2122-M-IF3 - Computer Vision

Summer project

Authors:
Henrik Adrian Hansen
Ahmet Onur Akman
Matthias Schmitz

July, 2022

Table of Contents

List of Figures ii

List of Tables ii

1 Introduction 1

2 Approach and Used Methods 1

2.1 Hand Detection and Localization with Deep Learning 1

2.1.1 Model Selection 1

2.1.2 Technique 2

2.1.3 Data 2

2.1.4 Hardware Requirements 2

2.1.5 Fusion and Non Maximum Suppression 2

2.1.6 Output Format 3

2.2 Hand Segmentation with OpenCV and C++ 3

2.2.1 Color segmentation 3

2.2.2 floodFill 3

2.2.3 grabCut 4

2.2.4 Combined methods and main algorithm 4

2.2.5 Use of combined methods 5

3 Results 5

3.1 Detection Accuracy 5

3.2 Segmentation accuracy 9

4 Discussion 13

4.1 Detection 13

4.2 Segmentation 13

5 Conclusion 13

Bibliography 15

Appendix 16

A Workload 16

List of Figures

1 Output of detection module, generated bounding boxes for first 10 images 6
2 Output of detection module, generated bounding boxes for image 11-20 7
3 Output of detection module, generated bounding boxes for last 10 images 8
4 Output of segmentation module, first 10 images 10
5 Output of segmentation module, image 11-20 11
6 Output of segmentation module, last 10 images 12

List of Tables

1 IoU scores calculated for each sample in test set 5
2 Pixel accuracy for both detected and ground truth boxes for first 20 images 9
3 Pixel accuracy for both detected and ground truth boxes for the last 10 images . . 9

1 Introduction

In the modern technology the human-machine interaction plays a more and more important role. To let a robotic system support the human in the work it does, the interface must be adjusted correctly.
An example for an interface is the recognition of human hands. Gesture and sign recognition or hand detection can be used by robotic systems to interact in a supportive way. Additionally the detection of body parts can also be used in a informative way. For instance, localization as well as tracking can be used in movement analysis and medical rehabilitation techniques.

The aim of the project is to detect several hands of various position, form, skin color and lightning in 30 different images. The hands shall be detected by marking them with a bounding box and segmenting them from the background.

2 Approach and Used Methods

To find and segment the hands in the given 30 images a combination of Deep Learning and classic computer vision techniques is used. The project is split into two modules. The task of the first module is to detect the hands and draw a bounding box around them. Therefore a Deep Learning Network is trained by performing Transfer Learning based on a data set of more than 4700 training images and their bounding box annotations. The output of the network is one text file for each input image containing all detected hands. In the text file, each line corresponds to the coordinates, width, and height of a bounding box.

In the second module, the bounding boxes from the first module are passed to a C++ program where they are used to specify the region of interest (ROI) in which the segmentation of hand and non-hand in the images takes place. A combination of methods is used to provide a versatile technique able to segment a variety of images. The module outputs two images: The first is the original image where the segmented hands are painted in separate colors, and the second is a binary segmentation mask. In the binary mask white is hands, and black is not hands. It is compared to given ground truth for performance measurement by pixel accuracy.

2.1 Hand Detection and Localization with Deep Learning

It was decided to use neural networks to benefit from their ability to extract features so that the first phase yields better results. This part of the project was particularly important because it sets the maximum achievable accuracy for the second part and the final product. But not every type of neural network and not every architecture is suitable for such a task. Moreover, considering that we do not possess a high computational power to work with, and neither a rich data set to examine each candidate model comprehensively, we felt the necessity of benefiting from already proven successful work to perform the best in the given facilities.

2.1.1 Model Selection

There are few other models to work with for tasks like ours presented in the literature. Just a few of them namely are Fast-R-CNN, Faster-R-CNN, Mask-R-CNN, and Region-based Fully Convolutional Network. However, given its popularity, and the personal experience of one of the group's members, it was decided to work with YOLO, specifically YOLOv3.

