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: August 1, 2018
Defended on: Not yet

TECHNISCHE UNIVERSITÄT DRESDEN

COMPUTER VISION LAB DRESDEN

Fakultät Informatik - Institut KI - Professur Bildverarbeitung

Aufgabenstellung für die Belegarbeit

Thema: Annotation of Laparoscopic Surgery Images with Online Proposal Generation

Name: Florian Blume

Studiengang: Diplom Informatik

Matrikel-Nr.: 3924990

Beginn am: 30.02.2018

Einzureichen am: 30.08.2018

Betreuer: Eric Brachmann

Verantwortlicher Hochschullehrer: Dr. Dmitrij Schlesinger

During laparoscopic surgery, a surgeon operates with special tools through the closed abdominal wall based on the view of an endoscopic camera. Although this type of surgery is beneficial for the patient, it is also very involved for the surgeon. Augmented reality could potentially help the surgeon by overlaying diagnostic information or navigation cues. One particular task that has to be solved to achieve this goal is the real-time 6D pose tracking (3D rotation + 3D translation) of all surgical tools in the endoscopic live stream. In recent years, major breakthroughs in computer vision, including object pose estimation, were driven by techniques from machine learning. However, these methods require a large amount of training data which is not readily available in the medical domain. The goal of this project is the development of an interactive tool for the annotation of laparoscopic surgery footage. The user should be able to accurately annotate the 3D rotation and 3D translation of all surgical tools visible in a random surgery frame. The annotation results should be stored, the frame removed from the stack of unannotated data, and a new frame should be presented. Since a large amount of data has to be processed, the annotation process must be very effcient in design. Based on previous frames already annotated, the system should propose an initialization of the 6D poses in the current frame. For this purpose the tool should include an online learning component, e.g. a CNN that learns to predict correspondences between the image and 3D models of the surgical tool.

Tasks:

- Survey related literature regarding data annotation and online learning.
- Create interaction concepts to enable a user to annotate the 6D pose of surgical tools

very efciently. The images are already annotated with segmentation masks and tool IDs. 3D models of the tools are also given.

- Implement an annotation tool based on the interaction concept.

Optional:

- Implement a component which can propose 6D poses of surgical tools as an initialization for the user. The user should be able to dismiss or refine the proposal.
- The component should be trained based on data already annotated (in a previous session or another user) and be further refined through online learning during the annotation process.

Unterschrift des Studenten

Unterschrift des Hochschullehrers

Here is an abstract.

CONTENTS

1 INTRODUCTION

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

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

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

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

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

The remainder of the work is structured as follows: Chapter **??** explains the basic concepts of deep learning and pose estimation. Chapter **??** introduces and discusses the latest research in the area of pose estimation, after giving a brief overview over earlier methods. We present the developed tool in Chapter **??**. Chapter **??** describes the workflow of annotating images with the aid of the neural network. In Chapter **??**, we explain the datasets used for the experiments and discuss the latter in depth. The last chapter summarizes this work, draws conclusions and gives a prospect into possible future research.

2 BACKGROUND

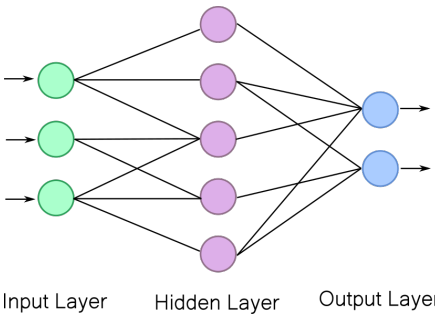


Figure 2.1: Abstract structure of a feed-forward neural network. Most networks have more layers and more neurons per layer.

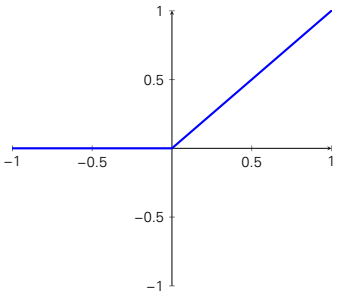


Figure 2.2: Visualization of the ReLU activation function $f(x) = \max(0, x)$.

2.1 NEURAL NETWORKS

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

2.1.1 NEURON

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

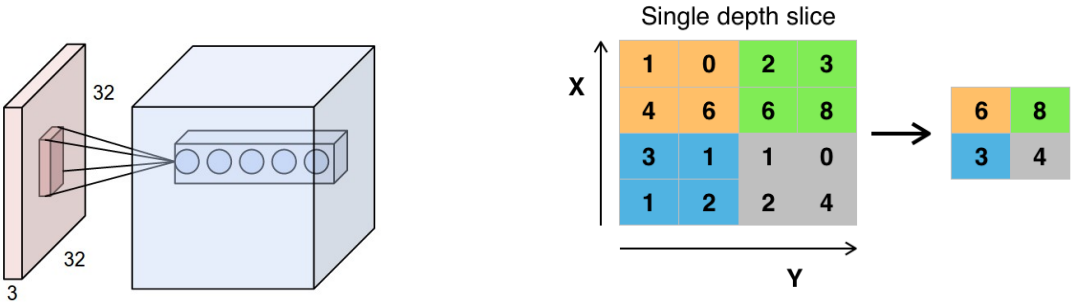
$$y = \sum_{i=0}^{k} w_i x_i + b \tag{2.1}$$

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

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

2.1.2 FEED-FORWARD NEURAL NETWORKS

A *Feed-Forward Neural Network* is a network consisting of layers of neurons. The first layer is called the input layer. Those are the neurons that receive the data that the network is supposed to process. The last layer is called the output layer. The form of the output of the neural network depends on the field of application. It can be object coordinates like in our case, class 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 [**?**], thus the name deep neural networks. All layers can have different numbers of neurons and connections to previous layers. A layer, in which each neuron is connected to every output



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

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

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

of the previous layer using individual weights for each connection, is called a *Fully-Connected Layer*. A schematic overview over the structure of a neural network is visualized in Fig. ??. The green circles on the left are the neurons of the input layer. The purple neurons in the middle belong to intermediate (hidden) layers. The output layer consists of the blue neurons on the right. A neural network can generally be seen as the function $y = f(x, \theta)$, where θ are the weights and biases of the neurons and y is the output of the network.

2.1.3 CONVOLUTIONAL NEURAL NETWORKS

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

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

where O is the output. The ranges of w_0 and w_1 depend on the size of the convolution window. The kernel can be learned but stays the same for the whole convolution. Fig. ?? shows an abstraction of a convolutional layer. The red plane on the left is the input image, the blue cube on the right is the actual layer. The blue cube is broader than the red input because it has multiple filters. Each filter consists of an independent kernel. This means that this layer has multiple outputs, each being a convolution of the input with a different kernel.

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

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

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

2.1.4 NETWORK TRAINING

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

ERROR-BACK PROPAGATION

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

OPTIMIZATION

There exist many different patterns how to update the network weights after obtaining the deltas. The *Stochastic Gradient Descent (SGD)* multiplies the delta by a learning rate and subtracts it from the weight. To ensure convergence, the learning rate is often decreased over time. A more elaborate method is the *Adaptive Moment Optimization (Adam)* [**?**] which computes adaptive learning rates for each parameter. The algorithm computes and stores an exponentially decaying average over past gradients as well as an exponentially decaying average over the past squared gradients. This serves to increase the learning rate in case of small gradients and decrease the learning rate in case of large gradients. Both averages are stored for the next iteration. There exist other optimization procedures that are not covered here.

REGULARIZATION

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

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

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

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

Figure 2.4: Example 3D coordinate representations.

