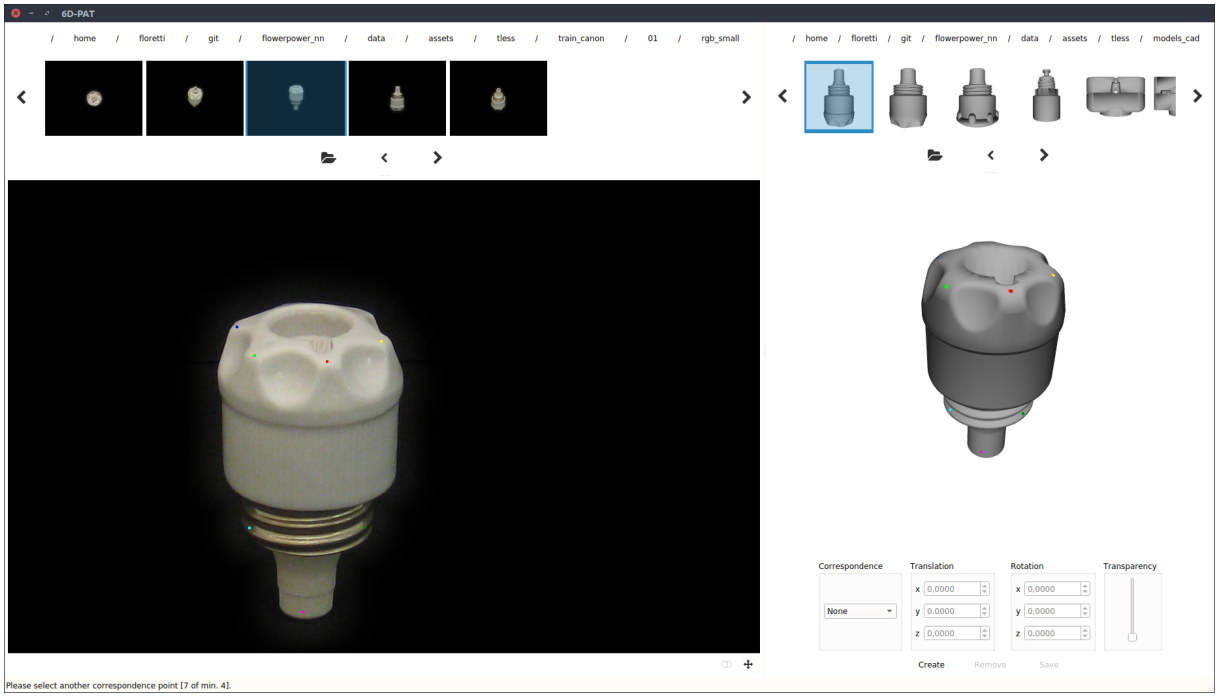


Figure 4.6.: The pose creation process by clicking corresponding 2D and 3D points on the displayed image and object model, visualized by the program as colored dots. Own image.

