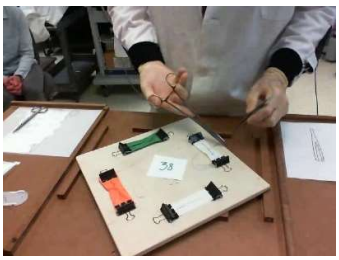
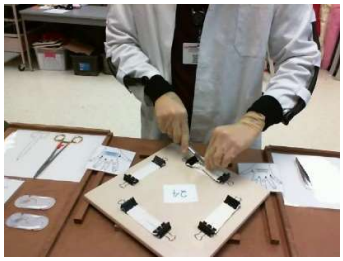
Computer Vision in the Operating Room – HW1

Niv Bardas 313151250
Luzan Faro 305283293

GitHub link - https://github.com/NivBar/CV_OR_097222_HW1.git

Exploratory Data Analysis





Before starting to work on the assignment we took time to examine the given data.
Looking at the given images and the labels led us to think of some insights:

- In the images, the synthetic human tissue comes in different colors, it may correlate the color of the training tissue to a specific tool and predict its usage by that, creating a possible bias.
- In the videos and in some of the images we saw that one hand can hide the other one, which may hide the tool usen in said hand. This can lead to confusion in the training process and harm results.
- The images and the videos are fairly noisy. The background is not neutral. For the purpose of the algorithm – training surgery students, maybe the data collectors should have aspired to gather a "cleaner" data to help the algorithm reach better results (unlike real world surgery where noise is obligatory).
- The surgical tools can be very small and indistinct, particularly they are mostly smaller than the hand holding them. This can lead to difficulties recognizing an empty hand vs hand + tool classification.
- The data is highly imbalanced, some of the labels are not even present in the training data (or at all for that matter). If not handled correctly this will lead to no predictions of these missing labels.

Experiments

To start the assignment, we were advised to choose an existing architecture to utilize in order to get good results. We examined YOLOv7 and YOLOv5 which was the one we decided to continue with because of its well written documentation and explanations.

YOLOv5 architecture description:

The YOLO network consists of three main pieces: backbone: A convolutional neural network that aggregates and forms image features at different granularities; neck: A series of layers to mix and combine image features to pass them forward to prediction; head: Consumes features from the neck and takes box and class prediction steps.

YOLOv5' backbone employs CSPNet strategy to partition the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. This strategy is also used in its neck. Other than that, YOLOv5 uses the same parts as its predecessors (v3/4). Additional information can be found in the links below.

YOLOv5 important links:

- YOLO v5 model architecture [Explained] – Depicts a high-level description of the YOLOv5 architecture, including activation functions used, loss functions and other improvements.
- YOLOv5 Documentation – Tutorials and more.
- YOLOv5 GitHub – The complete code for the model.
- YOLOv5 PyTorch Model – Pytorch module of YOLOv5 we utilized later on to create our compact submission.

Our process:

We trained ~40 different models changing hyper parameters such as learning rates {0.1 (default),0.15,0.2} and momentum {0.937 (default), 0.9, 0.95} used by the optimizers, which we examined different ones as well – namely SGD (default), Adam and AdamW which we finally saw gave the best results and was chosen.

In addition, we tried training the model using the pretrained YOLOv5 various models.
We used the small model, aka YOLOv5s, at some of the first tries. Later on, we used the more complex medium model, aka YOLOv5m, which we eventually saw yields better results.
Larger models were too large for our assigned machine to handle so reluctantly we did not get the chance to use YOLOv5l or YOLOv5x which are the most complex models of this series of models. We trained these models in different number of epochs ranging from 300 (mainly the small model which ran at a reasonable time) to 10 (mostly for testing hyper parameters. We saw that the training process, especially in the medium model, is relatively reflective of the training direction after ~10 epochs). Both models shown below were trained for 100 epochs and even could have been stopped early, reaching maximal results – we didn't do that because YOLOv5 saves both best and last weights and we didn't mind letting the training finish for the chance of slight improvement.