2.2 6D POSE ESTIMATION

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

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

2.2.1 OBJECT COORDINATE REGRESSION

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

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

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

where K is the camera matrix

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

which projects the 3D point transformed into the camera coordinate system on the 2D image

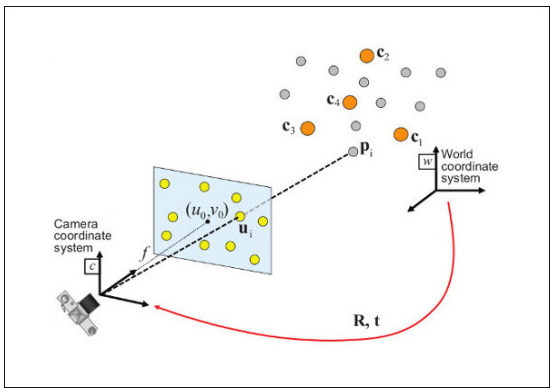


Figure 2.5: The relationship between the camera, the 3D points and their projections on the screen defined by the rotation matrix R and the translation vector t. Image from [?].

plane. The points u and p are both in homogeneous coordinates, i.e. $u = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$ and $p = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$.

The skew factor s is usually 0. The principal point (c_x, c_y) normally is the center of the image and f_x and f_y are the focal lengths in x and y direction, respectively. The values for those variables can vary, for example when cropping the image. Fig. ?? visualizes the relation of pixels and object coordinates. If equation ?? is overdetermined because there are more correspondences than variables, it cannot be solved directly.

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

Algorithm 1 RANSAC

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

3 RELATED WORK

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

3.1 RESEARCH ON 6D POSE ESTIMATION

Pose estimation has become an increasingly interesting problem with the advent of robots in production and the area of virtual and augmented reality [?]. 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 [?], handcrafted feature-based approaches were common for 6D pose estimation [?]. A lot of research matches sparse characteristics detected in the image against a database that contains the related pose information. The works of [?, ?] 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 [?, ?, ?] 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.

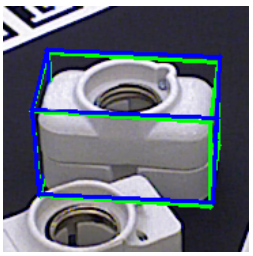
Template-based methods [?, ?, ?, ?] use generated views of discrete viewopints on the object at different angles which are matched against the given image to retrieve the pose. This approach relies mainly on the shape of objects and thus works well for poor-textured or textureless objects [?]. To cover 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 [?]. 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 [?]. 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 [?, ?] 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 [?], 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 [?]. The system developed in [?] 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 [?]. A prominent example of camera localization,



(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 [?].

which represents the problem of estimating the camera position in 3D space relative to the scene, is PoseNet [?]. The underlying architecture of the employed neural network is based on GoogleNet, which was introduced in [?]. The CNN estimates the camera position directly without regression from a single image and is fast. In some indoor cases, PoseNet has an error as high as 50 centimeters, but our application demands high accuracy.

BB8

The recently developed BB8 [?], 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 ??) like [?], they let a deep neural network estimate the object segmentations first (see Fig. ??) and then predict the 2D locations of the 3D corners of the object's bounding box. An example of the corners can be seen in Fig. ??.

Similar to [?], 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 [?] respectively but with the last layer cut off and replaced by a fully connected layer which is fine-tuned.

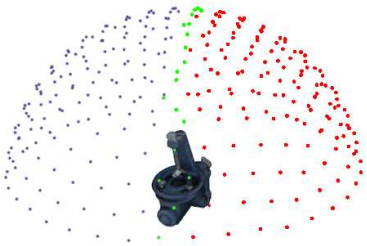
The first neural network, which segments the image, helps estimating the pose of the object in a way that the second network positions its window at the center of the segmentation and estimates the 2D locations of the 3D corners of the object's bounding box. The network reasons globally about the object 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.

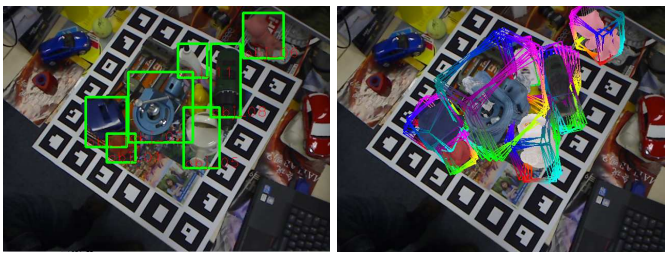
The proposed method offers good performance and can compete with and partly surpasses state-of-the-art research. Yet, object coordinates provide high accuracy too. Furthermore, we assume that the often merely partial visibility of the surgical instruments calls for a non-global reasoning design. Thus,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

SSD-6D

The system presented in [?] is based on the Single-Shot Multibox Detector (SSD) [?]. 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.



(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 [?]

The SSD-inspired network architecture produces feature maps of the input images, that are convolved to predict the class and 2D bounding box (see Fig. ?? left) of objects in the scene. The network also estimates the probability of a discrete viewpoint on the object. Those viewpoints are sampled equidistantly alongside a predefined step size (see Fig. ??). The pose is calculated taking into account the class, bounding box, viewpoint and a guess on the in-plane rotation of the object. The network is trained entirely on images from the MS COCO dataset [?]. 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 [?] or [?]. The optimization process could also be applied to the two mentioned works. Hence it is not investigated further.

KURMANN ET AL.

The work of [?] 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 [?], 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 [?] and [?]. 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 [?] 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 [?], Brachmann et al. adjusted their pipeline from [?] 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 [?] by replacing the energy function by a CNN [?]. 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 [?]. 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 [?] finds, that random forests partly offer a slightly superior performance, neural networks can compete and are therefore the focus of this work.

PERTSCH

Although [?] 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 [?], i.e. uses object coordinates (see section ??), 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 [?], 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 [?] 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 [?].

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 [?] and especially [?] are adopted and further complemented with current research and active and incremental learning.

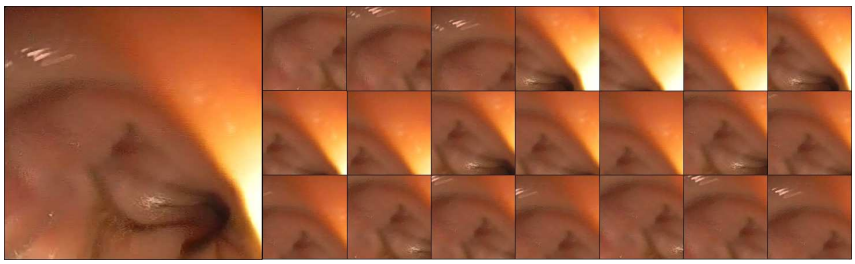


Figure 3.3: A candidate (left) and the patches generated sharing the same label. Images from [?].

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 [?] gives an overview over methods developed before 2011. Wang and Shang, the authors of [?], might be among the first to incorporate active learning into deep learning though, according to [?]. In [?], 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 [?], 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 Fig. ??, 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.

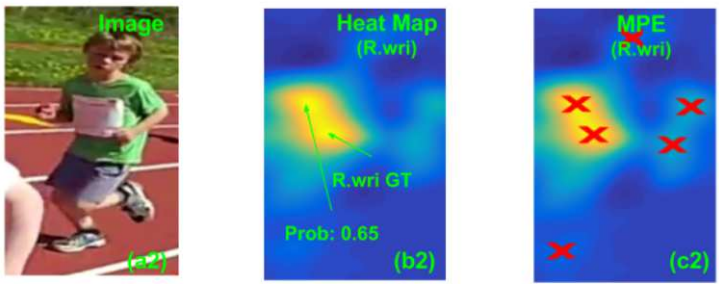


Figure 3.4: An image of a human body, the corresponding heatmap as well as the result of the Multiple Peak Entropy (MPE). Images from [?].

LIU & FERRARI.