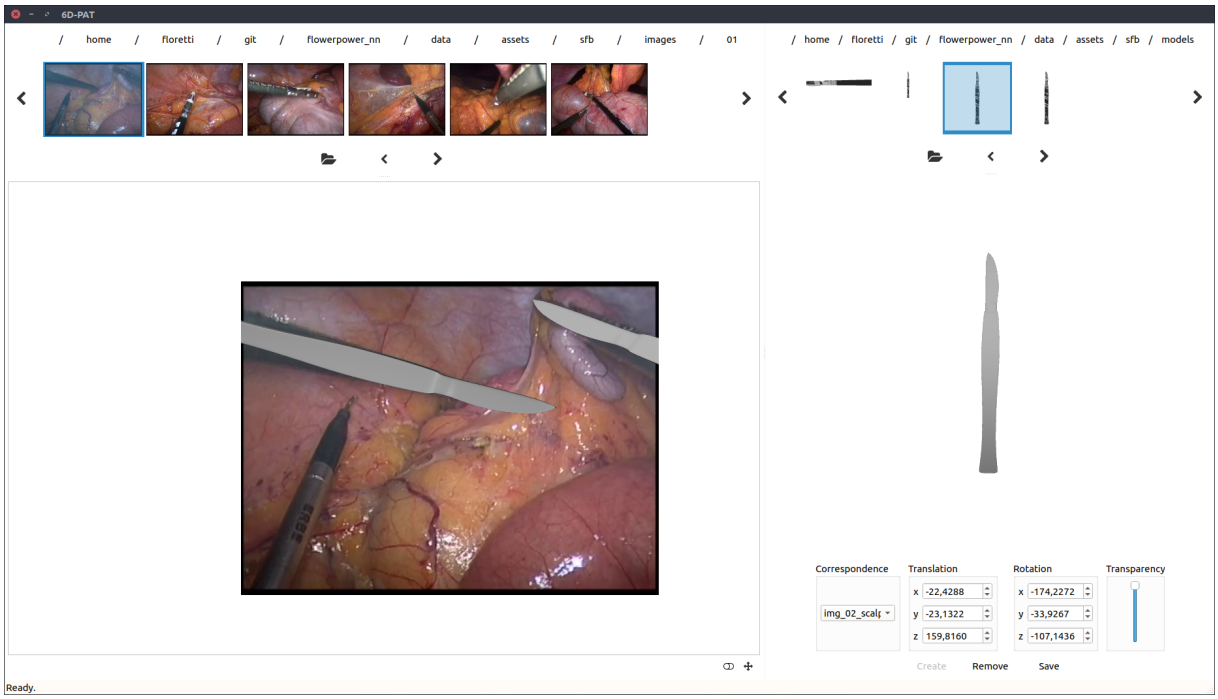


Figure 4.7.: 6D-PAT displaying an image from the medical images dataset together with the associated object models. The models are not from datast and thus do not fit the image entirely. Own image.