Moreover, we examined the flip hyperparameters (i.e., vertical flip and horizontal flip) that we discovered later on was interrupting training the model as well as we would have wanted, trying to compensate for the unbalanced dataset given. Horizontal flipping didn't add much to the results and vertical flipping harmed them actively as it labeled the flipped image the same way as the original (e.g., left needle driver stayed that way instead of changing it to right needle driver). We will elaborate on the way we handled this issue and solved the imbalanced data and flipping the images below.

Before the submission date was postponed, we had the first model we are about to show below – trained, yielding results, and tagging the test videos using the smoothing function we will discuss later in this report. Around this time, it got to our attention that the data given to us was not entirely correct like we believed we were given. Knowing this information, we decided to not only acquire the fixed data but to perfect the algorithm further, having more time until the postponed submission date. The final model we are about to show below consists of the correct data tagging, with fixed

labels (apparently the data correcting procedure dropped some of the unused classes. We actively processed the "fixed" data to be actually fixed and depicting the labeling as we intended). Moreover, we even doubled this input data by flipping it vertically and adjusting the bounding boxes and labels to fit. We managed to do it by switching the classes to show the intended class (e.g., right empty, left scissors turned into right scissors, left empty) and by adjusting only the x coordinate (by the formula fitting in case of mirroring, sources: source1, source2), as the other values y,w,h will not change in this case. By doing this we made sure that all classes are present in the train data and that made the training less obscure and imbalanced. The validation and test data were handled as such as well.
The chosen models we would like to show differ mainly by the data fed to them. These were the top two models, result-wise, we trained. Both were trained with the defaulted YOLOv5m (medium model) pretrained weights, using lr=0.1, momentum = 0.937. The first one, we will refer to it as the ***initial model*** was trained on the original data given with a 0.5 chance to be flipped vertically in every epoch (labels stay as they are, which we later on handled). The second model, we will refer to it as the ***final model***, was trained on the fixed data that was doubled by flipping the images and assigning them the expected labels.

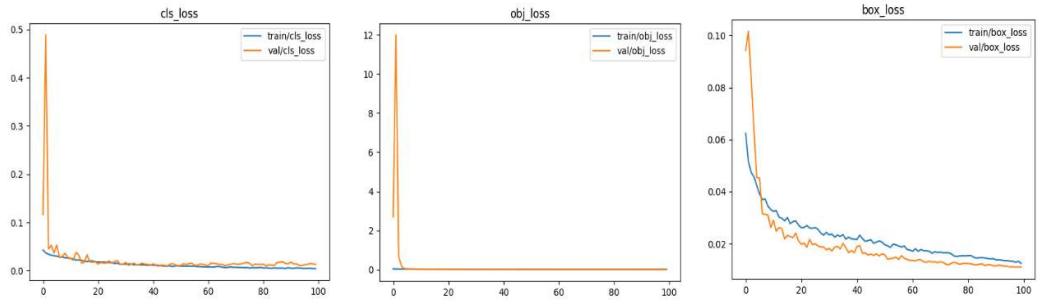
Loss Calculation

YOLOv5 calculates three distinct losses and uses them as the algorithm loss in a weighted sum manner: $Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_1 L_{box}$ (λ_i *is a scaler* $\forall i$).

