University of Heidelberg
Institute of Computer Science

Project Report

Advanced Machine Learning

Weakly-Supervised Surgical Tool Detection and Localization

---

by

Fabian Schneider
4017026

Konrad Goldenbaum
4076027

faberno/SurgicalToolLocalization

October 4, 2022

Abstract

Author: Fabian Schneider
Surgical Tool Localization is a discipline with lots of useful applications. In traditional surgeries it could be used to evaluate the work of a surgeon and to find possible mistakes, while it could also be applied to the vast amount of already existing surgical videos, that then in return could be used to train even more complex systems like Surgical Workflow Analyzers. Intuitively, such a system would be trained fully supervised with lots of spatially annotated image or video data, which however is infeasible because of the sheer amount of images that would first have to be processed by humans and least be checked by additional professionals. Instead we will show that it only takes binary tool presence labels - which are relatively easy to generate, even for lots of data - to train a Fully Convolutional Neural Network to accurately localize surgical tools. We solved the problem using a pretrained convolutional neural network as a backbone and a convolutional layer as the head of the network, in order to localize objects on a heat-map.

Contents

1 Introduction

1.1 Motivation

Author: Konrad Goldenbaum
Reliable surgical tool localization is a milestone on the way to applications that enable assessments of surgical performance and efficiency, identifications of the skillful use of instruments and choreographies as well as planning of operative and logistical aspects of surgical resources. In addition, the ability to automatically detect and track surgical instruments in endoscopic videos paves the way for much more complex problems such as computer-assisted or even fully automated surgery. This could foster advances in the field of telemedicine, for example by taking on simpler parts of surgeries fully automatically, or by making the automatic delivery of instruments more fluid. These advances are only possible with the help of large data sets indicating not only the presence but also the location of instruments. This is a difficult task because it is tedious and time consuming to obtain the annotations needed to train machine learning models, because the bounding boxes in videos have to be labelled frame by frame so far, and for a large number of surgical tools and operations. In addition, annotators need to be trained to keep up with innovations in surgical instruments. Automated localization and classification of surgical tools makes it possible to create higher-quality datasets cost-effectively by allowing a model to learn to classify and localize instruments based on a weakly annotated dataset. Weakly annotated in this context means that for each clip only the presence of the respective tools is given.

1.2 Structure

Author: Konrad Goldenbaum
In a first part we will describe the problem in more depth. After that we will give a short introduction into existing literature, both on image-classification as well as localization tasks, with a focus on papers that cover similar problems like the one at hand. Connected to this introduction into existing literature we cover ML-tools that should be known in order to better follow the actual content of the work. In *Methodology* (3) we firstly dive into the early considerations and present our first approach, since this allows us to better illustrate the challenges and discoveries made in the process. Additionally we describe the data-management-process. We then explain the model we finally developed and the training-strategy. We finish naming our quality metrics and success-definition. The perhaps most illustrating part, the experiments-section, helps explain the design-decisions we made on the way. Additionally, we describe in greater depth the three datasets we used. We close encouraging further work in the field.

2 Background and Related Work

Author: Konrad Goldenbaum

2.1 Problem Description

The problem we strived to solve was classifying and localizing surgical instruments in endoscopic images, and in extend in videos as well. To that end we registered to the challenge *SurgToolLoc - Surgical tool localization in endoscopic videos*, hosted by *Grand Challenge* in the course of the 25th International Conference on Medical Image Computing and Computer Assisted Intervention (*MICCAI*), that published their training-data on the 29th April, 2022 and excepted last entries until the 8th September, 2022. The data contains 14 possible classes and is composed of weakly annotated videos of surgical training exercises, which means that the annotation is not true for every frame, and no localization ground truth is available (more on the *SurgToolLoc*-dataset in section 4.1.1). Due to technical difficulties and slow responses on the side of the organizers, we were unable to test our localization-performance on the test set. For that reason we trained further models on the *Cholec80*-dataset(7 classes) that does not come with bounding-boxes, and a subset of it, the *M2CAI16*-dataset (7 classes) that is equipped with bounding boxes, which makes comparisons of localization average-precision of our model to state-of-the-art-models possible (detailed descriptions in section 4.1). In a nutshell, the demands came down to the following:

- classifying up to 3 out of 14 tools in one image (multi-label-classification)
- localizing the tools
- training only with weakly-annotated training-data
- (mostly) without bounding boxes and the like
- ability of the model to perform on multiple datasets, proving flexibility of the architecture.

2.2 Related Work

Visual recognition with only weak annotations is a popular challenge, for one main reason: The data-availability increases strongly, when expensive annotations are no longer required, thus enabling more domains to be unlocked. Since 2015 it appeared that localization can be interpreted as merely a by-product of image-level-classification challenges (Oquab et al., 2015) by using a fully-convolutional approach with only some pooling in the end to assign

class levels. This is already very close to our approach, and the method was applied to the COCO-dataset as well as the Pascal VOC 2012-dataset, both of which have more classes (80 and 20 respectively), and use backbones as well. This development has been going on since then, with progress in image-segmentation (Zhou et al., 2018), eventually sparking over to medical applications (Vardazaryan et al., 2018). In all cases, backbones trained on different domains have been applied successfully, meaning that localization or segmentation-tasks were build on efficient feature-extraction. There are alternative approaches where the pre-training-phase has not been "outsourced": Ross et al. (2018) showed that training a network on an auxiliary task, in their case a GAN that was supposed to recolour surgical images, proved very successful as a pre-training-technique, outperforming common backbones, in order to automate image-annotation of endoscopic images. Other developments, that are beyond our scope, are tools that can recognize phases of an operation using long short-term memory (LSTM) units, that have feedback-connections (Nwoye et al., 2019), or networks that utilize kinematic information for segmentation tasks, thereby avoiding further annotations altogether (da Costa Rocha et al., 2019).

On the other hand, self-supervised-tools gain traction, that work in the absence of external labels using en- and decoders to extract visual cues (Ramesh et al., 2022). However, they still appear to remain an unusual choice and hard to train.

2.3 ML-Tools

Introduction of relevant machine learning tools.

2.3.1 Convolutional Neural Networks

In contrast to fully connected neural networks, convolutional neural networks (CNNs) exploit spatially local correlation by enforcing a sparse local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume. This means that the hidden layers include layers that perform convolutions using a convolution kernel. As the convolution kernel moves along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This may be combined with other layers such as pooling layers, fully connected layers, and normalization layers. Essentially, CNNs take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input

2.3.2 (Variational) Autoencoder

Autoencoders (AE) can be considered as a pair of neural networks, where the first part (the encoder) translates the input to a lower dimensional latent space. The second part (the decoder) then tries to reconstruct the original input using the the information provided via

the latent space. Variational auto encoders learn a latent distribution (rather than representation), that can be drawn from. The decoder then proceeds as before. One advantage can be, that the variational auto encoder (VAE) avoids overfitting in the latent space. On a side note: The other main advantage of a VAE compared to an AE is that it is better suited for generative purposes.

2.3.3 Backbones

In the image-processing context, backbones are networks that provide feature-maps. Based on these feature maps, one trains a so called "head", that does the intended task of the model. Backbones are usually trained on ImageNet with usual choices being the AlexNets or ResNets.

2.3.4 Data Augmentation and Semi-Supervised Learning

Data augmentation are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. More importantly, it acts as a regularizer and helps reduce overfitting when training a machine learning model. Semi-supervised learning is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training.

2.3.5 Gaussian Process

Gaussian processes can be seen as an infinite-dimensional generalization of multivariate normal distributions.They are useful in statistical modelling, benefiting from properties inherited from the normal distribution. For example, if a random process is modelled as a Gaussian process, the distributions of various derived quantities can be obtained explicitly. In the context of our project we used Gaussian processes to optimize certain hyper parameters in our training.

2.3.6 Sliding Windows Approach

The sliding windows approach means to have a classifier evaluate smaller parts of images as to whether an object is present, to then compute the place of said object in the image. It is computationally expensive, since for every part of the image a forward-pass of the classifier is needed.

3 Methodology

Author: Konrad Goldenbaum

3.1 Early considerations

We started with the initial assumption that the project can be divided into two challenges: Firstly, the classification, and secondly the localization. We found an approach that allows for both challenges to be addressed together relatively early on. Nonetheless we will firstly cover an early alternative approach, because it will help us showcase some inherent difficulties of the challenges that had to be overcome.

We firstly decided to approach this as a problem where the images are, in a certain way, of a too high resolution. We mean this in the sense that there are tools and tissue, with the tissue obscuring the tools. This of course is a rather naive assumption. We followed the approach of He et al. (2015) to use a variational auto-encoder to reduce the images to little more than the tools, since those should be considered the most important parts of the image, or are at least the parts easiest to recognize for the De- and Encoder, since the tools appear more often than the respective tissue-structures. This type of unsupervised learning is appealing, but the output can be limited only to frequently occurring and visually consistent objects (Oquab et al., 2015), which might be the biggest underlying weakness of this approach.