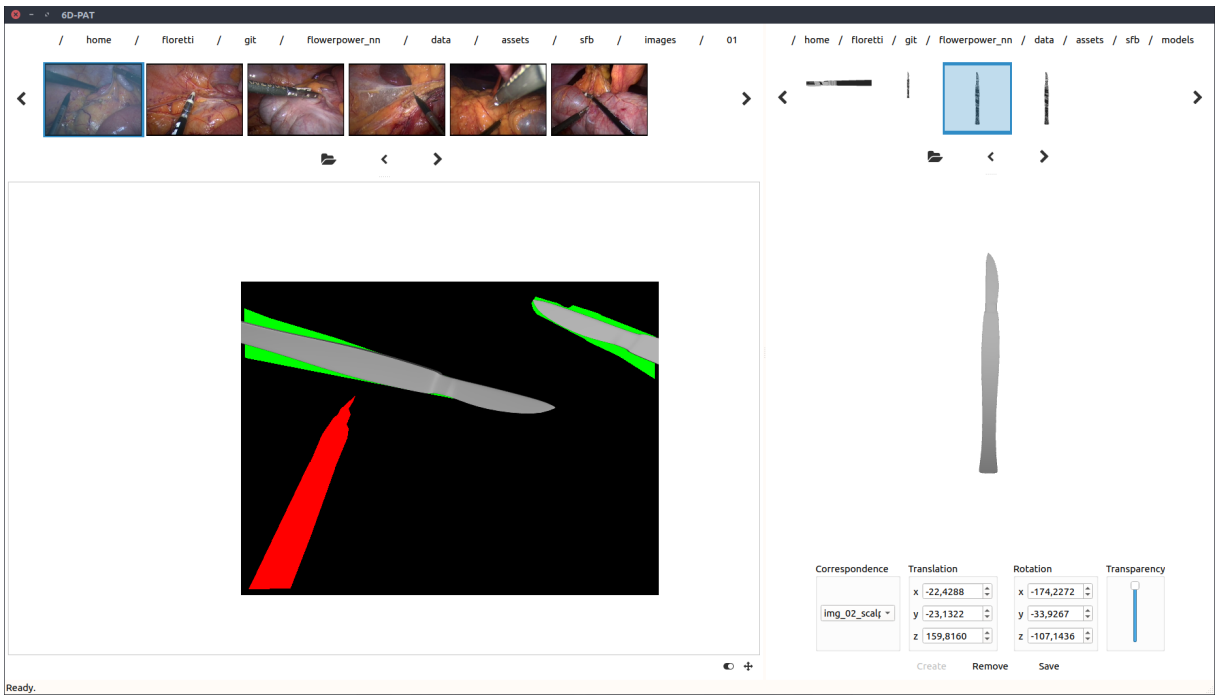


Figure 4.8.: 6D-PAT displaying the segmentation image of an image from the medical images dataset. Segmentation images can be viewed by activating the toggle at the bottom right corner of the pose viewer, if they are present. Own image.

4.3.4. Problems & Difficulties

During the implementation of the program multiple problems arose and complicated the development process. The most drastic ones are mentioned in this section. Issues contradicting the initial usage patterns and their impact on the final design of the program are explained, as well.

One of the major factors that prolong the implementation process was the usage of the only recently released *Qt3D*, which is part of the 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 more and more complications emerged in the almost completed product the Qt3D framework was deemed unusable and omitted in favor of a native OpenGL implementation.

Initially, a desired feature of the program was to present a next unannotated image when the user annotated all poses of an image. This is not feasible for multiple reasons. First of all, a user might still not be content with the poses and might signal that they are finished with the annotation process too early. Thus, it might be disturbing when the program automatically shows the next image.

The datasets to annotate can also have very distinct characteristics, which require the user to choose the next image personally. For a dataset like T-Less, many images have to be skipped due to their similarity. Intuitively, the more diverse poses are annotated, the better a neural network can be trained. Instead of forcing the user to annotate varying images the motivation should be the sped up annotation process as soon as the neural network is able to make plausible predictions.

The diversity of kinds of dataset is also the 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 medical images 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 in 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 medical images. The issue of the object models not fitting the segmentation mask can be seen in fig. 4.2b. Fig. 4.2a shows the actual image and the discrepancy between the models and the pixels belonging to the models.

Although Python provides C++ bindings the integration of it into the application is quite involved. This is due to the used development environment being more complex than the one needed for a standard C++ console program. Qt's own makefile generator called *qmake* is distributed with Anaconda (for details on Anaconda see chapter 5).

By integrating Python into the application Qt Creator's build system tried to use Anaconda's qmake, which caused the build to fail. Unlike normally, the Qt libraries have to be included explicitly in the project. This requires knowledge of the user which Qt libraries they have installed and further complicates the setup process and demands a computer science specialist.

Furthermore, the Python binding is not complete yet. Training is not possible and the 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.

Because the neural network is trained only for one object, the time-intensive (proportional to the overall time needed for inference on one image only) step of loading the trained weights has to be performed each time when running inference for different objects.

4.3.5. Future Improvements of 6D-PAT

To provide an outlook how the program can be improved in the future, some key problems as well as their solutions are listed here. An important feature that should be implemented is moving the object models around on the displayed image by dragging them. It should also be possible to rotate them with the mouse. The initial poses of the models using the clicking procedure are pretty accurate in many cases already. But sometimes, when the camera is looking along an axis of the object directly, it happens that the rotation is far off. The user can correct the poses with the provided controls. But using them takes some skill and time. The describe editing process would profit in terms of time needed per annotation.

The problem of the time needed for loading the weights of the neural network could be solved by offering an intermediate screen that allows to select many images and the object that is to be annotated. The time of loading the weights gets amortized when running inference for many images. Another possibility is to add a component that allows selection of the weights to be loaded and inference is then only run for the corresponding object. This way a reference to the network with the loaded weights could be held and inference could be run without the loading step.