YOLO is a convolutional model presented for object detection tasks. It is shown to be fast and efficient, also suitable for different tasks, with the help of transfer learning techniques.[6] The v3 adds to the previous version a higher accuracy and better time efficiency.[5]

2.1.2 Technique

We have used the weights obtained by prior training. The link to this weights file can be found **here**. We initialize our training with the weights obtained by training the Darknet with the COCO data set [1], which contains a wide range of objects. This seems to work well for custom object detection tasks, perhaps because a successful model to detect and classify a wide range of objects is ultimately very well trimmed for feature extraction and object classifying.

We have not done any updates on the architecture of the network. However, we have tuned these initial weights by training the model with several subsets of our data, for more than 2000 epochs.

2.1.3 Data

Since we were not given any training set, we were required to either find one or construct our own. We have done both. We have created a data set with our own images, some unlicensed images from Google Images, a few from EGO Hands, and a few from HOF Dataset.

For this part of the data, we have made sure that the training data does not contain the same or unfairly similar samples to the testing data. As the annotations were not given in our desired format (or not given at all) we were required to make them manually. Therefore we have done the bounding box marking tasks manually with related software. After that, we tried to make our data even more expressive by applying slight augmentations in some subset of the data, turning it into grayscale or sepia, or flipping the colors. This operation was limited, due to we were required to keep the validity of the bounding boxes.

Secondly, we have benefited from the Hands Dataset created by the Visual Geometry Group at the University of Oxford. This set contains a large number of images and already prepared annotations. We have converted these annotations into our desired "YOLO" format and fed the samples into our model.

All versions of the data sets we have used in this project are shared in the following Google Drive link.

https://drive.google.com/file/d/1beJPbXaB2e5CJKlhE0d5jldA43zleDxz/view?usp=sharing

2.1.4 Hardware Requirements

As much as it is a time-efficient model, training of YOLO requires good GPU usage and time. As we did not possess the required hardware for fast and efficient training, we have benefited from Google Colab's free plan. As it was very limiting about the GPU usage, we did not have the freedom of keeping our training going as long as we please, but a few times we were let to go reasonably far. However, as even the training length we were allowed to achieve with the free plan was not enough to achieve good accuracy, we have found a new workaround, with the help of a different approach.

2.1.5 Fusion and Non Maximum Suppression

As we proceed in the training, we kept building and modifying our data set and ended up with different training lengths on Colab. As a result, we have ended up with different models which are good at some parts of the testing data, and bad at others. Usually, the false positives were not an issue except for a few examples, but the false negatives were the problem to deal with. We have solved this issue by "fusing" our models and having a model with a minimum number of false negatives. In other words, we have detected a box if it is detected by any of the models. It is hard to call this method "bagging", as the training sets are different, network architectures are the same and the voting system is different than how it is usually.

Even one weight file is computationally demanding due to the size of the YOLOv3 network. Therefore it is not so time efficient to load and use every single weight file we have obtained. Therefore, we have chosen the 6 most accurate models out of 13 to work with.

One can imagine that this approach will yield a lot of bounding boxes in a single image. For this, we have used OpenCV's non-maximum suppression method for filtering out the "weaker" boxes in a given area, according to the "confidence" it represents.

2.1.6 Output Format

The final product of the first phase of this project outputs one image and .txt file per each given input image. Image output is essentially the same image but with the bounding boxes. The .txt file contains the coordinates of the top left corner of the bounding box and the height and width of the bounding box. These .txt files are fed to the second phase product as input.

2.2 Hand Segmentation with OpenCV and C++

The hand segmentation in C++ is based on different computer vision techniques, as each method on their own is insufficient for segmenting the whole data set. A combination of various techniques leads to a more versatile algorithm and improved results. The final segmentation method relies on first and second derivative filtering, thresholds, contour finding and filling, segmentation by color (RGB) and the GrabCut algorithm. A more detailed description of the used methods and their effect on segmentation can be found in the following subsections.

2.2.1 Color segmentation

Color segmentation was attempted based on detecting human skin with YCbCr and RGB color models based on a research paper. [7]. The YCbCr color model required first changing the color space and then applying thresholds on the Cb and Cr values, while keeping the whole lightning range Y. The RGB color model just required applying thresholds on the R, G and B values.