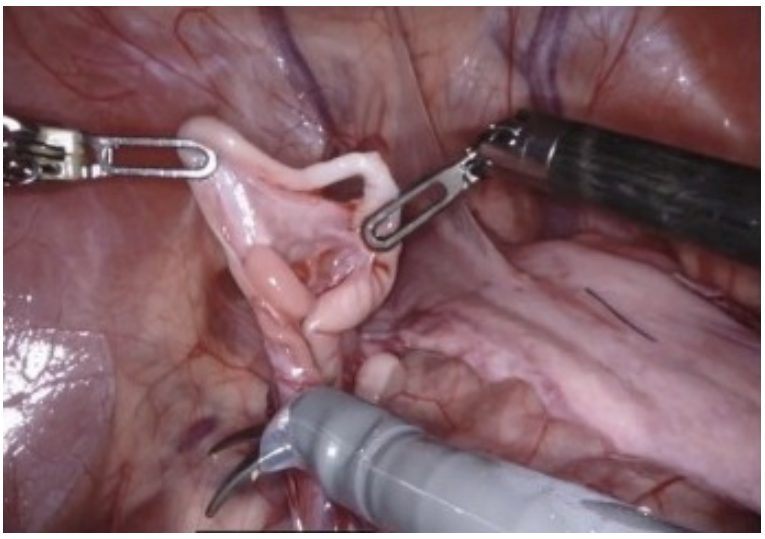


Figure 3.1: As one can imagine the tools do stand out - sometimes...

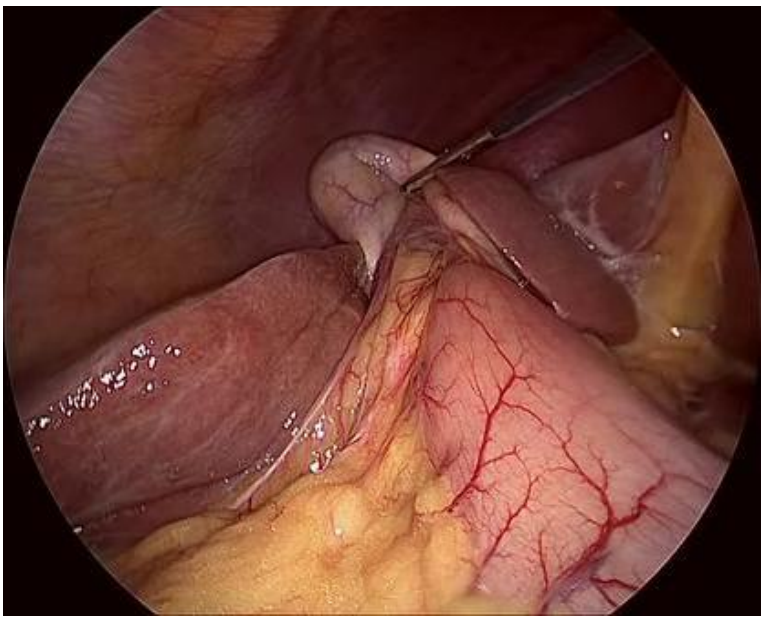


Figure 3.2: And sometimes not so much.

Han et al. (2017) presented a relatively straightforward VAE-model, that we decided to rebuild and adapt to our needs 3.3.

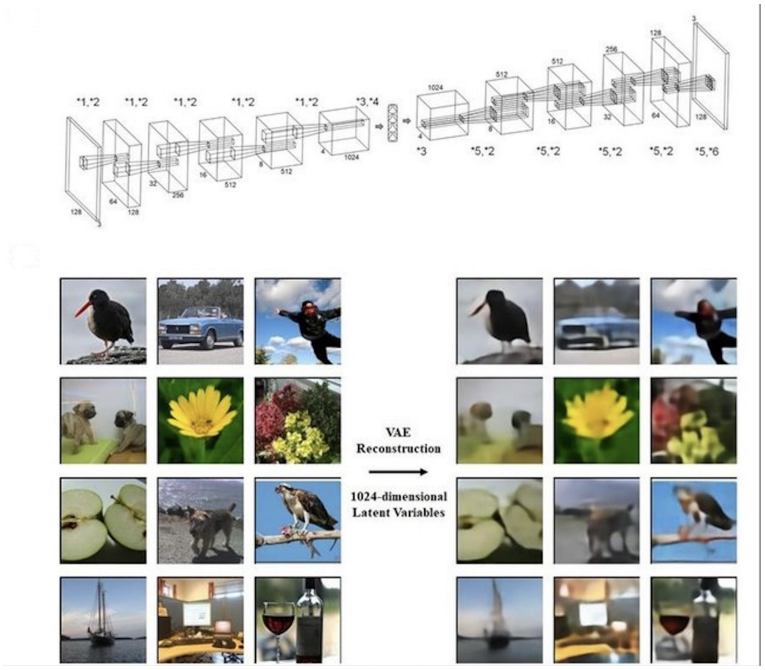


Figure 3.3: Architecture (top) and result (bottom).

Since we don't need any representation that keeps too much of the original image, we mainly trimmed the whole approach down. In the end, so we hoped. the instruments would be the parts that would at least to some extend be "saved over" to the other side of the image. We

also experimented with the latent representation (or distribution, rather), hoping it might encode the information preserving locality, so that the interpretation of the representation gives clues already. For this purpose we settled at around 500 latent variables, since less would not be useful in terms of placing an object within an image.

The more straightforward approach here is of course threading the image through a needle, meaning a rather small latent variable space again, and then analysing the reconstruction. The hope was that the image would preserve well distinguishable blobs where the tools were, so that we could filter the localization of the tools with some old-fashioned blob-detection via Otsus method or similar approaches.

However, although the concept of VAEs is very intriguing, there is still an issue to address: Given a successful localization of objects, how does one classify the image? The challenge is, that even when given only the part of the image with the tool itself, this would still leave us with having to solve a rather unpleasant classification problem, since all the images come annotated with all the classes originally in the image. This problem accompanied us during the whole project. The advantage in this case is, that we would stick with the well-known one-label-classification problem, meaning that ideally only one label is true at any point, although the calculation of the loss would be difficult. We are confident that this problem would have been solvable as well. But searching for a solution for the image-classification-part, we found that there was already a better approach.

3.2 Data management

We build a first prototype of the VAE, but training was impossible at first due to the handling of the huge dataset that we used first, that comprised of over 100GB of image data. More on that in the experimentation section. However, the sheer amount of data was not the main issue, but it points to a different one: We could not simply take the data, split it up into a training-, validation- and test-set and perform SGD on the training-set. This has two reasons: Firstly, the data was very unbalanced (see 4.2). Secondly, and more importantly, we had to somehow make sure that the model would not learn something else, an issue that can be illustrated by some funny and some rather sinister examples.

To this end we flipped the image with a 50% chance, rotated it between -90 and 90 degrees, masked and cropped the images. Masking means that we set some regions of the image to the mean value of the rest of the dataset (Singh and Lee, 2017). The regions were selected in the following way: Any image was divided into 24x24-tiles, with every tile having a 50% chance of being masked. Since this is basically the binomial distribution we can expect that around half of the tiles are masked everytime. This can look like this:

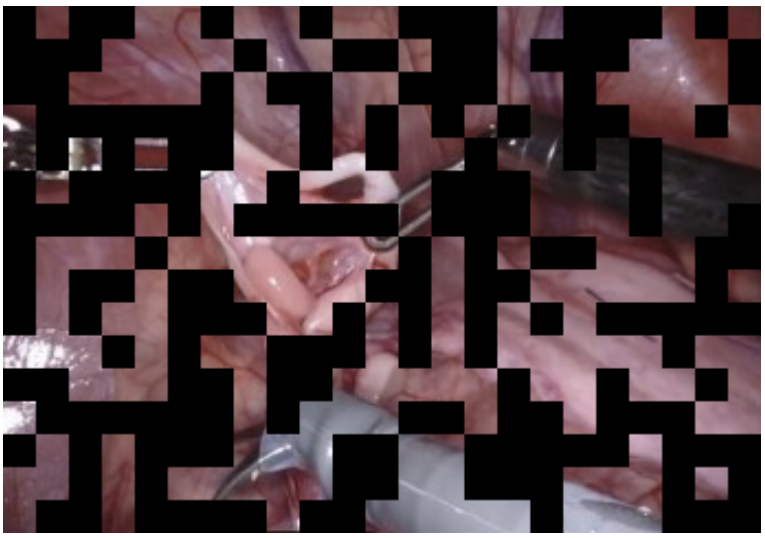


Figure 3.4: Example for a masked image.

Another measure covered in the lecture on data-augmentation was the fact, that some falsely-annotated data put into the training-set might also serve the generalization-abilities of the network. We decided against any such measure, since the multi-classification of the entire clip presented us with more falsely annotated images than we might have wished for.

Having the data handled we went on exploring possible solutions to classifying the images. We therefore turned to CNNs, that, as we covered in the lecture, proved quite useful in image-classification-challenges. One interesting aspect of CNNs like the ResNET-models is that, before reducing the output of the last hidden layer to logits of the class-probabilities, these models preserve some of the size and locality of the input since the transformations are mainly convolutional in character. This means that the ResNets forward-pass can be intercepted, while the locality of the discovered features is preserved. This also means that by computing a fully connected convolution one gets a heatmap for all the tools (Zhou et al., 2018). This procedure is fairly common already (see section 2.2). It is quite obvious that a well-performing network that returns heatmaps of the tools presence renders any other effort to localize tools obsolete. Since this is the concept we eventually pursued, we will cover it in more detail:

3.3 The final model

For the underlying architecture of the final model we followed a straightforward and established approach: We worked with a well-performing backbone to extract features of the image and trained a convolutional layer with kernel size 1x1 based off of those features. This results in a heatmap, that can be further processed. The fully connected convolutional layer thereby replaces the sliding-window-approach, because it essentially performs the work of a fully-connected layer on a small part of the image, but, exploiting the convolutional nature of the layer, more efficiently than iterating with a fully connected layer over parts of the image over and over again.