Two options arise to fully include the neural network in the annotation tool. The first is to complete the Python bridge to support all network operations. It is not possible to automatically set all parameters for the network but the need for a deep learning expert can probably be reduced to a minimum and limited to setting up the network. The other option is to completely re-write the application in Python. The initial intuition that a C++ program offers superior performance over Python is not tenable. Although Python is slower in general Qt bindings for Python allow usage of the C++ components by calling them from Python. The same holds for various OpenGL bindings for Python. The author of this work assumes that there would be no

significant performance loss when using Python as the main language. Including the network is a lot easier when using only Python. An application in Python is also fully portable and experienced users can easily and quickly modify the tool. Although the methods described in chapter 5 are a feasible way of operating the neural network, incorporating the network into the annotation tool would allow inexperienced users to make use of it, too.

5. Semi Automatic Annotation

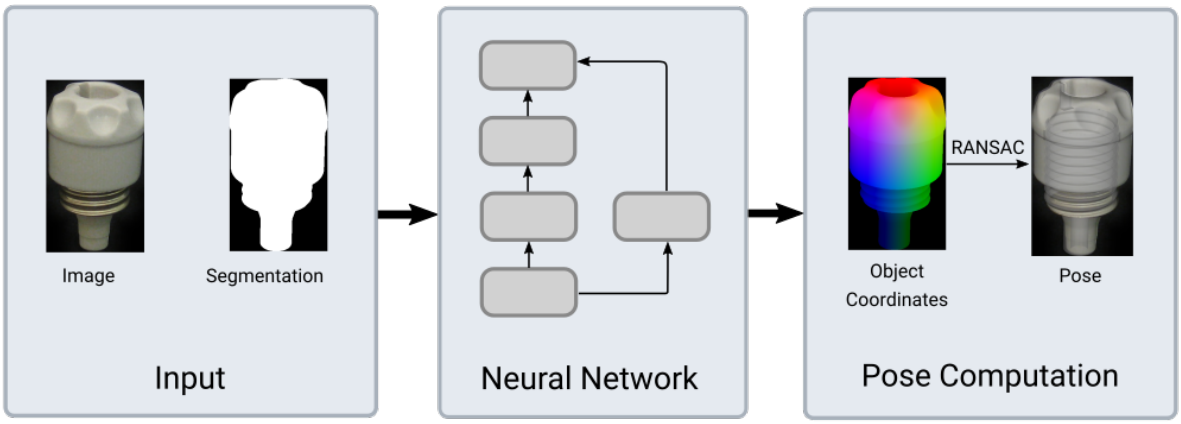


Figure 5.1.: The schematic pipeline of the neural network. The input to the network are the image and the corresponding segmentation image. The neural network then uses the segmentation to retrieve the pixels to predict object coordinates for. In the final pose computation stage the object coordinates and their 2D locations are used to retrieve the best pose using RANSAC. The object's transparency has been decreased to render the final pose. Own image.

To assist in the process of manual image annotation presented in chapter 4, we developed a neural network tailored to the task of 6D pose estimation with the medical images in mind. This chapter describes the network architecture and its variations, as well as different approaches to train the network and the modified annotation process resulting from the usage of the network.

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. Fig. 5.2a visualizes such a connection.