In the final algorithm a color segmentation in RGB color space is applied. The function *segmentBySkinColour* uses three conditions to distinguish between hand or non-hand. Following the naming **red, green** and **blue** for color values of channel two, one and zero, **min** and **max** for minimum and maximum value in a data set and **abs** for the absolute of a digit the three conditions imply

Cond. 1: (red > 95) && (green > 40) && (blue > 20)

Cond. 2: max(red, blue, green) - min(red, green, blue) > 15;

Cond. 3: (abs(red - green) > 15) && (red > green) && (red > blue);

Just for fulfilling Cond. 1, 2 and 3 the pixel is identified as part of a hand.

2.2.2 floodFill

Region growing was attempted by the OpenCV *floodFill*-algorithm, which attempts to fill a connected component with a given color [4]. It starts from a seed point and grows based on similarity to neighboring pixels based on brightness/color decided by upper and lower thresholds and stops based on a edge mask that marks the contours of the image. The edge mask was found with the Canny edge detector algorithm [2].

2.2.3 grabCut

The *grabCut*-algorithm aims to split foreground and background of a picture apart, requiring minimal user interface. Merely the ROI must be passed. It was designed by three researchers from Microsoft Research Cambridge, UK in 2004.[3] It is based on iterated graph cuts.

2.2.4 Combined methods and main algorithm

For improved results, not depending on skin color, lightning or positioning, various techniques are combined to one segmentation algorithm (*folldFillSegmentation1*) which is applied to all images. For further improvement two other segmentation algorithms (*folldFillSegmentation2*) & (*GrabCut-Segmentation*) are build by a combination of different techniques and changed parameters. Based on metrics they are applied on a set of images which did not lead to a sufficient segmentation after applying the first algorithm.

- **floodFillSegmentation1:** The first step of the *floodFillSegmentation1* algorithm is to create a snippet of the region of interest (ROI) – given by the coordinates of the boxes of the previous hand detection. In a second step the snippet is smoothed with two different bilateral filters. Subsequently both smoothed images are filtered with a second derivative Laplacian filter of the form

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

  to obtain edges. The weaker smoothed image keeps more edges, considering many false positive regarding the outer contour of the hand. The stronger smoothed image only keeps strong outer edges, but also misses weak outer edges.

  Applying a threshold on both Laplacian-filtered snippets removes small edges which are not relevant for finding the outer contour of the hands. Setting the threshold too low and keeping too many smaller edges leads to worse results in the following *findContours*-algorithm, setting the threshold too high deletes too many relevant edges which makes the contour finding impossible.

  Next to the second derivative filtering also first derivative filtering is used. The Canny-Edge-Detector is applied to the less smoothed snippet. Due to that, both thresholds for cohesive edges are set to high values. Low values led to more noisy results. Many contours were detected which did not belong to the required outer contour.

  findContours is then applied to the sum of all previous edge detection outcomes. Edges which were detected with all three derivative filters are stronger and contribute more, small edges which were just contained by one filter may help to close gaps and reduce leaking. Setting the flag "*RETR_EXTERNAL*" in *findContours* improves keeping outer edges while neglecting inner ones.

  Afterwards the *floodFill*-algorithm is used, coloring the inside of the detected contours, see Section 2.2.2. In the last step the *segmentBySkinColour*, see Section 2.2.1, is applied on the snippet. A bit-wise comparison with the outcome of *floodFill* as well as dilating with a 2x2 kernel provides the final segmented ROI. A black mask containing white pixels everywhere, where hands are detected, is computed.

- **floodFillSegmentation2:** The *floodFillSegmentation2*-algorithm uses the same technique as the the *floodFillSegmentation1*-algorithm. It differentiates in the preprocessing. Second derivative filtering is not used. The Canny-Edge-Detector is not anymore applied on a smoothed snippet, but on the original with different parameters. It provides the basis for the subsequent contour finding and filling. *segmentBySkinColour* as well as a bit-wise comparison follows.