Using backbones is so successful because although those we used were originally applied to other image-classification-tasks, it appears that until the last few layers, the main work that is being done by the net is extracting features within the image. This leaves the task of "making sense" of these features to the last layer (Vardazaryan et al., 2018). The opportunity therefore lies in the possibility of omitting these last layers, and to train only the last layer for the specific task at hand (see for example the paper by Elharrouss et al. (2022)). As they point out, there are certain networks that have become common choices. We tried and experimented with three backbones, each of which come with different depths: The ResNet by He et al. (2015), the AlexNet by Krizhevsky et al. (2017) and the VGG net by Simonyan and Zisserman (2014). All three of them are readily available via the PyTorch-libraries.

We will cover the advantages of the different backbones more thoroughly in the experiment section, since the use of different nets can be understood as a hyper-parameter to optimize.

The localization and classification of the tools is being determined via the heatmaps: The network returns one heatmap for each class. Firstly, it is being decided via a pooling-layer which classes are present. The pooling functions are considered a hyper-parameter, although the use of an min-max-pooling in the ResNet-architecture left us with a good starting point. After having decided which heatmaps to consider, we eventually chose the peaks in these maps as the place of the object. This differs from the SurgTool-challenge, where they want bounding boxes, but proved easier to compare to existing literature like the extremely helpful paper by Vardazaryan et al. (2018) (section 3.4).

Taken altogether we ended up using a ResNet-18-backbone with stride 2×2 with a 1×1-kernel convolution-layer to generate the heatmaps for the classes and min-max-pooling for classification.

3.4 Training

The training was carried out using mini-batch stochastic gradient descent with different batch sizes, ranging from 2 to 50. We monitored both training and validation losses as well as average precision and accuracy. After every iteration over the dataset we validated the training progress on a completely separate set. As mentioned earlier, contrary to what we were used to we looked at a multi-label-classification problem. This had to be taken into account when choosing a loss function. We went with the multi-label soft margin loss, a weighted cross-entropy loss presented in equation 3.1, where k_c and v_c are the ground truth and predicted tool presence for class c, σ is the sigmoid-function and W_c is the weight for class c. The weights serve as a counter-measure against the effect of class-imbalances (discussed in section 4.1).

$$Loss = \sum_{c=1}^{C} \frac{-1}{N} [W_c k_c log(\sigma(v_c)) + (1 + k_c) log(1 - \sigma(v_c))] \tag{3.1}$$

3.4.1 Training workflow

Training a neural network is challenging and resource-draining. Therefore developing a code base and workflow to overcome these problems is not a trivial task and should be considered part of the overall effort.

We used ColabPro to compute the training. Doing so means that both the data and the code-files must be uploaded into a Google-Drive-account and then be run using
%run train.py -...
This implies one main requirement, namely that the hyper-parameters should be easy to change without touching the code base. We did this using configuration-files. They have been implemented to choose different compositions of backbones, convolutional layers and hyper-parameters like the strides of the convolutional layer. We also implemented means of saving the trained parameters and resuming the process at any time, even keeping a history of the different states, as well as utility-functions that kept us up-to-date about the training progress. More on these in the quality metrics-section.

3.4.2 Hyper-parameter tuning

We tuned the following hyper parameters: We did so by training 16 combinations for each 50 epochs on a smaller dataset, during each epoch we collected the quality metrics of each for each parameter-combination and epoch. Based off of these data points we ran a gaussian process by iterating over a much tighter net of hyper parameters, searching for a combination that appeared promising in terms of further training. We picked the gaussian process from the sklearn-library and went with the recommended parameters. The main difficulty that emerged using this process is that we had to assign a value to the data points that we collected for each combination of parameters. It seems important to take not just the best-performing model at any epoch, but also a model that shows continued improvement as well as overall good performance. We tried different approaches, but stuck to a simple one by calculating the mean and adding the weighed minimum to it.

Apart from that we just had to try some options intuitively, especially which pooling-function or backbone to use. This will be one main part of the experiment section.

3.5 Quality metrics

We used the precision-recall-curve, the F1-score (that merely combines the precision- and recall-values), the average precision, the average distance between a bounding box centre and the closest correct predictions in percent of the diagonal length (for localization). The average precision metric was also used to interpret the localization performance. All these metrics were applicable, since we took care in balancing the classes, which is, at least for the classification part, an important prerequisite. Some of the metrics provide redundant information (especially true for the F1-score and the precision-recall-metric), although some of them are easier to interpret on the spot. But it does not seem to hurt to in doubt calculate more metrics than necessary. When it comes to assess the quality of the model, we mainly looked towards papers that covered similar problems. For instance, Vardazaryan

et al. (2018) used average precision to evaluate the binary tool presence classification as well as the localization. We not only looked at these papers for inspiration, but also as a benchmark we wanted to achieve as well, meaning that average precision of over 90% for all tools was the overall goal (as achieved by Vardazaryan et al. (2018), Twinanda et al. (2016) and Nwoye et al. (2019)).

4 Experiments

Author: Fabian Schneider

4.1 Datasets

In the course of this project, we used 3 different datasets:

4.1.1 SurgToolLoc

The dataset of the *SurgToolLoc* Challenge is provided by the *International Society for Computer Aided Surgery* (ISCAS). Its a result of surgical training exercises using the *da Vinci* robotic system and contains 24,695 video clips, each 30 seconds long, captured at a resolution of 1280×720 pixels and 60fps, resulting in 1800 frames per video and almost 45 Million frames in total. Each video contains 3 out of 14 different tools (4.1), of which not always every tool is active and visible, resulting in a certain amount of label errors. While this training dataset is only annotated with video-level tool labels, there exists a non-public test dataset additionally annotated with bounding boxes around the robotic tools. Sadly, we could not get access to this test dataset, which is why we will only test the tool classification on it and not the localization.

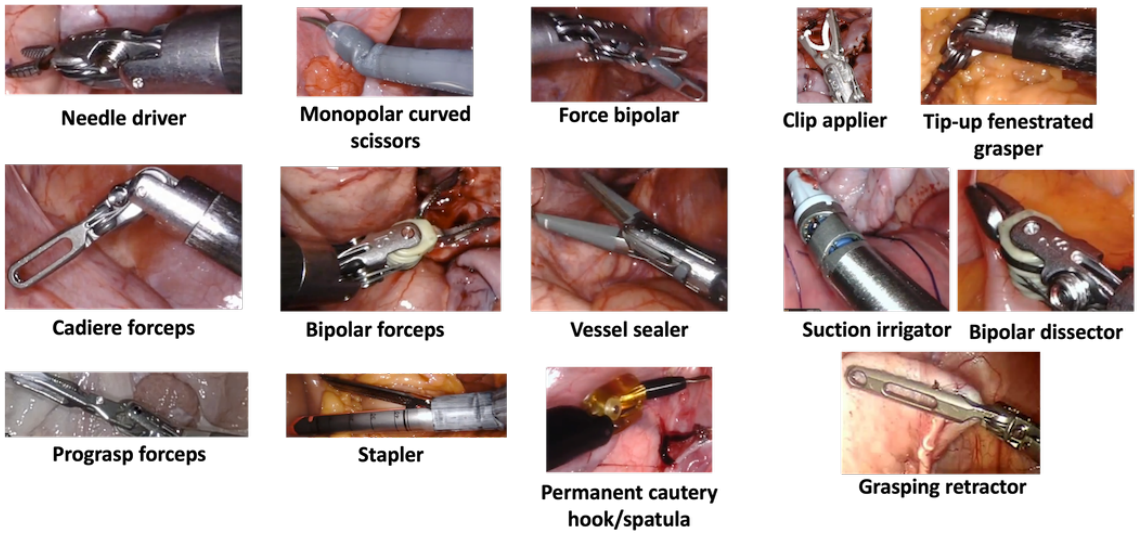


Figure 4.1: All 14 tool classes of the *SurgToolLoc* dataset.

As training on such a massive dataset would have been unfeasible for our project, we created subset containing 10,000 frames, randomly sampled from all videos and downscaled to 900×620 pixels. 7000 of those frames were used for training and 3000 for testing. This re-sampling had the additional advantage of improving the extreme class-imbalance in the original data, where the most frequent class occurred 1000 times more often the least one (4.2).