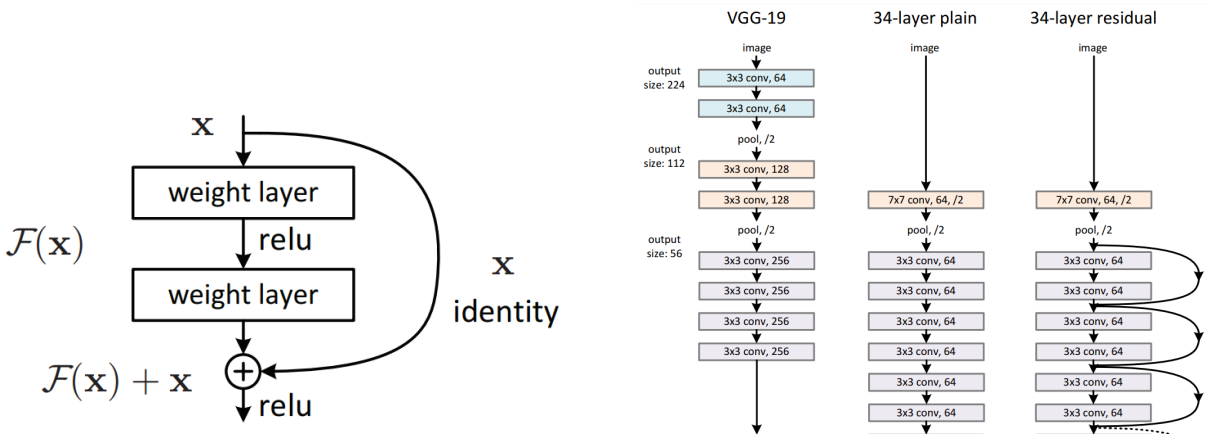
Depth Image. A *depth image* is an image D that contains the distance d of the camera to the surface for each pixel u in an image I. Depth images can be created using special cameras or from stereo images and are often used in pose estimation and other computer vision tasks. A depth image can also be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

Training Set. A *training set* is a subset of the dataset intended to train a neural network with. The training set is the set that the network actually uses 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.

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 (thus the name) the networks new configuration which emerged from a run within the training.

L1 Loss. The *L1 loss* is a loss function that sums up the absolute differences of the single elements, i.e. $L1(x, y) = \sum_i |x_i - y_i|$.



(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. The ResNet allows the creation of deeper network as the problem of vanishing gradients, i.e. very small gradients at deeper layers, can be targeted this way. The image has been cropped.

Figure 5.2.: A key component of ResNet: the residual connection. Left shows the connection in detail and right shows a comparison with the VGG architecture. Images from [7].

L2 Loss. The *L2 loss* is a loss function that sums up the squared differences of the single elements, i.e. $L2(x, y) = \sum_i (x_i - y_i)^2$.

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.

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

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 for example and the size of the layer's kernel can be adjusted, as well. A larger kernel results in more pixels used to compute one output value. Similarly, skipping pixels while the kernels still overlap, or at least no pixels are completely omitted in between, means that the receptive field size grows. The field size has to be adjusted to the problem.

5.2. Network Architecture

The architecture of the network is adapted to the problem specific to the medical images. This means that the network expects segmentation masks as input together with the actual images. There is no mechanism of inferring the masks. This significantly reduces the complexity of predicting the position of the object, although object coordinates implying different poses can still lead to significant errors. This precondition yielded to the pipeline visible in fig 5.1 which takes the actual image and the segmentation image as input. The images shown are already cropped to the segmentation mask but in later applications the image size and proportion of pixels belonging to the object can be arbitrary. It is important to note that the network does not take depth images as input opposed to many other pose estimation approaches. This is owed to

the nature of the provided medical images which do not contain depth information. The network then 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. ResNet was first presented in [7] and enabled researchers to create deeper networks than before. Two major problems arise with very deep neural networks. The vanishing gradients problem, which occurs in deeper networks, means that gradients gradually become 0 in later layers. This can problem be targeted with batch normalization. The second problem of an increasing training error the deeper the network becomes can be solved using ResNet (or flavors of it). 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 $f(x) = x$, ontop of the network increases the training error. This phenomenom can be partly compensated with residual connections [7]. 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* [37]. The residual connections are clearly visible as the arcs of the rightmost architecture. ResNet has proven to produce reliable and accurate results in many computer vision tasks.

The basic structure of the network can be seen in fig. 5.3. The recurring blocks are the building blocks of the network and consist of three convolutions on the long path (the left part of the block) and depending of the type of block, i.e. convolution or identity block, the right path directly adds the identity to the output or performs another convolution before the addition. The number of operations on both paths is higher than the number of convolutions only because activations and batchnormalization are added, as well.