In [?], Liu and Ferrari describe new strategy for active learning for human pose estimation, a time-consuming task to produce groundtruth annotations for. Their key contributions consist of an uncertainty estimator for the joint predictions produced by a convolutional pose machine (CPM), and an annotation interface that reduces the time needed by a human annotator to click joints. The predictions by the CPM are heatmaps, with high temperature resembling the most probable position of the joints (the input image can be seen Fig. ??(a2), the heatmap in ??(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 Fig. ?? (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 [?] look auspicious, as a reduction in annotation time can be achieved through the interface and the selection process. Regrettably, it cannot be directly applied to our problem, as we need a problem-specific certainty estimator. But the idea of requesting similar images to be annotated might yield a performance gain, and similarly to the presented interface we present the user with an initial probable pose, too, to make only small corrections necessary.

4 MANUAL ANNOTATION

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

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

4.1 TERMINOLOGY

Image. An image I is a 2D matrix of pixels. The pixel u at position (i, j) is referenced by the tuple (x, y), where $x = j$ and $y = i$. The author of this work chose this notation over the row-major matrix indexing because images are often column-major indexed.

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 $T \subseteq M^3$, also called a mesh. The type of object is not restricted. In the T-Less dataset [?], the objects are mostly screws and other hardware.

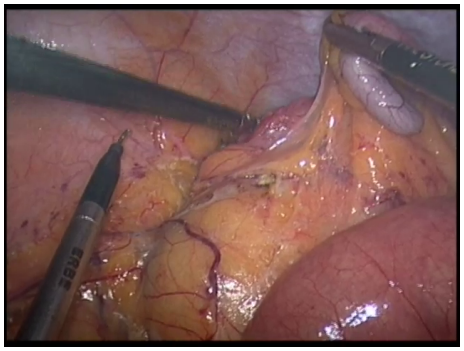
6D Pose. A *6D pose* P is the tuple (R, t). R is the 3×3 rotation matrix and $t \in \mathbb{R}^3$ the translation vector used to transform an object model into camera coordinates (see Section ??).

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

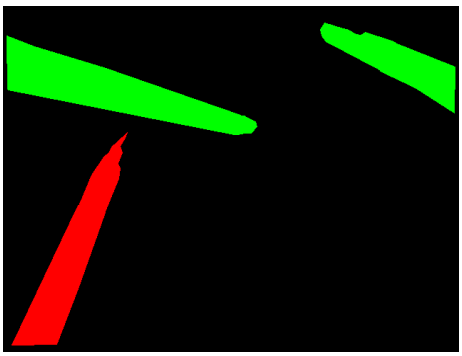
Segmentation Mask. A *segmentation mask* (or *segmentation image*) S for an image I is a second image of the same size. Each position (x, y) of the mask encodes the class of the pixel at (x, y) in I. The set of classes can be defined arbitrarily. In the context of this work, each class represents a type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

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

Depth Image. A *depth image* is an image D belonging to an image I of the same size that contains the distance d between the camera and the surface for each pixel u in I. Depth



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



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

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

images can be obtained from stereo images or using special cameras and are often used in pose estimation and other computer vision tasks. A depth image can also be denoted by *RGB-D*.

4.2 IMAGES OF THE ENDOSCOPIC VISION CHALLENGE

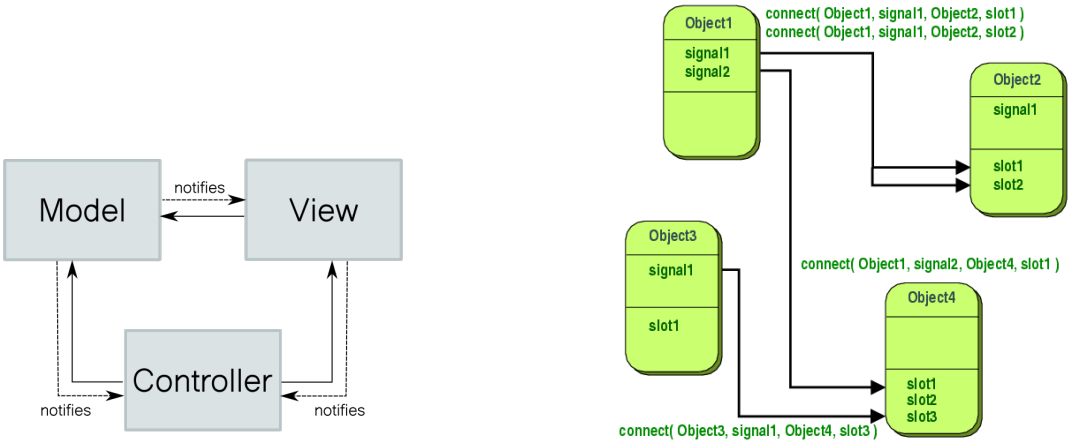
The goal of this work is to provide a system to successfully and efficiently annotate the images of the *Endoscopic Vision Challenge* [?]. The dataset includes segmentation masks but neither object models nor pose annotations or depth images. An example image together with the corresponding segmentation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

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

The creation of sufficient training data for neural networks can be a time-consuming and tedious process. Using non-specialized tools designed for other purposes, like 3D modeling or CAD programs, require the person creating the annotations to get accustomed to complex user interfaces (UIs). The goal of the annotation tool is to provide a system that allows easy and efficient annotation of images - images of the Endoscopic Vision Challenge in particular. The ground-truth poses recovered using the program can then be used to train a neural network. The program is written mainly in the language C++ and named *6D - Pose Annotation Tool (6D-PAT)*. Its logo can be seen in Fig. ??. We developed the program on the *Linux*-based operating system *Ubuntu* but it is compatible to other operating systems, as well.

4.3.1 REQUIREMENTS

Fulfilling the goal of providing a tool for annotating large datasets implies some requirements. Datasets can consist of thousands of images and many object models. To guarantee a fluid workflow, we need to provide the user with a browsable overview over the dataset. This prevents a disruptive annotation process, as the user doesn't need to open the next image or a different object model individually after recovering a pose. The program has to offer the possibility to select and view images and object models, respectively. The most essential part of



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

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

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

such an annotation tool is the incorporation of functionality to recover poses and to edit misaligned poses. Different manual pose recovery mechanisms are possible. Due to the limited time frame of this work, we implemented only one method, which we expected to be the most efficient one.

4.3.2 FRAMEWORKS & THIRD-PARTY LIBRARIES

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

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

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

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

Assimp. *Assimp* [?] 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.