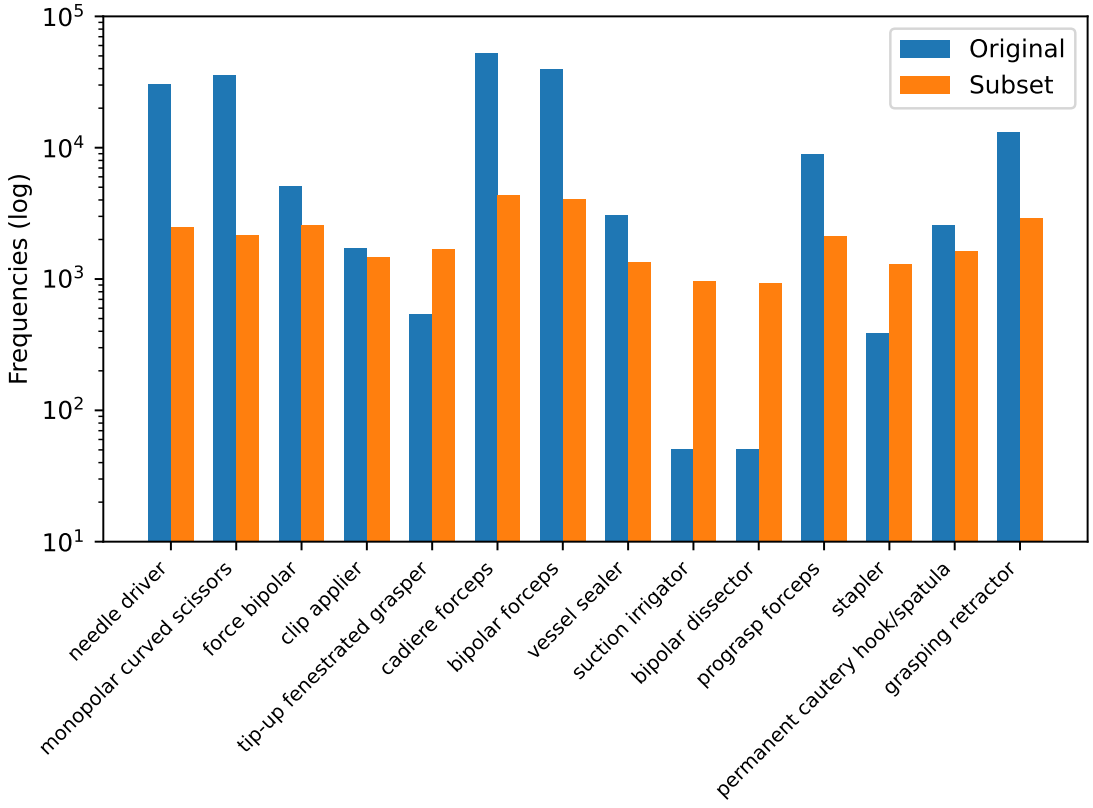


Figure 4.2: Tool frequencies of the *SurgToolLoc* dataset (video-level) and its subset (frame-level).

4.1.2 Cholec80

The *Cholec80* dataset Twinanda et al. (2016) is provided by the *CAMMA* research group at the University of Strasbourg and contains 80 videos of cholecystectomy surgeries, which combined result in about 180,000 frames, most of them captured at resolution of 854×480 pixels. It includes 7 different tool classes (4.3), from which a frame contains 0 to 3 and on average 1.32 instances. In contrast to the *SurgToolLoc* dataset, *Cholec80* is annotated with frame-level tool labels, resulting in much less label errors. Bounding box labels were generated for the publication Vardazaryan et al. (2018), but to our knowledge, have not been published.

Similar to before, this number of frames would have been unmanageable for us, so we again created a subset, of this time 6000 frames, randomly sampled from all classes. While in the

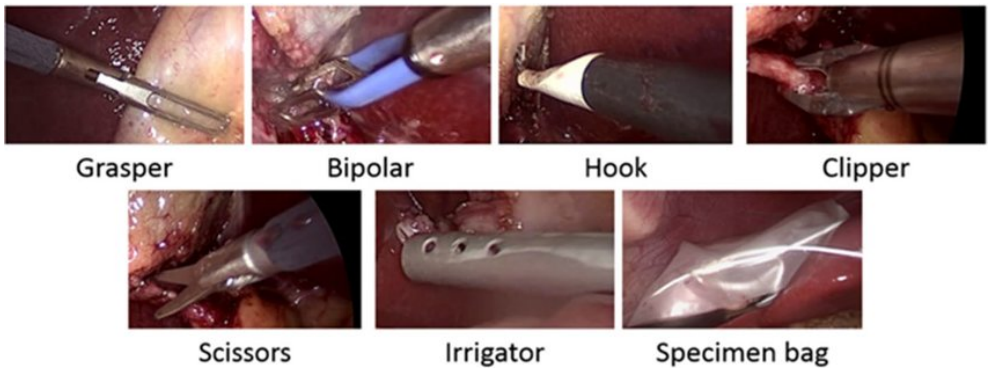


Figure 4.3: All 7 tool classes of the *Cholec80* dataset.

original set, the factor between the least and most appearing tool is about 30, it's only 4.5 in the subset. A complete class-balance would be hard to achieve, as some of the tools, like the Grasper, are used almost exclusively with the other tools together.

4.1.3 M2CAI16

The *M2CAI16* dataset contains 2811 frames (2248 for training and 563 for testing) gathered from the videos 61-76 in *Cholec80*. Every frame is fully annotated with tool presence labels and bounding boxes. Its size makes it manageable for us and its bounding box annotations can be used to evaluate the tool localization, which is why we will mainly use it for the following experiments as a validation set.

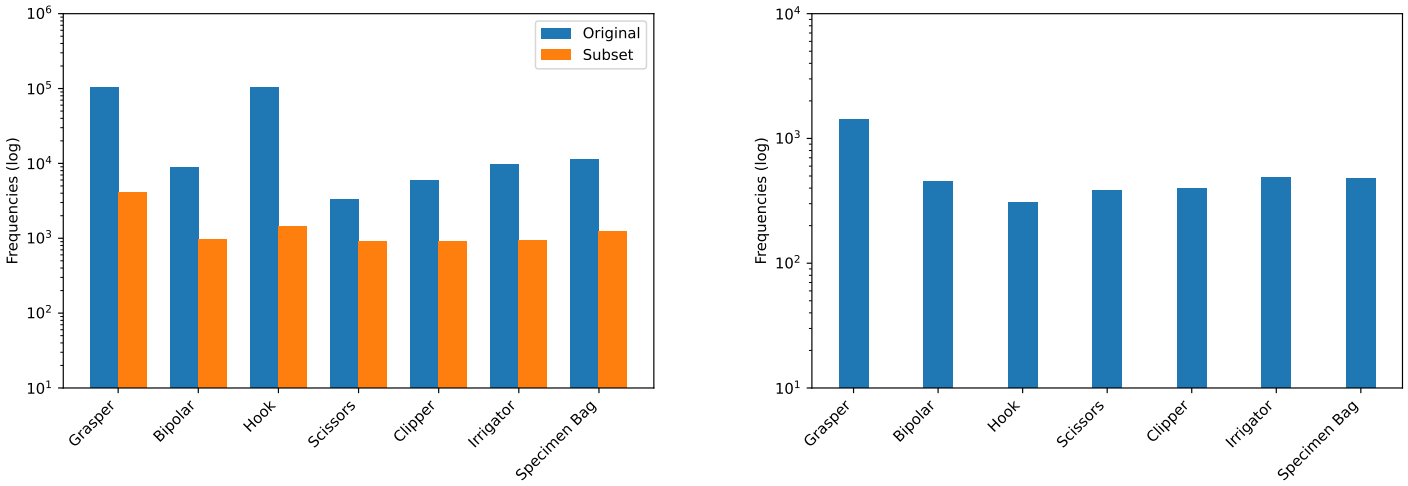


Figure 4.4: (left) Frame-level tool frequencies of the *Cholec80* dataset and its subset (right) Frame-level tool frequencies of the *M2CAI16* dataset

4.2 Experimental Setups

In the following, we will define a set of standard parameters that can the be used as baseline to compare different parameter setups. All of them will have in common, that the model will be a fully convolutional neural network consisting of a backbone, an additional convolutional layer with (1×1) kernel and spatial pooling in the end. The stride of the last two convolutional layers of the backbone will vary. As optimizer we will use Stochastic Gradient Descent with momentum and weight decay. The learning rate will be reduced stepwise by a factor γ and can be specified independently between the backbone (lr_1) and convolutional layer (lr_2). The train loss is calculated with class weights according to this formula

$$w_c = \frac{m}{F_c}$$

where w_c is the corresponding weight of class c, m is the median frequency across all tools and F_c the frequency of c (Nwoye et al., 2019).

4.2.1 Standard Parameters

Vardazaryan et al. (2018) and Nwoye et al. (2019) both worked with the following hyperparameters:

lr_1, lr_2	0.001, 0.1
γ	0.1
weight decay	10^{-4}
momentum	0.9
strides	1, 1
backbone	ResNet18
pooling	WildCat

They chose to train the backbone with a learning rate 100 times smaller than the one of conv. layer to account for its pretraining on the ImageNet dataset.

However, by tuning lr_1, lr_2, γ and weight decay using Gaussian Processes, we came up with the following set:

lr_1, lr_2	0.01, 0.01
γ	0.45
weight decay	10^{-3}
momentum	0.9
strides	1, 1
backbone	ResNet18
pooling	WildCat

As can be seen, lr_1 should start 10 times higher and lr_2 10 times lower, so that they start at the same value, with the reduction factor γ being higher, too. Also, the L2-Penalty in form of the weight decay, is recommended to be 10 times higher. The impact of this new parameters can be seen in 4.5 and 4.6.