5.2.1. Loss & Optimizer

Since the network outputs the 3D coordinates for each pixel in the input image the straightforward approach is to penalize the differences with respect to the ground-truth coordinates. The loss function sums up the absolute differences 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 is the L1 loss. The reason why we chose the L1 loss over the L2 loss is that the L2 loss penalizes outliers more but outliers potentially get eliminated in the RANSAC algorithm during pose computation. We used L2 regularization for the weights. Due to its performance (see chapter 6), we selected Adam as the optimizer for all networks.

5.2.2. Variations

To find the optimal network structure a total of 6 network architectures were created and later compared in chapter 6. The first design of the network consisted of 23 convolutional layers with a receptive field-size of 99. A characteristic of the medical images is that the tools have writings on them which is often not or only partly visible. This is the reason why we reduced the field-size to keep the network from detecting poses based and the letters. Due to the complications described in section 4.3.4 annotating the medical images was not possible and we were thus not able to fully prove our statement. The second architecture added more layers to a total of 35, while decreasing the receptive field-size to 67. The third architecture kept the number of layers of the first but reduced the field-size to 51. We also created three deeper networks of 50 layers, of which two omit bachtnormalization and use dropout instead, both with a receptive field-size of 59. The difference between the two is the number of dropout-layers and the dropout percentage. The third still employs batchnormalization, again with a receptive field-size of 59.