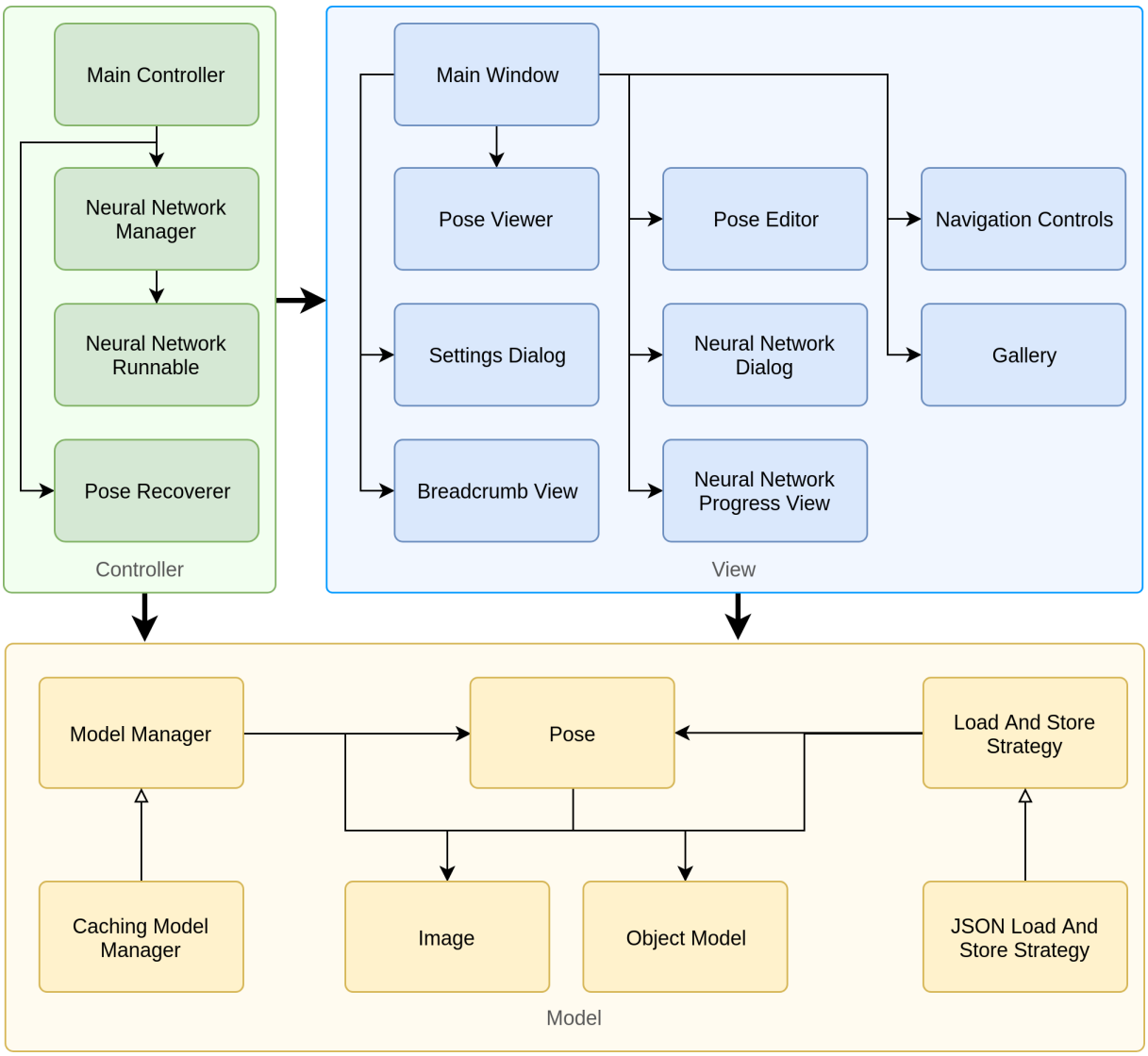


Figure 4.4: An abstract high-level class diagram of a subset of the classes of 6D-PAT. The large rectangles show the affiliation of a contained class in the MVC pattern. Arrows between those rectangles imply which class can use which target class, although not all classes of the source rectangle necessarily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

4.3.3 ARCHITECTURE & CODE DESIGN

6D-PAT is primarily a UI program, i.e. its purpose is to display a window and enable optical interaction for the user, like clicking. Thus, we chose the *Model-View-Controller (MVC)* pattern as the underlying architecture. MVC separates the concerns of data management (Model), displaying data (View) and high level logic (Controller). The schematic of the MVC architecture is shown in Fig. **??**. The indirect connections are realized via Qt's signals and slots mechanism, which is visualized in Fig. **??**. To speed up interface creation, we used *Qt Designer* to layout the views. Qt Designer is a graphical tool that allows placement of UI components and linking of signals and slots without directly writing code and is part of the standard Qt framework.

The most important classes of the program are displayed in Fig. **??**. The diagram is simplified for easier understanding. The large rectangles group the classes by their affiliation in the MVC pattern. The bold arrows between the groups denote which classes can use or know which

other classes. This does not necessarily mean, that all classes of the source rectangle use all classes of the target rectangle.

CONTROLLER CLASSES

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

VIEW CLASSES

The main controller creates the main window, which holds all views of the UI. It is the central view counterpart for the main controller and delegates all alterations requested by the controller to the other views. All signal and slots that are necessary between view classes get connected by the main window. Most of the signals and slots communication takes place between the *Pose Viewer*, the *Pose Editor* and the galleries. Whenever the user clicks an image or an object model in the gallery, the gallery notifies the viewer and editor to display the respective entity. The view classes know the model classes, especially the *Model Manager*, by reference. For this end, the main window passes the model manager reference it received from the main controller to the other view classes.

MODEL CLASSES

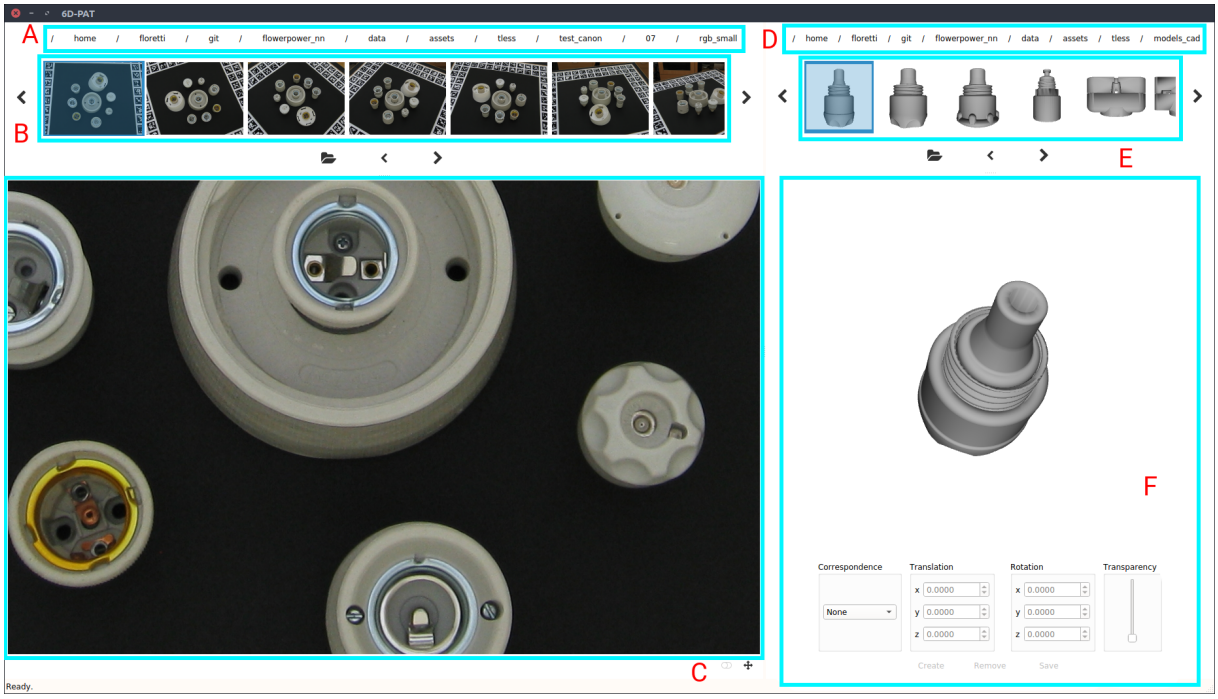
The classes that contain the the paths to the data are the *Image* class and the *Object Model* class. The image also contains the camera matrix K. The *Pose* class references an image and an object model and the respective rotation matrix R, as well as the translation vector t. The model manager and the load and store strategy interfaces were abstracted from the implementations *Caching Model Manager* and *JSON Load And Store Strategy* with extensibility in mind. This way, the strategy can be adapted to use a database instead of a JSON file, for example.

MISCELLANEOUS CLASSES

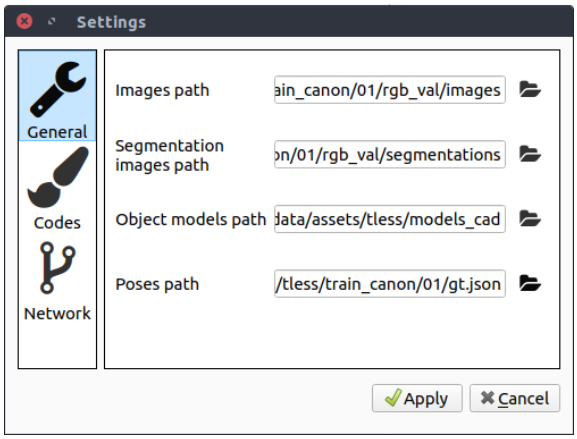
Some classes are not displayed in Fig. **??**. We created a dedicated class to store the current settings of the program. This ensures compatibility between classes whenever the number of parameters of the settings or their type changes.