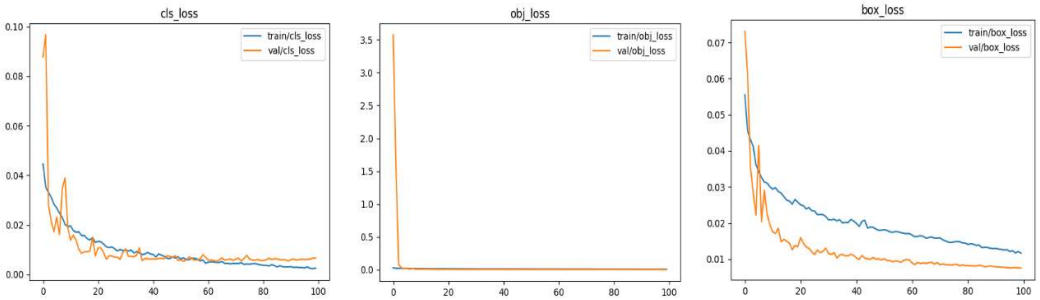
- *Classification loss* (L_{cls}) *-BCE loss.* $\lambda_{cls}\Sigma_{i=0}^{S^2}\Sigma_{j=0}^{B} I_{i,j}^{obj}\Sigma_{C\in class} p_i(c)\log(\hat{p}_l(c))$.
- *Confidence loss* (L_{obj}) *– BCE loss.* $\lambda_{noobj}\Sigma_{i=0}^{S^2}\Sigma_{j=0}^{B} I_{i,j}^{noobj}(c_i - \hat{c_l})^2 + \lambda_{obj}\Sigma_{i=0}^{S^2}\Sigma_{j=0}^{B} I_{i,j}^{obj}(c_i - \hat{c_l})^2$.
- *Bounding box regression loss* (L_{box}) *– CIoU loss.*

  $\lambda_{coord}\Sigma_{i=0}^{S^2}\Sigma_{j=0}^{B} I_{i,j}^{obj} bj(2 - w_i \cdot h_i)\left[\left(x_i - \hat{x_l}^j\right)^2 + \left(y_i - \hat{y_l}^j\right)^2 + \left(w_i - \hat{w_l}^j\right)^2 + \left(h_i - \hat{h_l}^j\right)^2\right]$

Initial Model



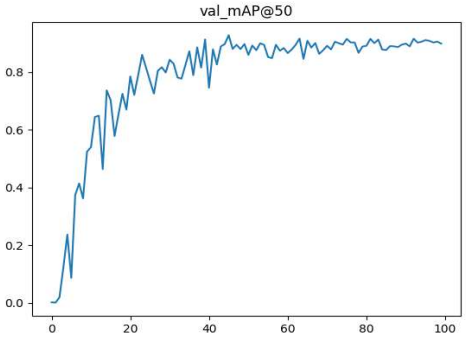
Final Model



Results ($AP@k, mAP@k$)

We will show the line graph of the validation data's $mAP@50$ and the final $AP@k$ $k \in \{25,50,75\}$ for all classes and $mAP@k$ overall calculated on train and validation data, and most importantly on the test data. *(Deviating slightly from the instructions, approved by Adam)

Initial Model

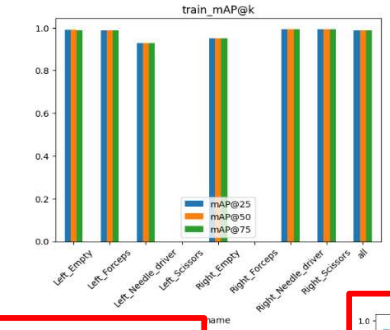


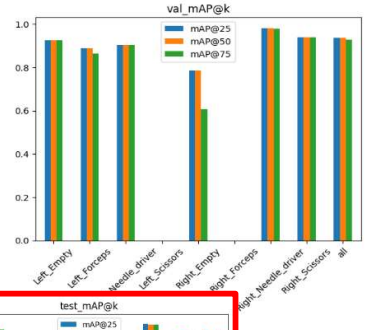
train results

instances	mAP@25	mAP@50	mAP@75	name
186	0.994785	0.994785	0.994785	Right_Scissors
2	0	0	0	Left_Scissors
532	0.994014	0.994014	0.994014	Right_Needle_driver
32	0.928258	0.928258	0.928258	Left_Needle_driver
255	0.987648	0.987648	0.987648	Left_Forceps
65	0.952152	0.952152	0.952152	Right_Empty
502	0.989297	0.989297	0.988161	Left_Empty
1574	0.987241	0.987241	0.986878	all