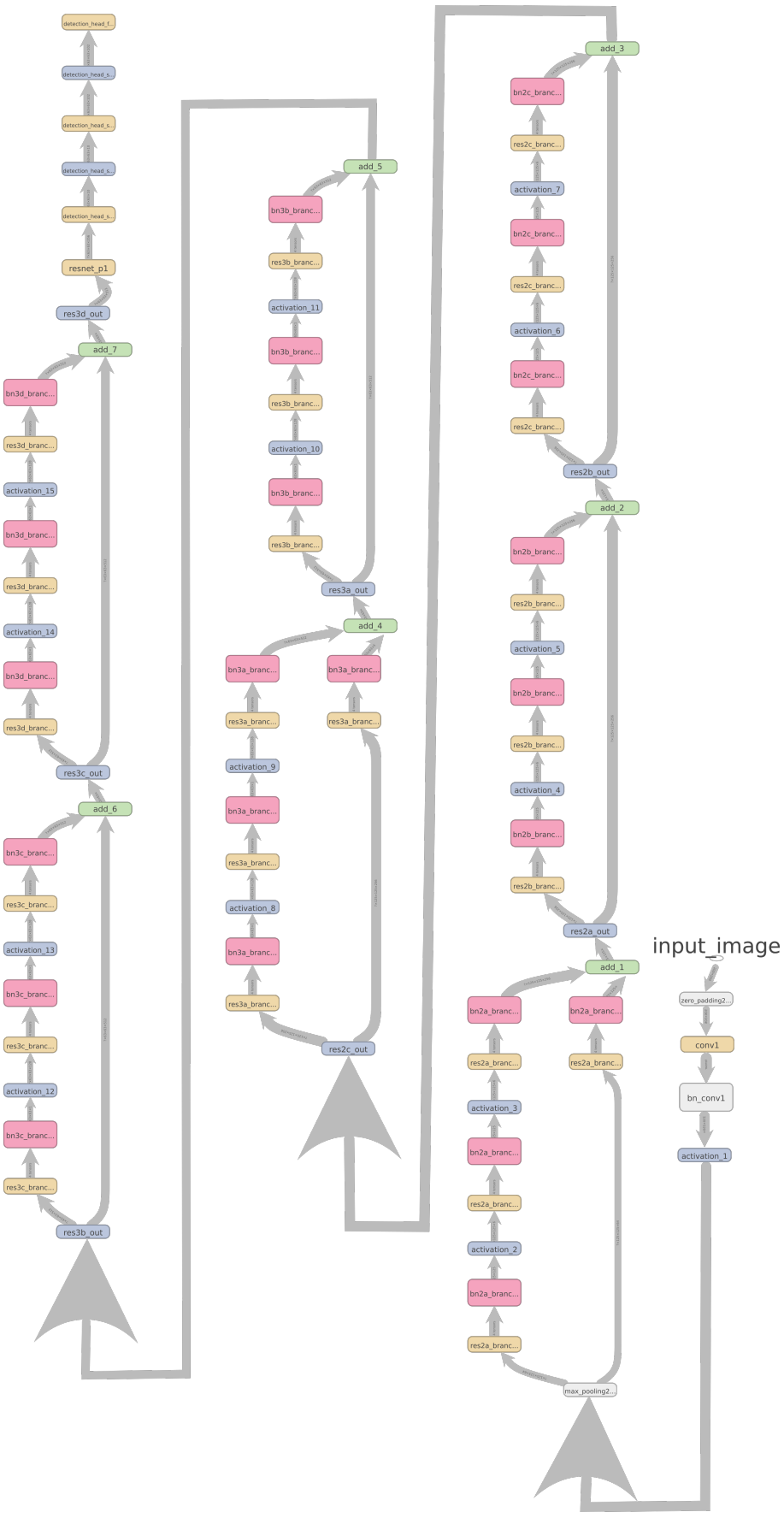


Figure 5.3.: An overview of the shallower architecture used in the project. The visualization was created using *Tensorboard* but modified to fit one page. The large arrows have no meaning and are only there to emphasize where the network continues. Own image.

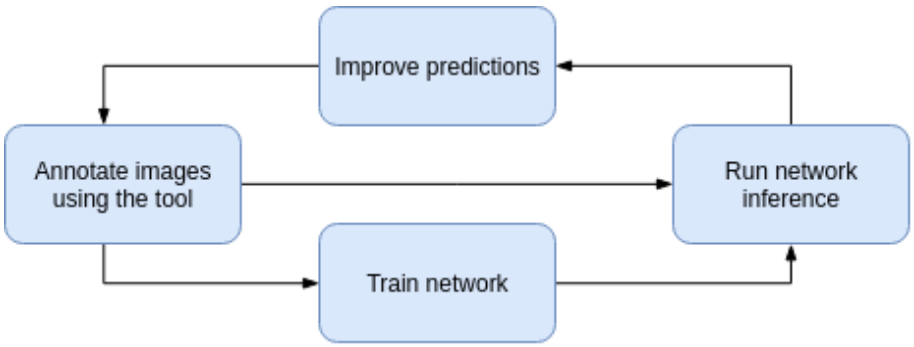


Figure 5.4.: The new workflow of the annotation procedure including the neural network. First, the user has to annotate enough images from the dataset to be able to train the network. After successfully training the network it can be used to predict poses on a subset of the dataset. The user can then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

5.3. Implementation

The neural network and all of its utility functionality is implemented in Python using Tensorflow [55] and Keras [56]. Tensorflow is a deep-learning framework that uses C++ do perform the actual calculations. It provides many different layers, as well as tools to modify images, automatic differentiation of the loss function and a lot of other functionality. Keras is a framework that makes using Tensorflow easier by providing even higher-level access and allows to create neural network in a few lines of code. Because most operations of a neural network written in Tensorflow and Keras are performed using C++ code and because computation can be transferred to the GPU at nearly no extra effort, such a network is considerably faster than an implementation in Python might imply.

5.4. 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 annotate images, a user can use the annotated images to train the neural network. After training the neural network, the user can run the network on new images to predict poses and correct them with the aid of the annotation tool. The improved pose predictions can serve as input to further train the network. There are two main strategies of training a neural network that we will consider here. Those two are training from scratch and an incremental training approach.

Training from Scratch. *Training from scratch* describes the method of training the network with all images (usually divided into training and validation set) and initializing the weights of the network to 0 or randomly. This procedure requires a lot of time as all images of the already annotated ones have to be processed. It also requires many training examples but can on the other hand work with uninitialized weights.

Incremental Training. *Incremental training* can be performed when the user has already annotated some images and trained the network to some extend. After more images have been annotated the network is trained again. But instead of training from scratch, the pre-trained weights are used as the initial weights and the network is fed examples from the freshly annotated images. This process is usually finishes faster, as the network gets to see fewer training examples but requires that the user trained the weights of the network already to some extend.