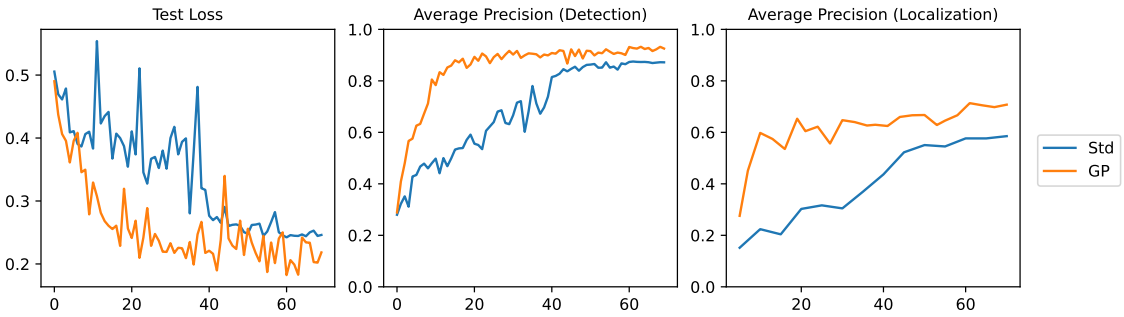


Figure 4.5: (standard vs. optimized model) Progress of Test Loss and AP over epochs

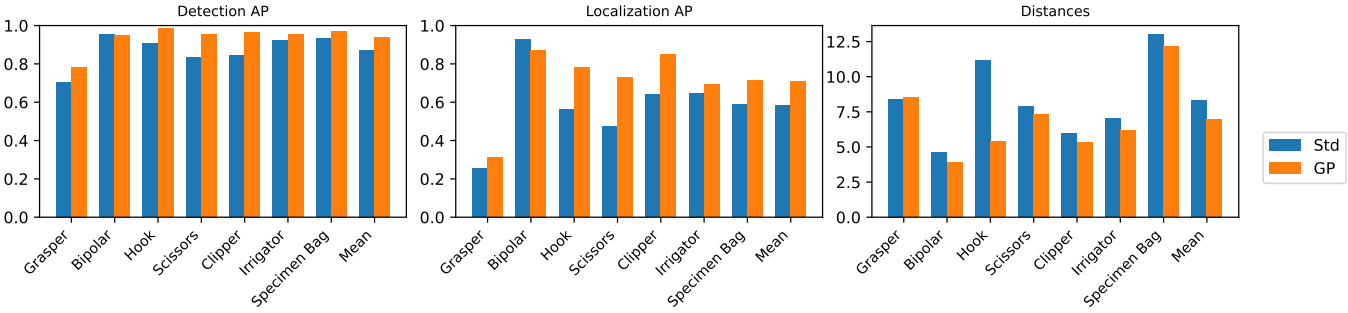


Figure 4.6: (standard vs. optimized model) left: Classwise Average Precision of detection, middle: Classwise Average Precision of localization, right: Average distance between bounding box center and predicted position in % of the image diagonal

What we can see in 4.5, is that one hand the model with optimized parameters seems to train a lot faster and more stable. While the test loss of the standard model sinks slow and with large peaks in between, the optimized one sinks very fast and converges then. The AP of the detection already reaches its maximum at about 30 epochs, while the other one achieves that after about 50. This is a major improvement for us, as our computing resources or scarce and training one model takes several hours. Because of this improvement we feel comfortable to lower the number of epochs to 50 instead of 70, as not much progress is made anymore after this point.
On the other hand, the optimized model also achieves a noticeable higher AP by about 6 percentage points for the detection and almost 12 for the localization. Figure 4.6 shows us these differences for each class. It is noticeable that in both models, the *Grasper* has much lower scores than the other tools. We guess this is the case, because the *Grasper* is often very small and easily confused with other artifacts, like seams or dried blood. The *Bipolar* on the other hand achieves very high scores, probably because of its unique shape and color. Besides an improvement in the AP, we can also see that the distances between the bounding boxes and peaks reduced noticeably, by about 1.5 percentage points of the image diagonal and even halved for the *Hook*.

In **future experiments**, we will use this optimized parameter set for our standard model.

4.2.2 Strides

In the next experiment, we will evaluate the influence of the stride values in the last two conv. layers of the backbone on the models performance. For that we compared our standard model of stride (1,1) to two different ones of stride (1,2) and (2,2). They result in the following heatmap size:

Strides	Heatmap size
(1,1)	44×76
(1,2)	22×38
(2,2)	11×19

Note: With stride (x,y) we don't mean stride x across axis 0 and y across axis 1, but rather, (x,x) in the second-to-last conv. layer of the backbone and (y,y) in the last.

Vardazaryan et al. (2018) and Nwoye et al. (2019) chose (1,1) to increase the accuracy peaks in the heatmaps. We can see in 4.7 and 4.8, if this worked for us, too.

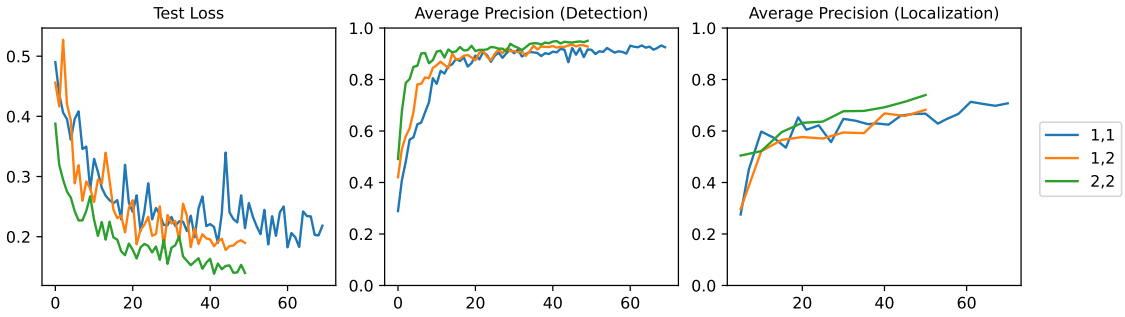


Figure 4.7: (Stride (1,1) vs (1,2) vs (2,2)) Progress of Test Loss and AP over epochs

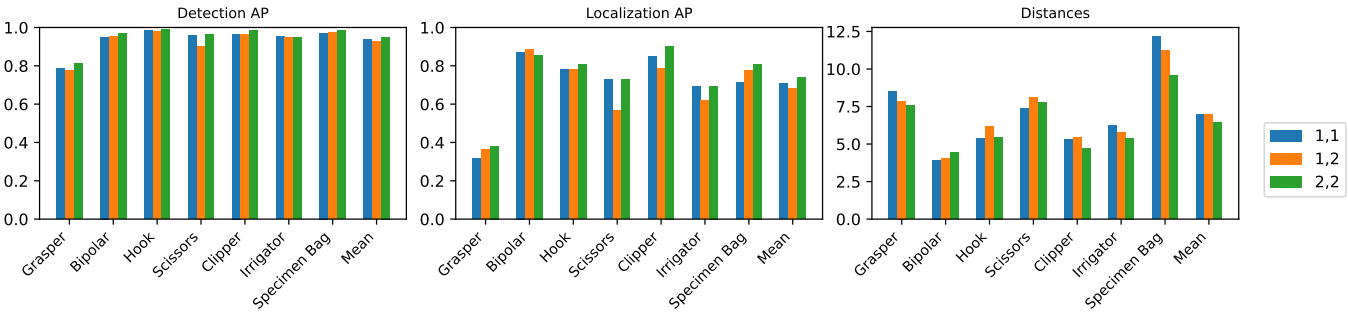


Figure 4.8: (Stride (1,1) vs (1,2) vs (2,2)) left: Classwise Average Precision of detection, middle: Classwise Average Precision of localization, right: Average distance between bounding box center and predicted position in % of the image diagonal

As can be seen in 4.7 using strides (2,2) brings again a noticeable improvement over the current standard model of stride (1,1) and that in every regard. The test loss converges faster and more stable, while the maximum AP lies higher and is reached faster.

In 4.8 we see, that especially the *Specimen Bag* benefits, as its distance sunk by almost 3 percentage points, which in return increases its localization AP.

4.2.3 Backbones

While Vardazaryan et al. (2018) and Nwoye et al. (2019) chose a ResNet18 as their backbones and Zhou et al. (2018) a ResNet50, we wanted to find out if another of the popular classification networks could bring further improvements. We will compare the already mentioned ResNet18 and ResNet50, against a AlexNet and VGGNet16. The results are seen in 4.9 and 4.10.