4.3.4 MANUAL ANNOTATION

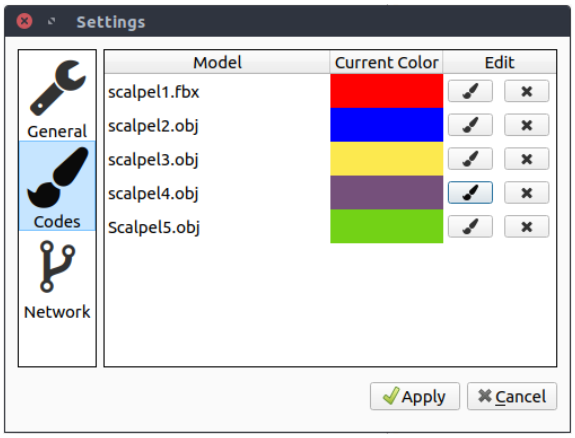
This section describes the user interface of 6D-PAT and the steps required to annotate images with 6D poses.



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



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



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

Figure 4.5: The UI of 6D-PAT.

# Object \ Image	0000.jpg	0150.jpg	0250.jpg	0350.jpg	0500.jpg
01	2:30 min	1:23 min	2:05 min	3:06 min	0:50 min
15	2:16 min	1:51 min	1:35 min	2:20 min	3:07 min

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

PREPARATION

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

The folder of the images has to contain a JSON file that holds the camera matrix K for each individual image. If no camera info file exists, a Python script that is distributed with the neural network can be used to 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 matches image I at index j with the segmentation image S at index j.

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

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

CREATION OF CORRESPONDENCES AND POSE RECOVERY

The component corresponding to the letters that we use in the following paragraph can be taken from Fig. **??**. To annotate a new pose, the user has to select an image I from the images gallery (see B in Fig. **??**) first. The pose viewer (see C in Fig. **??**) then displays the image. Gallery E is used to select the object model O that the image is to be annoated with. The pose editor (see F in Fig. **??**) shows the selected object model. The user can rotate the object to view otherwise hidden areas. Using the arrow keys, the user can move the object along the x and y axis and also, 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. This defines the 2D location 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 $C_1, \cdots, C_n$ have been defined to recover the new pose P. The minimum number of correspondences is 4.

The creation process is shown in Fig. **??**. 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 can be refined using the controls of the pose viewer. After pose refinement, it is necessary to click the *Save* button.

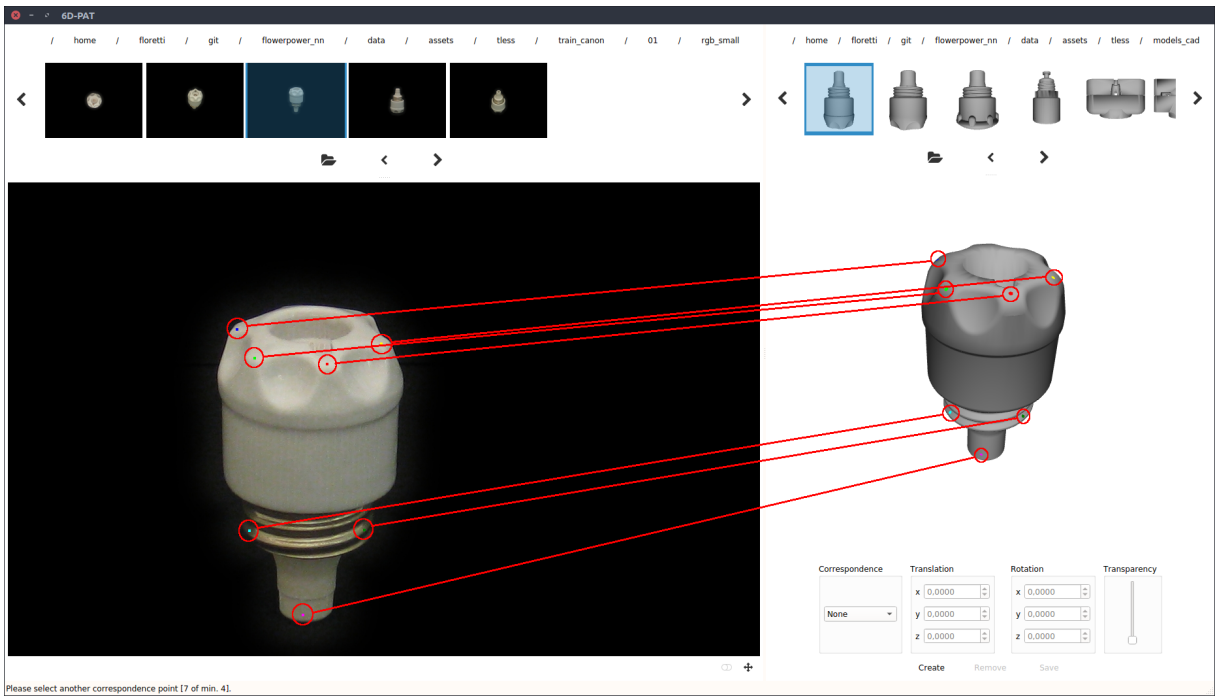


Figure 4.6: The pose creation process. The user has to click the image first and then the corresponding 3D location on the object model. The red circles and red lines were added afterwards to emphasize the colored dots drawn by the program.

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

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

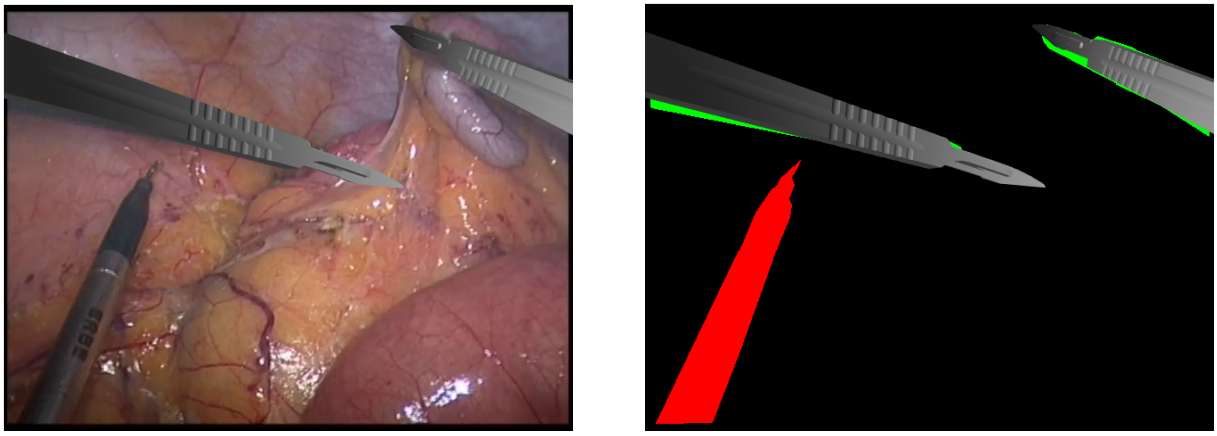
Example times of the manual annotation process (without aid of the network) are given in Table **??**. The measured times were achieved by the author of this work trying to recover a single pose of the specified object model. The discrepancy between the measurements depends on the computed initial pose. A view that shows the object along multiples axes works best because defining correspondences along those axes fixes the rotation.