Inference. After training the network using one of the two training strategies, it can be used to predict poses on unseen images. To amortize the time needed to load the weights of the network

it is best to run inference for many images and not only one. The network inference can be directly called from the annotation tool that we presented in chapter 4.

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 [12].

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 compensate this as best as possible. First, we test the different network architectures that were described in section 5.2.2 using the same parameters. We then choose the best architecture for further experiments, in which we evaluate the two different training schemes introduced in section 5.4: training from scratch and incremental training.

6.1. Terminology

TIFF. *Tagged image file format* is digital image format that supports 32-bit floating point values. This is the reason why this format was chosen as the container for the ground-truth object coordinates.

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

6.2. Datasets

The initial intention to completely annotate the medical images provided at the beginning of the project 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 [57], [58] and [59]. But those resemble surgical tools less than the objects of the T-Less dataset.

The T-Less dataset was released in 2017 by Hodaň et al. [12]. 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 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 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 provided a tool set to render the objects at a given pose,

etc. 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 one shown. The dataset provides depth images, as well, but this is irrelevant to our setting.

6.3. Data Preparation

The *T-Less* dataset does not provide 3D object coordinates. This is the reason why we rendered them ourselves. The script is part of the network package. Segmentation masks were rendered as well. Because the 32-bit TIFF object coordinate ground-truth files used up too much disk space when in original size they were cropped to the relevant area by determining the smallest box around the segmentation pixels. Because a part of the dataset had to be cropped the author of this worked deemed it best to crop all input to the respective segmentation masks. As a result, the network expects all input to be cropped. Or, to be precise, the size that the network is configured to expect as input has to be larger than the size of the segmentation images. The network runs with other configurations but doesn't produce any usable output in this case. There are no other alterations made to the data.

6.4. Training Experiments

This section describes the configuration of the different training experiments, i.e. which model was used, how many images, etc. The chronological order of the experiments is reflected here. First, we compare the SGD and Adam optimizer in 6.4.1. Then we evaluate the different architectures against each other in 6.4.2. Finally, asses the two training strategies training from scratch and incremental training in 6.4.3.

6.4.1. Optimizers

We used the first network architecture to compare the SGD and Adam optimizer. The first network architecture consists of 23 layers and has a receptive field-size of 99 per output pixel. The decreasing learning rates for SGD were set to ... TODO: insert learning rates, 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. Fig. **??** shows the loss of both experiments during training. Since Adam declines much faster than SGD, we chose Adam as the optimizer for all other experiments.

6.4.2. Architectures

The experiments run on the different architectures were all performed using the Adam optimizer with the parameters mentioned above. The major difference between the experiments is the architecture itself. The training process were run for a different number of iterations because some architectures converged earlier than others. The different losses are displayed in fig. **??**.

6.4.3. Training Strategies

6.4.4. Runtime Analysis

This section analyses the inference runtimes of the different architectures. Deeper networks need longer to perform an inference run on an image, which is reflected by table ...

7. Conclusions

In this chapter we summarize to work presented on the previous pages. We also draw conclusions from the results and give a an outlook of possible research based on this work in the future.

7.1. Summary

7.2. Future Work

A. Network Architectures

B. Experiments

List of Figures

List of Tables

8. 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. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, 01 2012.

[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] 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

[7] 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

[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] 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

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

[11] 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

[12] 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.

[13] 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

[14] 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

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

[16] 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

[17] 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

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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

[24] 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.

[25] 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

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

[27] 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.

[28] K. Pauwels, L. Rubio, J. DÃŋaz, 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.

[29] 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.

[30] 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

[31] 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

[32] 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

[33] 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

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

[35] 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

[36] 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.

[37] 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

[38] 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.

[39] 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

[40] 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

[41] 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

[42] 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

[43] 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.

[44] 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.

[45] 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

[46] 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

[47] 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

[48] 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.

[49] 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.

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

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

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

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

[54] The Qt Company, "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] 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

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

[57] 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

[58] 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

[59] 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.