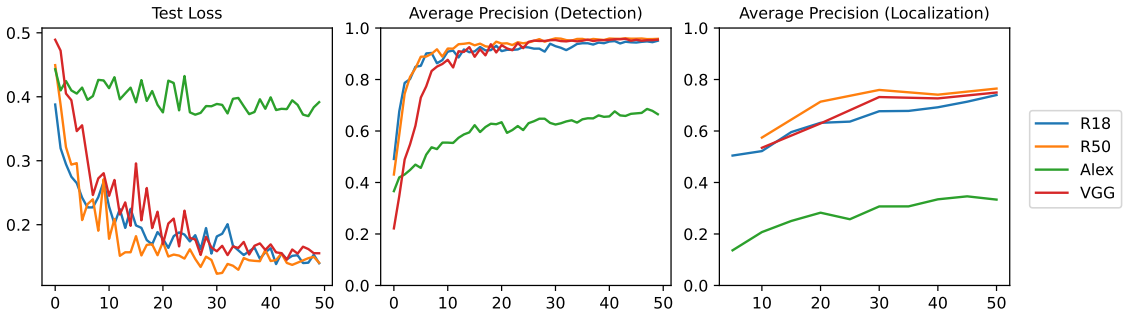


Figure 4.9: (ResNet vs AlexNet vs VGGNet) Progress of Test Loss and AP over epochs

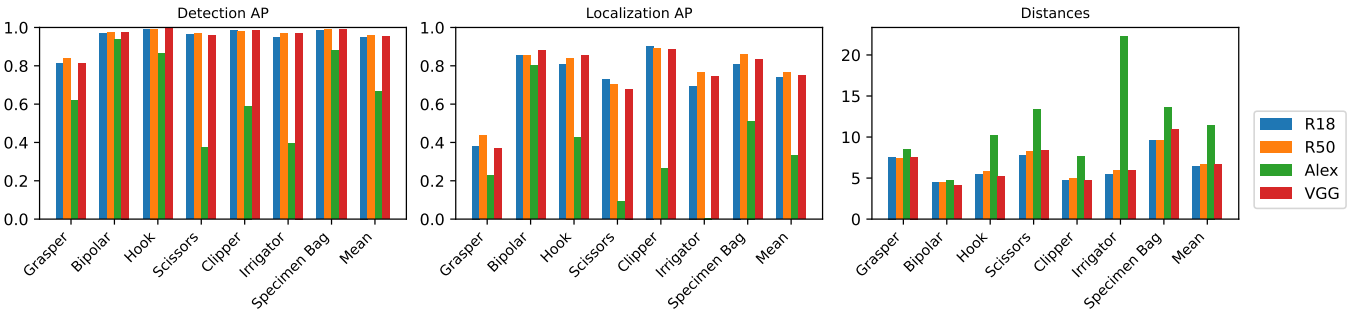


Figure 4.10: (ResNet vs AlexNet vs VGGNet) left: Classwise Average Precision of detection, middle: Classwise Average Precision of localization, right: Average distance between bounding box center and predicted position in % of the image diagonal

Probably the most obvious result is, that the AlexNet performs a lot worse than all other 3 backbones. The reason for this is probably, that it is much simpler and and smaller than even the ResNet18, and is not able to learn as complex as the others. Interestingly, all other backbones perform roughly the same, even though ResNet50 and VGG16 are more complex networks, than the ResNet18. As they also require significantly longer training time and more memory, while giving barely any benefits, we will continue to use the ResNet18.

4.2.4 Spatial Pooling

While Vardazaryan et al. (2018) examined the impact of WildCat (Durand et al., 2017) instead of normal MaxPooling and showed that it performs better in most situations, we will instead compare WildCat to PeakResponse Pooling, which does not only take the highest peak in the heatmap into account, but also all the others with respect to their value.

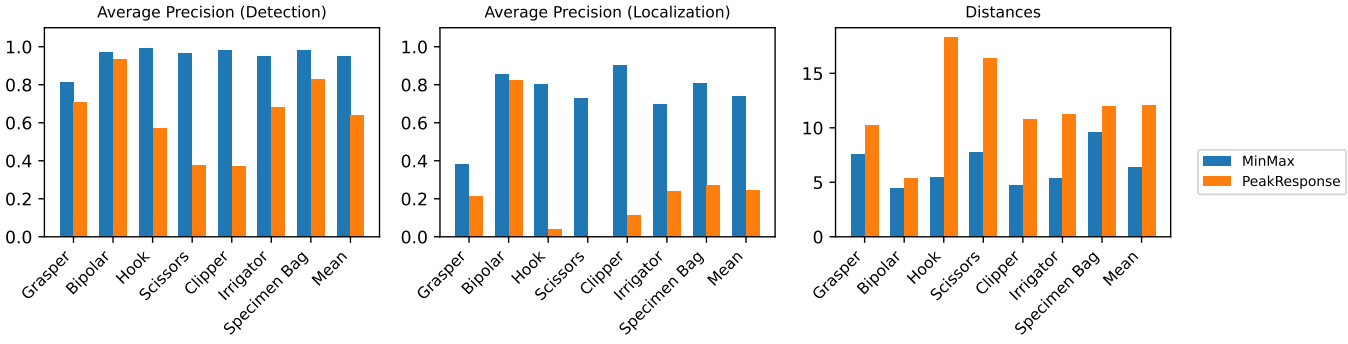


Figure 4.11: (WilCat (MinMax) vs PeakResponse Pooling) left: Classwise Average Precision of detection, middle: Classwise Average Precision of localization, right: Average distance between bounding box center and predicted position in % of the image diagonal

It stands out, that for every tool, except the *Bipolar*, the PeakResponse Pooling performs significantly worse. This is surprising, as according to Zhou et al. (2018) it should actually perform better. Even though we reused the original implementation by the author, it is still possible, that this results occur due to a bug in our implementation.

4.2.5 Pre-Segmentation

Another small change we hoped could improve our results, was to pre-segment a small number of images from the train dataset, by roughly deleting all parts of the image, that were of no interest for us (4.12).

Our idea was that we have three pre-segmented images for every class (working time: 20 minutes), that we train as one batch after every epoch of training. This should help our model to find the right tool, respectively the correct region of the tool. Our results can be seen in 4.13 and 4.14.

Sadly, as can be seen theres barely any difference between the two results, so we can assume that pre-segmentation with such a small amount of images does have no impact.

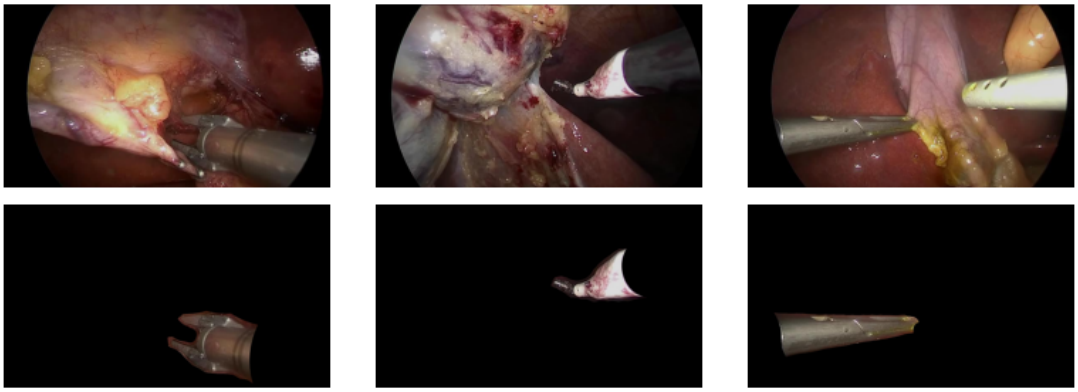


Figure 4.12: A few examples of images pre-segmented images

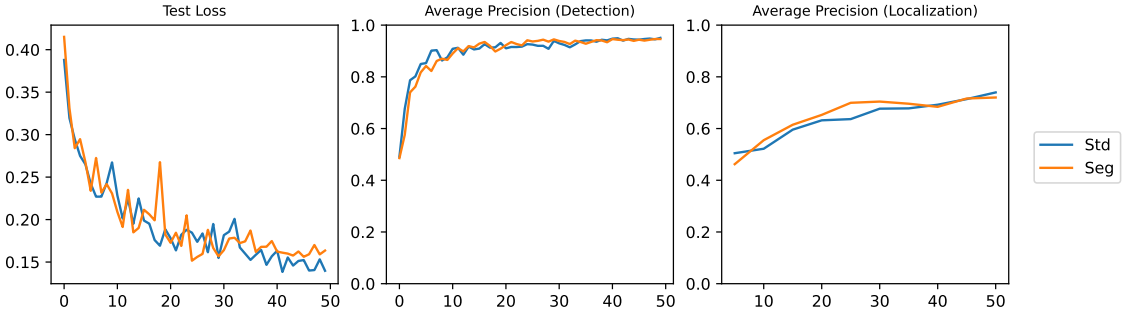


Figure 4.13: (Pre-Segmentation) Progress of Test Loss and AP over epochs

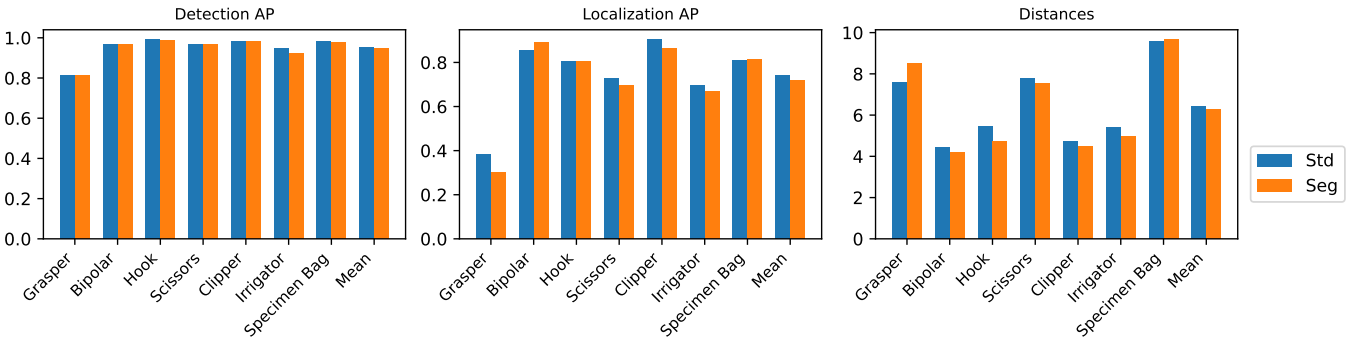


Figure 4.14: (Pre-Segmentation) left: Classwise Average Precision of detection, middle: Classwise Average Precision of localization, right: Average distance between bounding box center and predicted position in % of the image diagonal

4.2.6 ChannelSwitcher

Our last experiment is based on the observation, that our tools are mostly gray, while everything around them has a dominant red channel. Our idea was, that if we randomly switch the color channels of the images during training, the model will be more likely to learn the features of not so much affected gray areas, instead of the strongly affected red areas. As can be seen in the following graphs, there has been no noticable effect.