We gave this way of recovering poses preference over other possible techniques for multiple reasons. Another intuitive way of recovering poses is dragging the object model from the pose editor onto the pose viewer at the approximately correct position. We assumed that the chosen procedure is faster, because when enough precise corerspondences are clicked, rotation and position are accurate immediately.

4.3.5 PROBLEMS & DIFFICULTIES

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

One of the major factors that prolonged the implementation process was utilizing the *Qt3D* framework, which is part of the main Qt framework, to realize all graphical processing. 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



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

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

Figure 4.7: Example annotations of images of the Endoscopic Vision Challenge dataset. The object model is taken from [**?**].

without consultation. After complications still emerged in the almost completed annotation tool, the Qt3D framework was deemed unusable and omitted in favor of a native OpenGL implementation.

Initially, a desired feature was to present another unannotated image after the user finished annotating all poses in the current image. This is not feasible for multiple reasons. First of all, a user might still not be content with the poses. Selecting the next image programmatically could happen too early and thereby disrupt the annotation process of the user.

The datasets can also have very distinct characteristics, which require the user to choose the next image manually. For a dataset like T-Less, many images have to be skipped due to their similarity. A network trained on one view on an object only, will likely predict inaccurate poses for a different view showing other characteristics of that object. The support of an accurate network in the annotation process can be used as a motivation for the user.

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

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

The Python binding is not complete, yet. Training is not possible and a deep learning expert has to setup the network and its parameters to enable its usage in the program. The current

state of the incorporation of the network is to be seen as a proof of concept that requires further development.

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

4.3.6 FUTURE IMPROVEMENTS OF 6D-PAT

To provide an outlook how the program can be improved in the future, we mention solutions to some of the problems of Section **??**. An important feature that should be implemented is allowing to move the object models around on the displayed image by dragging and rotating them using the mouse. The initial poses using the clicking procedure are already sufficiently accurate in many cases. But when the camera is looking along an axis of the object, the rotation can be far off. The user can correct these poses with the provided controls. The described editing process could profit from this functionality in terms of time needed per annotation. To provide the user with a better overview of the image and its annotated poses, a zoom option should be implemented, as well.

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

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

5 SEMI-AUTOMATIC ANNOTATION

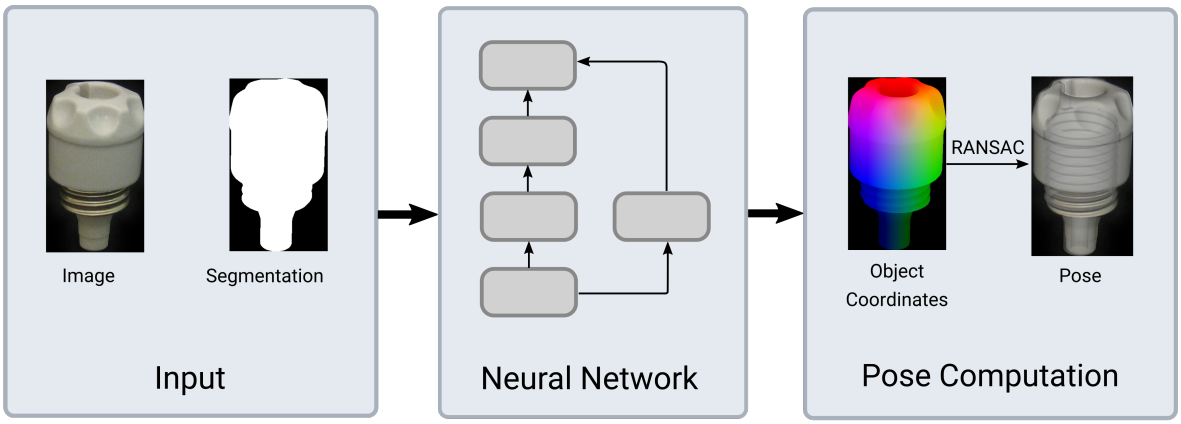


Figure 5.1: The pipeline of the neural network. The two inputs of the network are the image and the corresponding segmentation image. The neural network then uses the segmentation to retrieve the pixels to predict object coordinates for. In the final pose computation stage the object coordinates and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

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

5.1 TERMINOLOGY

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

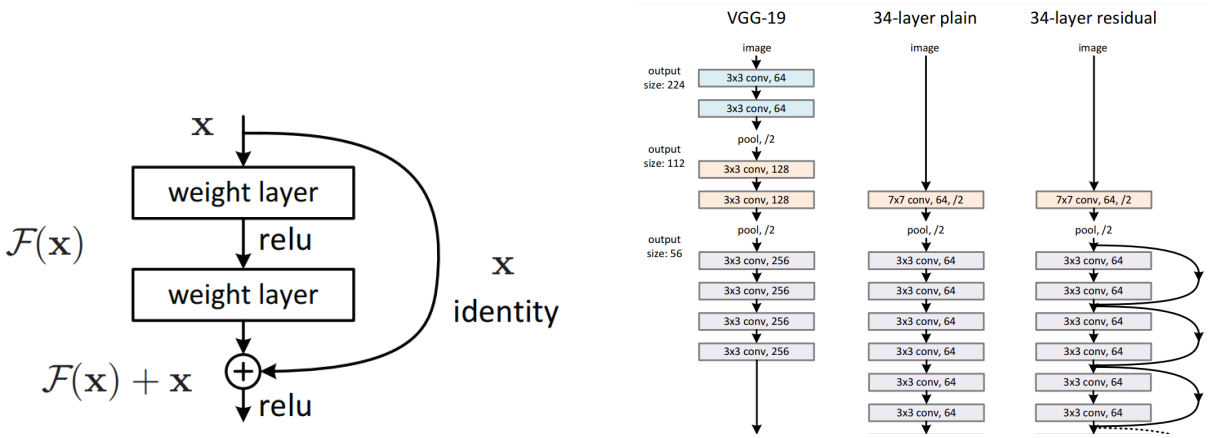
Training Set. A *training set* is a subset of the dataset intended to train a neural network with. The network uses the training set to adjust its weights as opposed to the validation set.

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

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

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

L1 Loss. The *L1 loss* is a loss function that sums up the results of the function g applied to the k elements in a training batch, i.e. $L_1(x, y) = \frac{1}{k}\sum_k |g(x_k, y_k)|$. We set g to be the euclidean distance: $g(x_k, y_k) = \sqrt{\sum_i (x_{ki} - y_{ki})^2}$, x_{ki} and y_{ki} being the i-th entries of the vectors x_k and y_k.



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

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

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

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

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

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

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

5.2 IMPLEMENTATION

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

5.3 NETWORK ARCHITECTURE

The architecture of the network is adjusted to the problem domain of the images of the Endoscopic Vision Challenge. There is no mechanism of inferring the masks and the network expects RGB-only images without depth information. This significantly reduces the complexity of predicting the position of the object. Heavily contradicting object coordinates can still lead to positional errors, though. Fig. **??** shows the processing pipeline of the network which takes the image and the segmentation image as inputs. The network processes each pixel which is part of the object according to the segmentation image and outputs the 3D coordinates. As explained in Chapter **??**, we then compute the optimal pose using RANSAC and the 2D-3D correspondences.

The basic architecture that we chose for the network is called ResNet. He et al. first presented ResNet in 2015, which enabled researchers to create deeper networks than before [**?**]. Two major problems arise in very deep neural networks. One is the vanishing gradients problem, which means that gradients gradually become 0 in deeper layers. A 0 gradient implies that the weights won't be changed. This problem can be targeted with batch normalization. The scond problem is that making a network deeper without further adjustments can lead to an increasing training eror. 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 [**?**]. An example of a residual connection can be seen in Fig. **??**. Fig. **??** shows a comparison of ResNet with the architecture *VGG* [**?**]. The residual connections are clearly visible as the arcs of the rightmost architecture.