- **GrabCutSegmentation:** The *grabCut*-algorithm, compare chap. 2.2.3, is applied on the snippet of the ROI. A black mask containing white pixels everywhere, where hands are detected, is computed and bit-wise compared with the previous *floodFillSegmentation1*-mask. Subsequently the ratio between white pixels of the comparison and all the pixels of the source image is computed. If the ratio is bigger than a threshold *GrabCutSegmentation* returns the *floodFillSegmentation1*-mask, if it is smaller than the mask computed with the *grabCut*-algorithm is returned.

2.2.5 Use of combined methods

floodFillSegmentation1 is initially applied to all images. Subsequently the ratio between white pixels of the *floodFillSegmentation1*-mask and all the pixels of the source image is computed. If the ratio is smaller than a threshold *GrabCutSegmentation* is applied. For the return of *GrabCutSegmentation* the ratio is calculated again. If it is smaller than a threshold *floodFillSegmentation2* is used.

3 Results

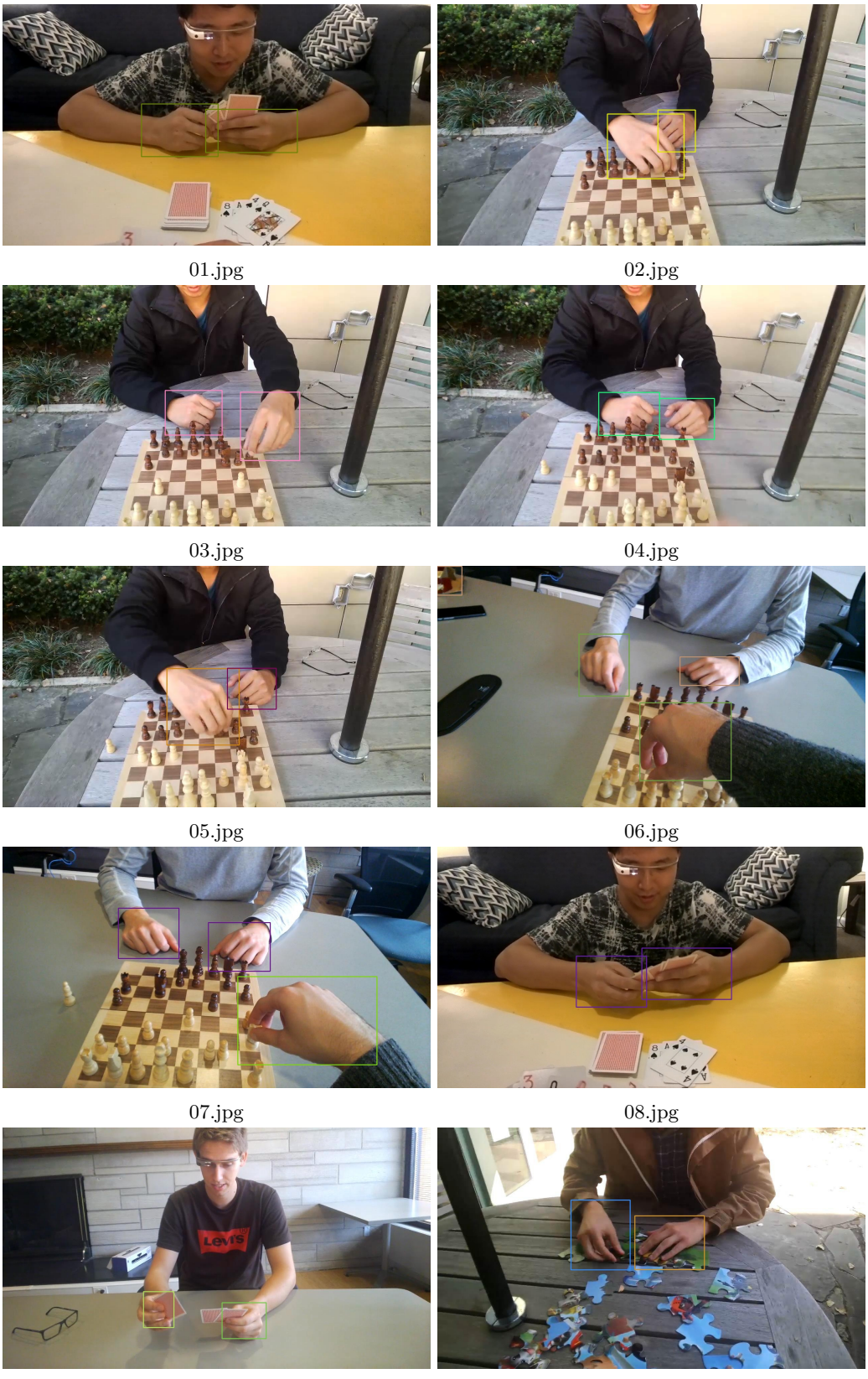
3.1 Detection Accuracy

Performance of the first part of the project, detection, and localization via deep learning, was measured by the IoU score of the final model compared to the ground truth. This metric was calculated by comparison of two masks, one contains pixels detected by our model, and the other contains the pixels that were supposed to be detected according to the ground truth. The number of the pixels in the intersection of these two masks divided by the union of them yields the following scores for each one of the test samples. Formally, the calculation formulated at Equation-(1) for the final model yields scores given in Table-1. The average IoU score of the model for every testing sample is 0.714.

$$IoU = \frac{\#(DET \cap GT)}{\#(DET \cup GT)} \tag{1}$$

1	0.750	**11**	0.717	**21**	0.763
2	0.810	**12**	0.757	**22**	0.437
3	0.760	**13**	0.750	**23**	0.636
4	0.758	**14**	0.780	**24**	0.672
5	0.800	**15**	0.700	**25**	0.660
6	0.741	**16**	0.765	**26**	0.653
7	0.749	**17**	0.790	**27**	0.597
8	0.830	**18**	0.744	**28**	0.732
9	0.716	**19**	0.711	**29**	0.516
10	0.649	**20**	0.776	**30**	0.766