validation results

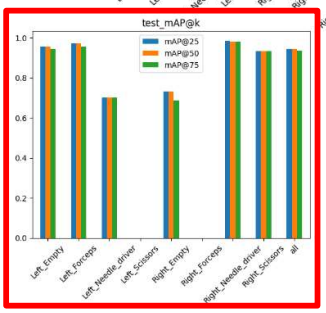
instances	mAP@25	mAP@50	mAP@75	name
29	0.940725084	0.940725084	0.940725084	Right_Scissors
95	0.982622109	0.982622109	0.977231152	Right_Needle_driver
6	0.903	0.903	0.903	Left_Needle_driver
44	0.88913254	0.88913254	0.865311667	Left_Forceps
7	0.7857	0.7857	0.60695625	Right_Empty
82	0.926402359	0.926402359	0.926402359	Left_Empty
263	0.937775107	0.937775107	0.927085123	all





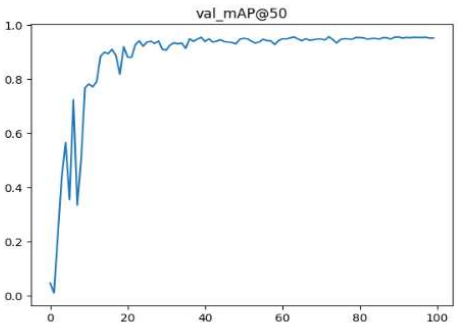
test results

instances	mAP@25	mAP@50	mAP@75	name
42	0.933951804	0.933951804	0.933951804	Right_Scissors
1	0	0	0	Left_Scissors
137	0.984643426	0.980280504	0.980280504	Right_Needle_driver
10	0.7	0.7	0.7	Left_Needle_driver
62	0.973385417	0.973385417	0.955779139	Left_Forceps
20	0.731282051	0.731282051	0.686608392	Right_Empty
126	0.954047803	0.954047803	0.945362312	Left_Empty
398	0.945496697	0.943994887	0.936257614	all



*Note that there are classes that are absent or underrepresented, this problem was solved in the next model

Final Model

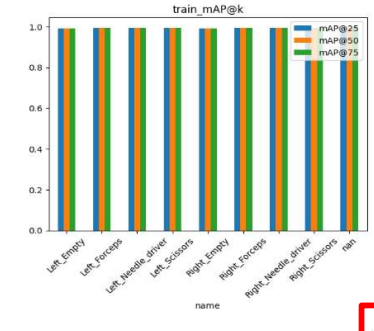


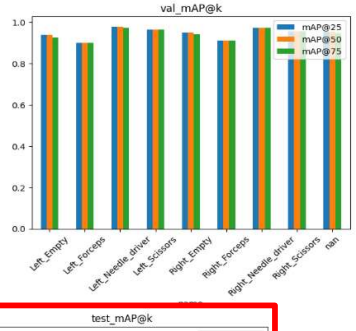
train results

instances	mAP@25	mAP@50	mAP@75	name
199	0.995	0.995	0.99419598	Right_Scissors
199	0.995	0.995	0.995	Left_Scissors
569	0.995	0.995	0.995	Right_Needle_driver
569	0.995	0.995	0.995	Left_Needle_driver
255	0.994843137	0.994843137	0.994843137	Right_Forceps
255	0.995	0.995	0.995	Left_Forceps
540	0.991203442	0.991203442	0.991203442	Right_Empty
540	0.991201474	0.991201474	0.991201474	Left_Empty
3126	0.993675193	0.993675193	0.99362401	all