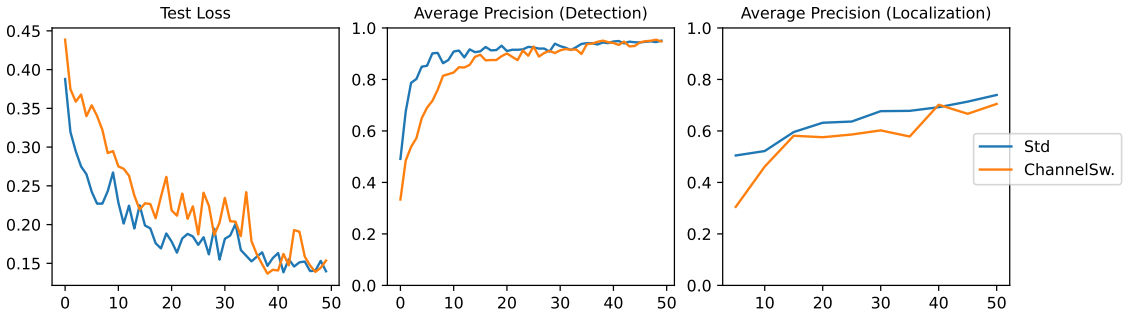


Figure 4.15: (ChannelSwitcher) Progress of Test Loss and AP over epochs

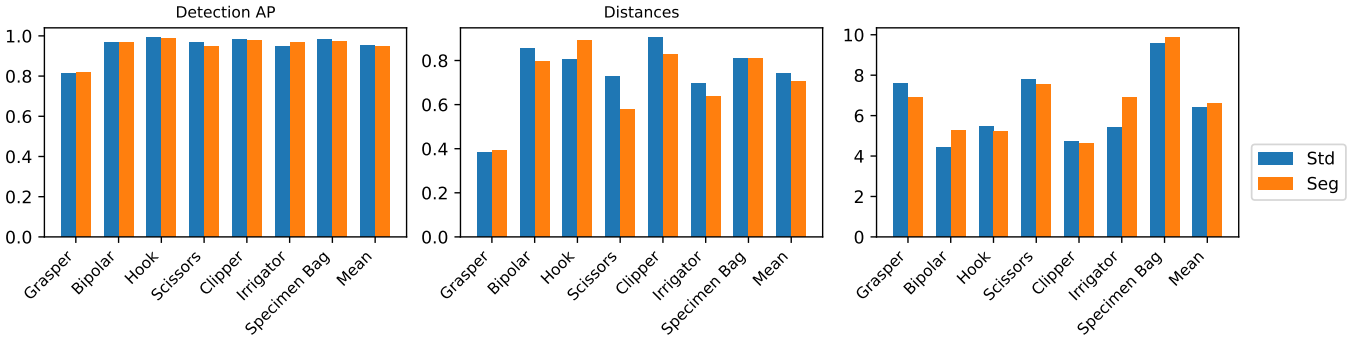


Figure 4.16: (ChannelSwitcher) left: Classwise Average Precision of detection, middle: Classwise Average Precision of localization, right: Average distance between bounding box center and predicted position in % of the image diagonal

4.2.7 Visualization

In the last sections we have seen, that we could improve the model performance significantly. To get a better understanding, how the models outputs have changed, due to the improvements, we visualized a set of images together with their corresponding heatmaps and tool predictions. We can see in 4.17 and 4.18, that our optimized model has much more distinct peaks, while the original ones are very spread, smaller and often lying on tissue that is not a tool.

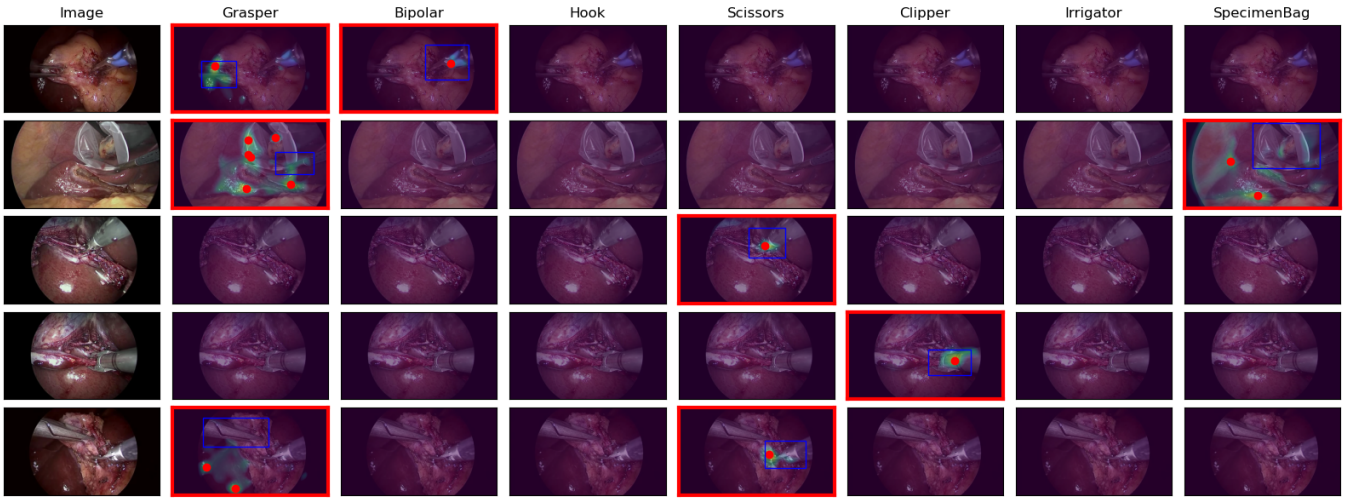


Figure 4.17: (Original model) heatmaps, predicted tools (red dots), ground truth bounding boxes (blue)

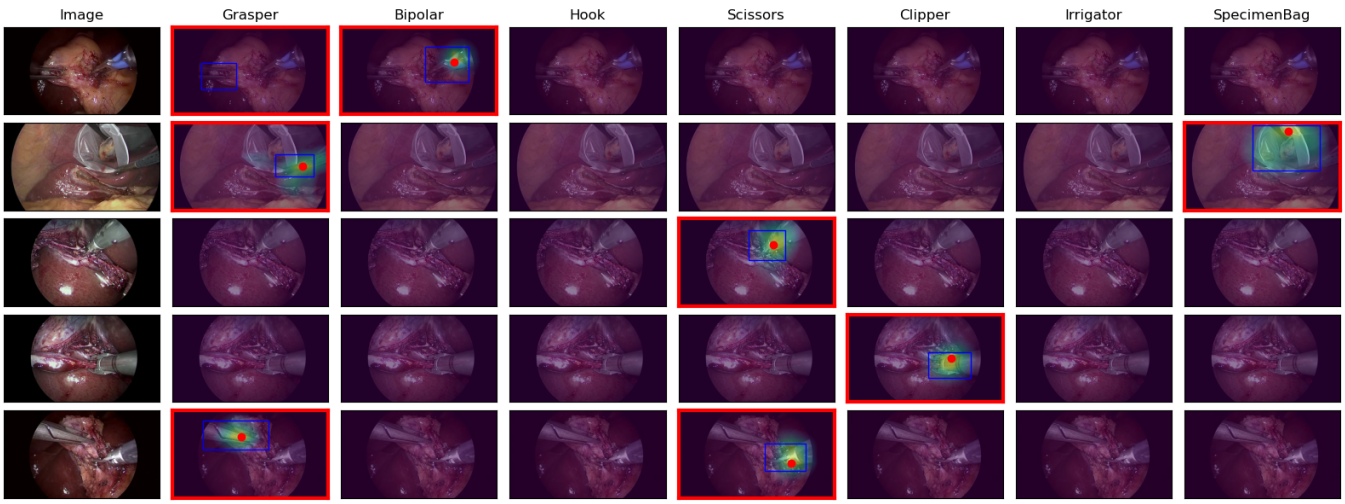


Figure 4.18: (Our model) heatmaps, predicted tools (red dots), ground truth bounding boxes (blue)

4.2.8 SurgToolLoc

As already mentioned, we sadly missed the official challenge deadline by just a few days and couldn't let our model be tested on the official validation dataset. Nonetheless, we were eager to see how well our approach performs on the *SurgToolLoc* dataset, so we created our own validation set, where we cared about that no frames from the same video occur in train and test at the same time. The results of our best model, the ResNet18 with optimized parameters and strides (2,2), can be seen in 4.19 and 4.20a.