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

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

5.3.1 LOSS FUNCTION & OPTIMIZER

Since the network outputs the 3D coordinates for each pixel in the input image a straight-forward approach is to penalize the euclidean distance with respect to the ground-truth coordinates. The loss function sums up the absolute distances of the individual XYZ components of the predicted object coordinate and the ground-truth, but only at the relevant positions according to the segmentation mask. This resembles the L1 loss (see Section **??**). The reason why we chose the L1 loss over the L2 loss is that the L2 loss penalizes outliers more. Outliers will already get eliminated in the RANSAC algorithm during pose computation. The L2 loss might strive for a worse compromise. We used L2 regularization for the weights. We compared SGD and Adam to find the best optimizer for the network (see Chapter **??** for details on the optimizers).

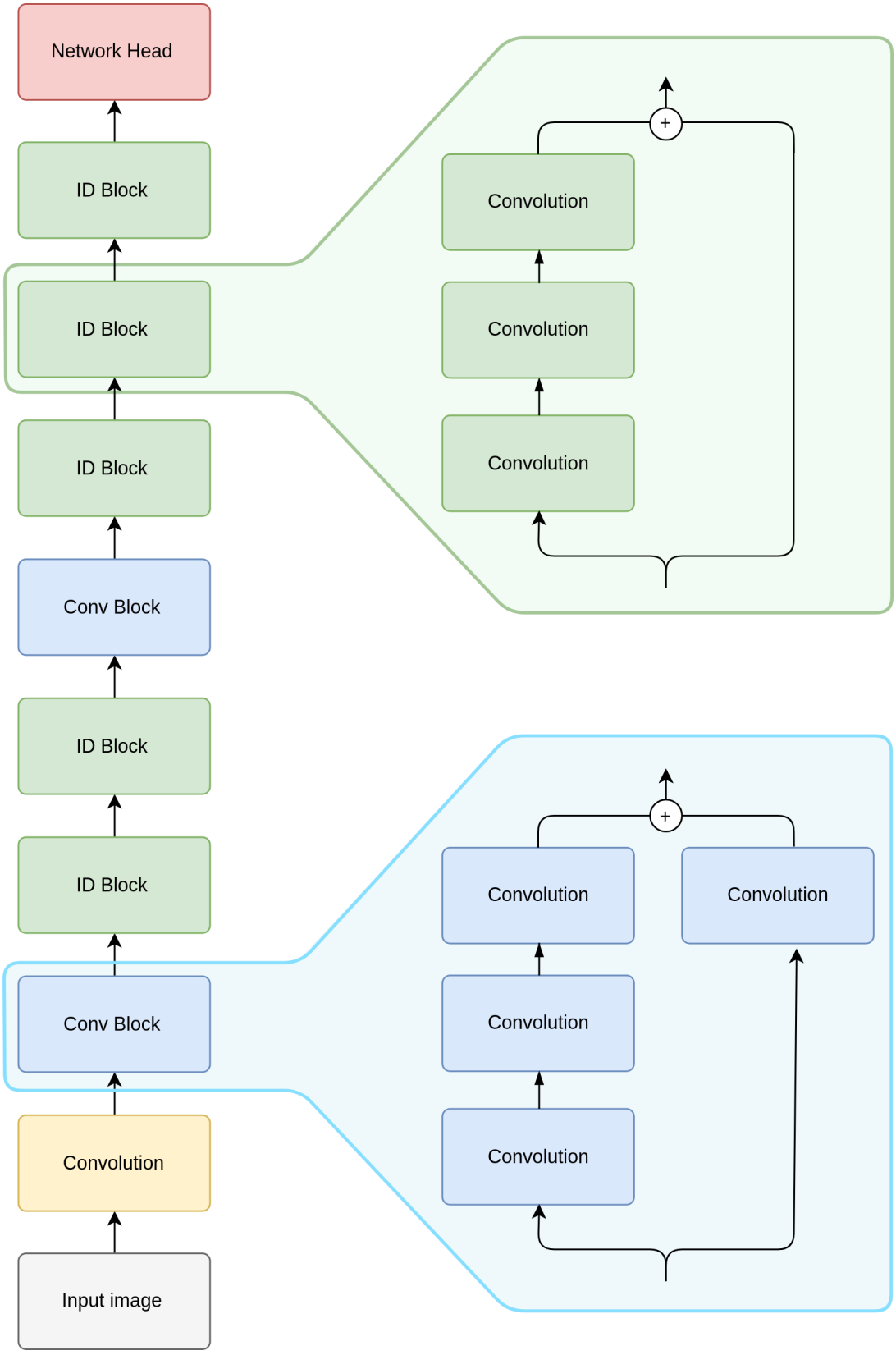


Figure 5.3: Architecture 1 with a total of 23 (convolutional) layers. The components of the **Conv Block** and **ID Block** elements are visualized on the right. The characteristic of the Conv Block is that it applies a convolution on the shortcut connection while the ID Block doesn't. The yellow **Convolution**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

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

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

5.3.2 VARIATIONS

To find the optimal network structure a total of 6 network architectures were created and later compared in Chapter **??**. The first design of the network consisted of 23 convolutional layers with a receptive field-size of 99. A characteristic of the Endoscopic Vision Challenge dataset is that the tools have writings on them which is often not or only partly visible. To keep the network from detecting poses mainly based on the letters, we reduced the receptive field-size for the following architectures. We enlarged the second architecture to a total of 35 layers, while decreasing the receptive field-size to 67. The third architecture has the same number of layers like the first architecture but a reduced receptive field-size of 51. Architectures 4 to 6 all have a receptive field-size of 59 and the 50 convolutional layers. While architecture 5 uses batch normalization for regularization, the architectures 4 and 5 use dropout instead. The difference between those two is the dropout percentage. An overview over the architectures can be seen in Table **??**.

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

The reason why such diverse architectures were compared is that a focus of this work is to provide a prediction system that already produces satisfactory results with few images. A network with a higher number of trainable parameters can take longer to learn the training data and might overfit to it. A shallower network might produce worse results after the first training, but subsequently offer a better accuracy as soon as more data is available.

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

5.4 SETUP

Before the user can run the network to train on a dataset, some steps to setup the network have to be undertaken. Depending on the format of the file that holds the ground-truth poses of the images of the dataset, a conversion to the format expected by the network needs to be performed. An example conversion function is provided, which converts the T-Less format to the JSON format that the network is able to read. The T-Less dataset does not provide segmentation masks, nor object coordinates. The mask and coordinate images can be rendered using the images, the ground-truth poses file and the utility scripts of the network. After obtaining all the necessary data for training, the user needs to write a JSON configuration file, which holds

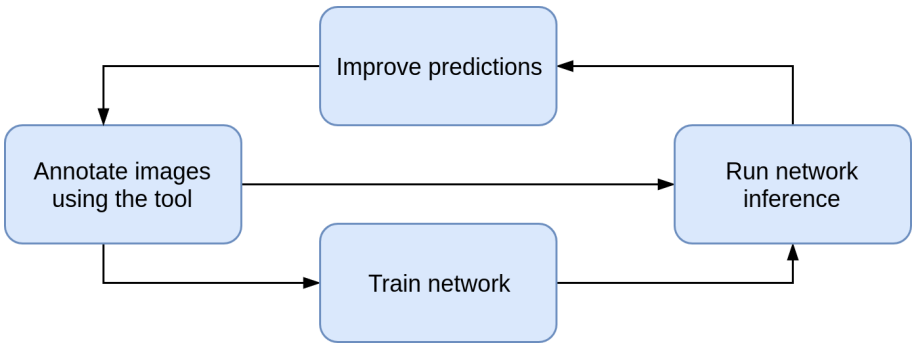


Figure 5.4: The new workflow of the annotation procedure including the neural network. First, the user has to annotate enough images from the dataset to be able to train the network. After successfully training the network it can be used to predict poses on a subset of the dataset. The user can then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

the paths to the data and the parameters of the network. The network architecture that is to be trained can be named directly in the file. The network consists of convolutional, batch normalization and pooling layers only. This means it can be applied to arbitrary image sizes without having to retrain the network, as in contrast to networks employing fully-connected layers. If the image size changes, a fully-connected layer has to be retrained. The maximum image size needs to be specified in the configuration file, nevertheless. This is necessary, because the network is trained in batches instead of single training examples. Keras expects the samples to all have the same dimension. The size in the file can be simply increased if the user wants to run inference on larger images.