validation results

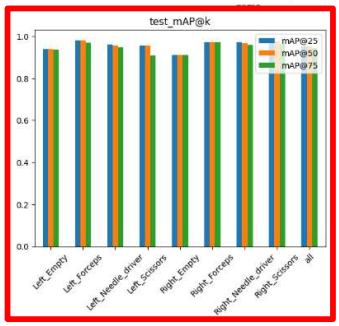
instances	mAP@25	mAP@50	mAP@75	name
34	0.95618274	0.95618274	0.95618274	Right_Scissors
34	0.964784874	0.964784874	0.964784874	Left_Scissors
101	0.972835301	0.972835301	0.972835301	Right_Needle_driver
101	0.979083255	0.979083255	0.973566989	Left_Needle_driver
44	0.910379437	0.910379437	0.910379437	Right_Forceps
44	0.89920915	0.89920915	0.89920915	Left_Forceps
84	0.948767034	0.948767034	0.942321327	Right_Empty
84	0.939859944	0.939859944	0.924466945	Left_Empty
526	0.951945318	0.951945318	0.947398561	all





test results

instances	mAP@25	mAP@50	mAP@75	name
48	0.975377101	0.975377101	0.964370435	Right_Scissors
48	0.956478664	0.956478664	0.906499563	Left_Scissors
150	0.970685465	0.965335683	0.956734165	Right_Needle_driver
150	0.961807858	0.955623232	0.947679006	Left_Needle_driver
62	0.971208784	0.971208784	0.971208784	Right_Forceps
62	0.981558372	0.981558372	0.969755803	Left_Forceps
138	0.910334887	0.910334887	0.910334887	Right_Empty
138	0.938272625	0.938272625	0.934506597	Left_Empty
796	0.953244316	0.951070747	0.942703096	all



Results – Tool Usage Evaluation

To better perform on the test data, we constructed a smoothing function to smooth out both the labeling output of the videos and their bounding boxes. First, we ran the inference session on a video frame by frame using the model we trained used by the predict function we wrote. At this point we had a bounding box, predicted label and confidence level for every frame in the video.

Next, were going over the predictions list index by index taking 2 (chosen empirically) frames from its past and two from its future. Taking only frames with confidence level of 0.75 (chosen empirically) and above into consideration and adding the current prediction (to have double impact on staying the same) we will choose the smoothed label as the majority vote and the smoothed bounding box as the weighted average of all chosen frames using the confidence levels as weights.

The result are as follows:

class	precision	recall	F1 score	accuracy	total instances
Right_Scissors	0.778710016	0.798387097	0.788425803	0.798387097	3720
Left_Scissors	0	0	0	0	0
Right_Needle_driver	0.969475183	0.984326257	0.976844278	0.984326257	39429
Left_Needle_driver	0	0	0	0	0
Right_Forceps	0	0	0	0	0
Left_Forceps	0.809184225	0.952556601	0.875036513	0.952556601	15724
Right_Empty	0.814631735	0.604770642	0.694187026	0.604770642	2725
Left_Empty	0.974243874	0.878208955	0.923737092	0.878208955	30150
macro average	**0.869249006**	**0.843649911**	**0.851646142**	**0.843649911**	
Weighted average	**0.931237469**	**0.92519728**	**0.925909492**	**0.92519728**	

Discussion and Conclusions

- Before commencing in training a new model, it is of highly importance to check the validity of the given input data. Understanding it and preprocessing it is one of the most important steps, if not the most important one.
- Enriching the input data, when done correctly, can increase the results of the model by a lot and should be always considered if possible – by flipping the data horizontally, vertically, rotating it and rescaling it. Here we benefitted from vertical flip but not so much using horizontal and even got worse results when trying it.
- There are tools that are commonly used in the right or left hands, almost exclusively even. This can lead to skewed datasets (like ours). This real-world preference should be handled to correctly classify out of the ordinary scenarios (e.g., left-handed surgeon).
- The more complex YOLOv5 model used (s vs m), the better results were yielded. This is probably because of the more intricated architecture used by the model and the better understanding the correlations provided by the train data.
- Train, validation, and test are sampled from the same distribution and the model infers similarly using each of them – during the training process and post training. This means the data is split correctly for our usage but may not resemble real-world data which differ from it. For further analysis we should consider using different data sets to use our model on to avoid overfitting on this specific dataset.