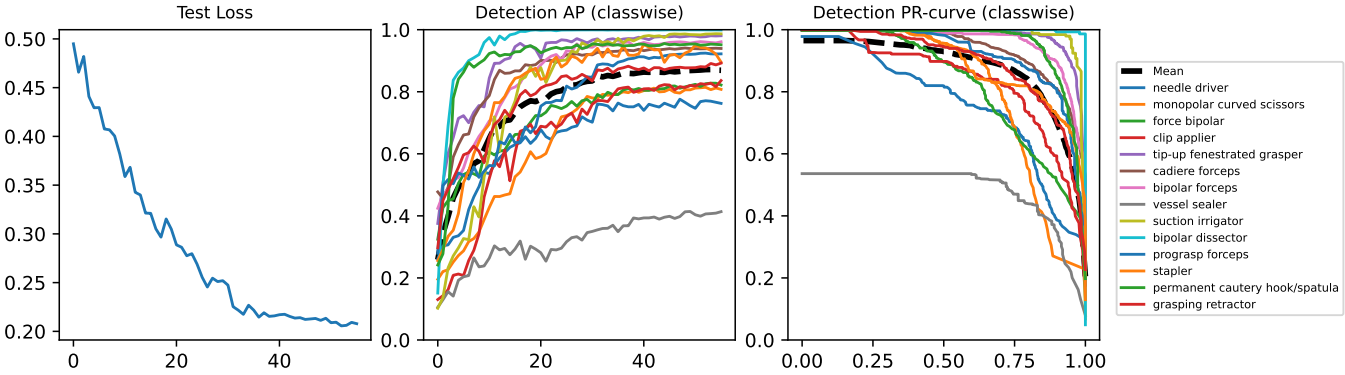
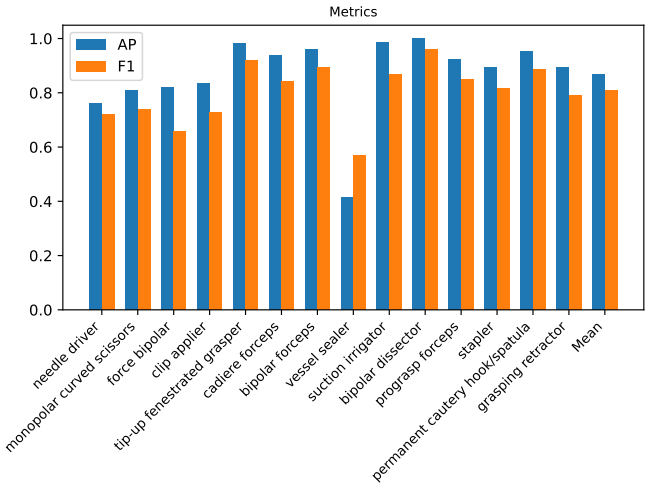


Figure 4.19: (*SurgToolLoc* Dataset) left: Progress of test loss, middle: Progress of the detection AP, right: classwise Precision-Recall Curves of the trained model



(a) Average Precision and F1-Score of every tool

Rank	Team	Average f-1 score	Submission status	Institution
1	HRI_MV	0.7485	Complete	Hikvision Research Institute
2	HKMV	0.72	Complete	South China University of China
3	NVIDIA	0.7055	Complete	NVIDIA
4	ANL-Surg	0.6691	Complete	Argonne National Lab
5	HVRL	0.6605	Complete	Keio University
6	sk	0.643	Complete	Muroran Institute of Technology, Niigata University of Health and Welfare, Konica Minolta
7	TeamZERO	0.6299	Complete	University of West of Englang Bristol, University of Glasgow, University of Bristol
8	justforfun	0.6206	Complete	Vanderbilt University
9	UKE	0.6203	Complete	University Medical Center Hamburg
10	Gatech	0.5986	Complete	Georgia Institute of Technology
11	ITeM	0.5978	Complete	Furtwangen University, University of Lipzig
12	MM	0.5637	Complete	Chinese University of Hong Kong, National University of Singapore, University College London
13	BioMedIA-2022	0.5521	Complete	Muhmammad bin Zayed University of Artificial Intelligence
14	Vision_HK	0.7278	Incomplete	Tianjin University
15	WhiteBox	0.6102	Incomplete	The University of Tokyo
16	CCM	0.5744	Incomplete	University of Strasbourg
17	Isgroup	0.5582	Incomplete	Xiamen University

(b) Final results of the *SurgToolLoc* Challenge

While most tools end up having an AP of around 0.8, some tools, like the *Bipolar Dissector* are detected with almost 100% precision. The *Vessel Sealer* is the only tool that deviates from the rest, with only an AP of around 0.4.
As the official quality metric of the Challenge was the F1-score, we chose to compute it additionally, to get a comparison to the official results (4.20b). We are aware that they can not really be compared as totally different validation datasets have been used and because

of the bias that our results have, due to being sampled from the same dataset as the training instances. Nonetheless it is interesting to see, that with an F1-score of 0.8, we could have competed with the top teams or even won.

5 Summary and Future Work

Author: Konrad Goldenbaum

Automatic localization and classification of surgical tools in endoscopic videos has many advantages, especially as an auxiliary task to provide cheaper annotated datasets for segmentation-tasks, that can ultimately improve current endeavours in medical practices and research. We managed to score a mean of over 0.95 and 0.70 average precision across all classes in detection and localization respectively on the M2CAI2016-dataset. These results are very satisfactory, although we hoped for better localization-results. We achieved them by combining a ResNet-18-backbone with stride 2×2 with a 1×1-kernel convolution-layer to generate the heatmaps for the classes and min-max-pooling for classification. We trained with stochastic gradient descent and batch-sizes ranging from 20-50 and the multi-label soft margin loss as the loss function. The hyper-parameters were successfully improved over the base-parameters by applying a Gaussian process on a small set of hyper-parameters and model-performance-indicators over 50 epochs. One possible improvement here might be to combine Bayesian optimization with Gaussian processes to further improve these parameters. We tried three pretrained networks (ResNet, AlexNet and VGGNet) as backbones, but due to extensive training we were able to show that ResNet was, for our setup, the superior choice. We were further able to decide for a 2x2-stride in the last two ResNet-convolutions as well as for min-max-pooling for classification. Taken together our architecture and training-strategy proved successful and consistent on both the M2CAI2016-dataset as well as the more complex SurgToolLoc-dataset (although we could only show that for classification in this instance).

For the future it might be up to us (or others) to try the more challenging task of phase-detection. Other possible advances might lay in segmenting the images, rather than localizing objects. Additionally, further improving localization both by using peak-responses as well as by employing other exploits is a task still ahead. Further options lay in self-supervised techniques based on generative models that might achieve considerable results without relying on annotations altogether.

Comment: We added a ReadMe to the repository to improve legibility of the projects code.

Bibliography

da Costa Rocha, C., Padoy, N., and Rosa, B. (2019). Self-supervised surgical tool segmentation using kinematic information. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8720–8726. IEEE.

Durand, T., Mordan, T., Thome, N., and Cord, M. (2017). Wildcat: Weakly supervised learning of deep convnets for image classification, pointwise localization and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5957–5966.

Elharrouss, O., Akbari, Y., Almaadeed, N., and Al-Maadeed, S. (2022). Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches.

Han, K., Wen, H., Shi, J., Lu, K.-H., Zhang, Y., and Liu, Z. (2017). Variational autoencoder: An unsupervised model for modeling and decoding fmri activity in visual cortex.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Nwoye, C. I., Mutter, D., Marescaux, J., and Padoy, N. (2019). Weakly supervised convolutional LSTM approach for tool tracking in laparoscopic videos. *International Journal of Computer Assisted Radiology and Surgery*, 14(6):1059–1067.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2015). Is object localization for free? - weakly-supervised learning with convolutional neural networks.

Ramesh, S., Srivastav, V., Alapatt, D., Yu, T., Murali, A., Sestini, L., Nwoye, C. I., Hamoud, I., Fleurentin, A., Exarchakis, G., Karargyris, A., and Padoy, N. (2022). Dissecting self-supervised learning methods for surgical computer vision.

Ross, T., Zimmerer, D., Vemuri, A., Isensee, F., Wiesenfarth, M., Bodenstedt, S., Both, F., Kessler, P., Wagner, M., Müller, B., et al. (2018). Exploiting the potential of unlabeled endoscopic video data with self-supervised learning. *International journal of computer assisted radiology and surgery*, 13(6):925–933.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

Singh, K. K. and Lee, Y. J. (2017). Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization.

Twinanda, A. P., Shehata, S., Mutter, D., Marescaux, J., de Mathelin, M., and Padoy, N. (2016). Endonet: A deep architecture for recognition tasks on laparoscopic videos.

Vardazaryan, A., Mutter, D., Marescaux, J., and Padoy, N. (2018). Weakly-supervised learning for tool localization in laparoscopic videos.

Zhou, Y., Zhu, Y., Ye, Q., Qiu, Q., and Jiao, J. (2018). Weakly supervised instance segmentation using class peak response.