5.5 MODES OF OPERATION

The goal of the neural network is to make the annotation process presented in chapter ?? 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. ??. In addition to manually annotating images, the user can train the network with the annotated data. After training the neural network can be used to predict predict poses in unseen images. As the network will likely make errors, the poses can be corrected afterwards using 6D-PAT. The improved pose predictions can serve as input to further train the network. A characteristic of the Endoscopic Vision Challenge dataset is that there aren't existing any annotated poses. Training the network is thus not as straightforward compared to a fully annotated dataset. The following sections describe different approaches of training the network, which are evaluated in Chapter ??.

5.5.1 ONLINE LEARNING

During the annotation process, the user can train the network using the new data. One option is to fully retrain the network with all annotated images. This procedure is called *training from scratch*. Another option is to train the network incrementally, instead. *Incremental training* means that the user trains the network using the only the annotations added after the last training run. In this approach, the weights of the network are not initialized to 0 or randomly (only for the first run), but the weights of the last training run are loaded and trained further using the newly available data. Because the weights are already set meaningful values, this method is likely to converge faster. The next chapter experimentally compares those approaches.

5.5.2 ACTIVE LEARNING

To help the user decide which images annotate, a procedure can propose images for annotation. The proposals should be based on inaccurate predictions of the network. Correcting these predictions can help the network to improve its accuracy. This method requires a measure of the accuracy of the predicted poses. We provide an example measure in the next chapter.

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 [?].

The following section describes the setups of the experiments conducted with the neural network presented in chapter ??. 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 ?? 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 ??: training from scratch and incremental training.

6.1 TERMINOLOGY

Hyper Parameter. *Hyper parameters* are the tunable parameters of a network. The number and kind of parameters 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 ??). 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 [?], [?] and [?]. 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. [?]. 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. ?? shows an example frame from the training set and fig. ?? 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.

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

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

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. There are no other alterations made to the data.

6.4 TRAINING EXPERIMENTS

This section describes the configuration of the different training experiments. The chronological order of the experiments is reflected here. First, we compare the SGD and Adam optimizer in **??**. Then we evaluate the different architectures against each other in **??**. Finally, asses the two training strategies training from scratch and incremental training in **??**. We conducted all experiments on the object model 01 of the T-Less dataset. The image dimension was set to 500 pixels (for width and height), which is larger than the largest area covering an object in any image.

6.4.1 OPTIMIZERS

To obtain the best optimizer for further experiments, we compared SGD and Adam using the first model we constructed. This architecture consists of 23 layers and has a receptive field-size of 99 per output pixel. β_1 and β_2 of the Adam optimizer were left at the default values set by Keras, which are 0.9 and 0.999, respectively. The more complex configuration of the SGD optimizer is given in Table **??**. 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 LOSS FUNCTIONS

6.4.3 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. **??**.

6.4.4 ONLINE LEARNING

6.4.5 ACTIVE LEARNING

6.4.6 RUNTIME ANALYSIS

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

6.4.7 LIMITATIONS

7 DISCUSSION

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

Appendices

A NETWORK ARCHITECTURES

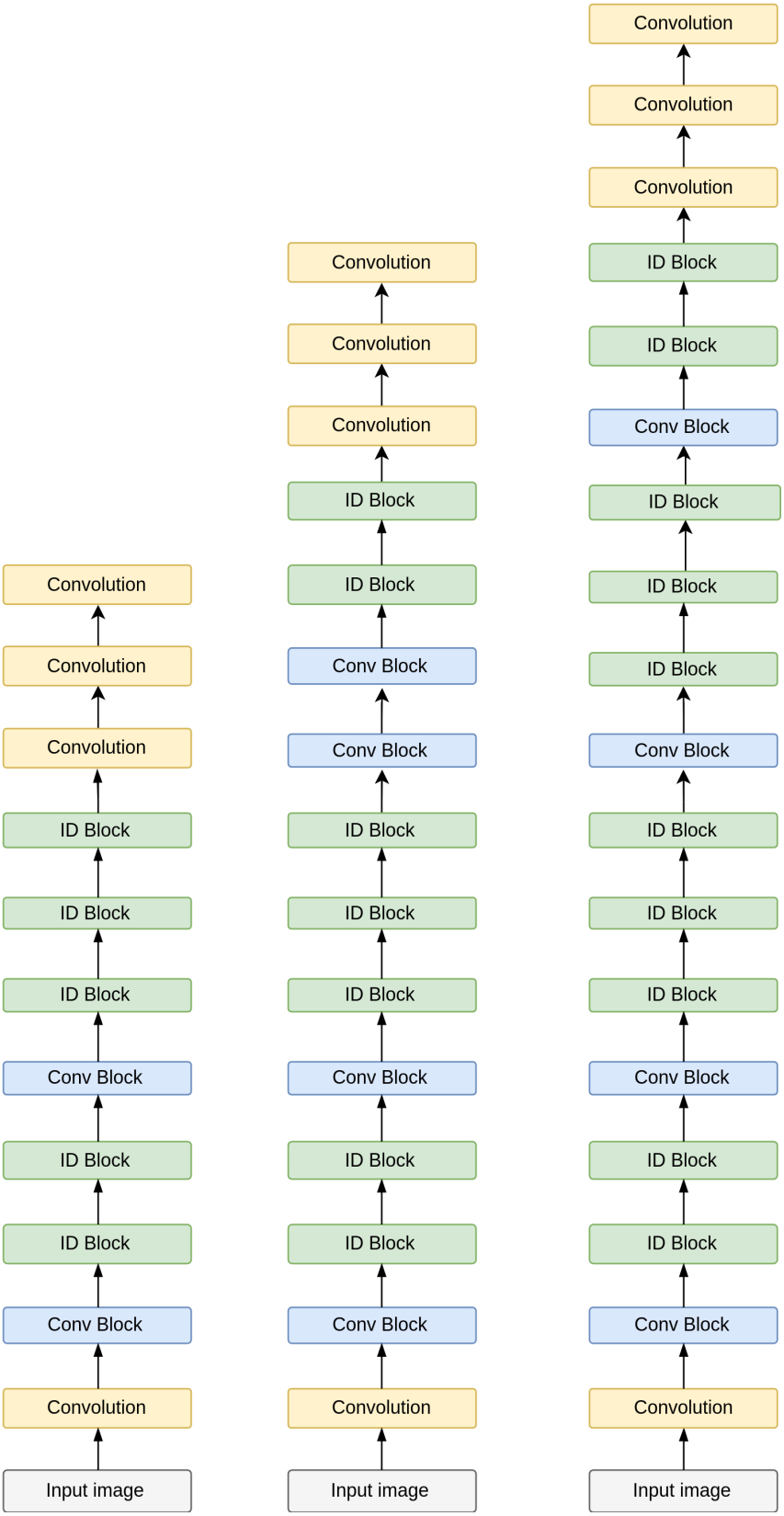


Figure A.1: The different architectures used in this work. Architectures 1 and 3 use the left-most structure with a total of 23 convolutional layers. We constructed architecture 2 using the layout in the middle with 35 layers. Architectures 4, 5 and 6 use the 50 layers structured as shown on the right,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

B EXPERIMENTS

LIST OF FIGURES

LIST OF TABLES