Table 1: IoU scores calculated for each sample in test set



01.jpg 02.jpg

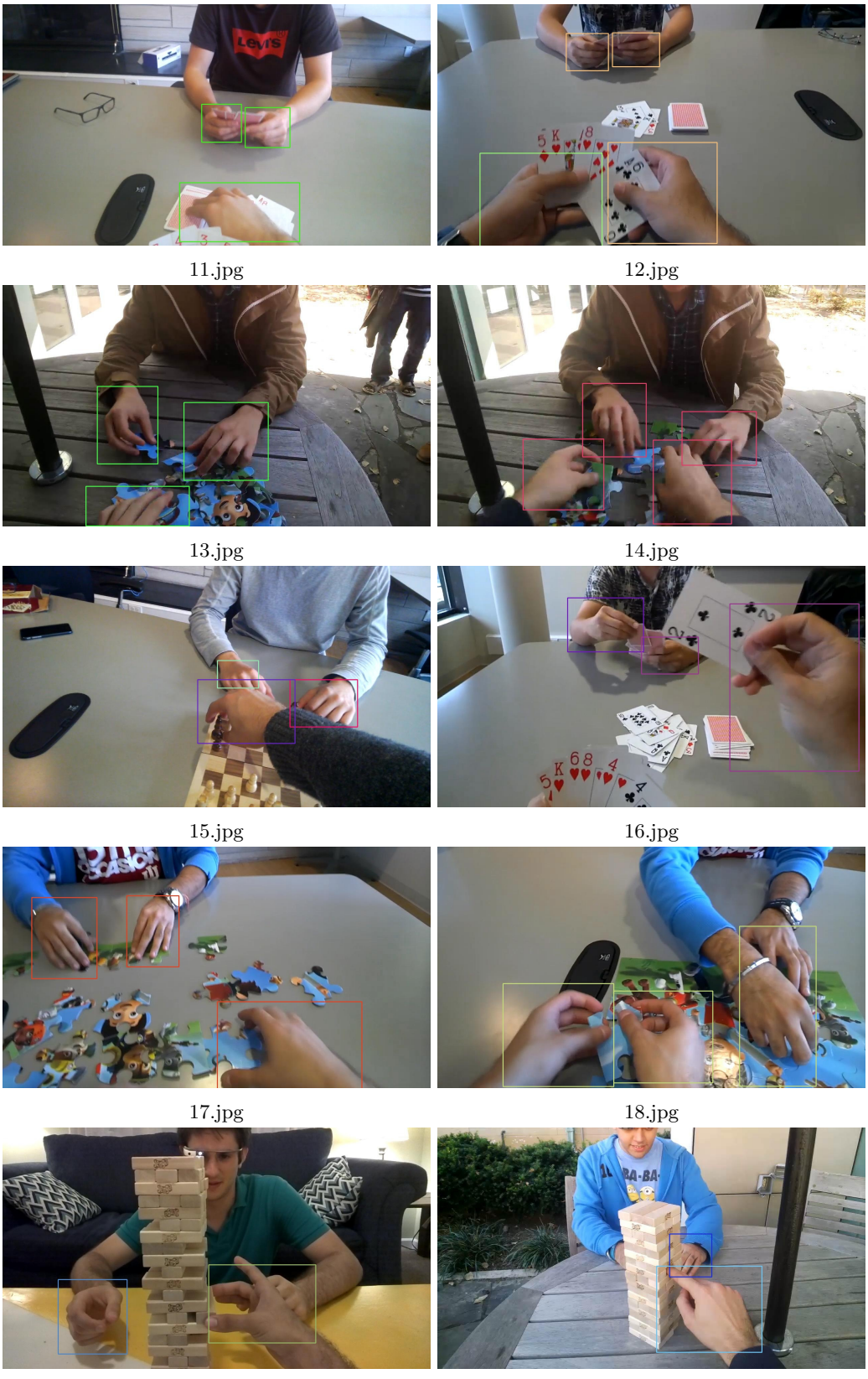
03.jpg 04.jpg

05.jpg 06.jpg

07.jpg 08.jpg

09.jpg 10.jpg

Figure 1: Output of detection module, generated bounding boxes for first 10 images



11.jpg 12.jpg

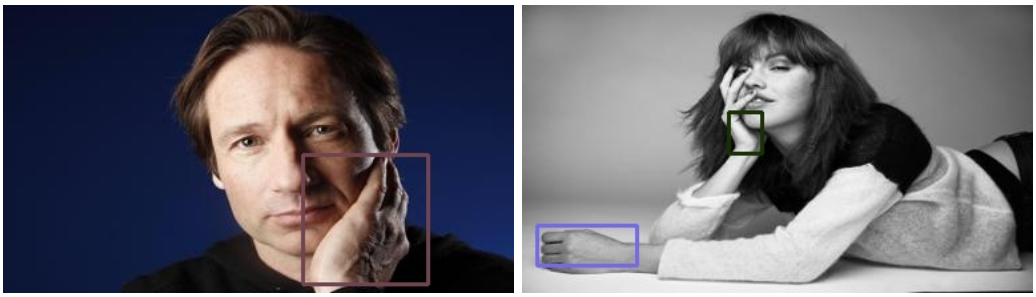
13.jpg 14.jpg

15.jpg 16.jpg

17.jpg 18.jpg

19.jpg 20.jpg

Figure 2: Output of detection module, generated bounding boxes for image 11-20



21.jpg 22.jpg

23.jpg 24.jpg

25.jpg 26.jpg

27.jpg 28.jpg

29.jpg 30.jpg

Figure 3: Output of detection module, generated bounding boxes for last 10 images

The bounding boxes that are generated by our model in the first module are shown in Figure-1,2,3.

Image	Ground truth	Detected	Image	Ground truth [%]	Detected [%]
1	83.0763	76.0445	11	87.7256	86.1715
2	87.4338	86.9272	12	69.8859	72.8419
3	89.5404	90.4322	13	77.1273	82.5066
4	85.3865	88.1882	14	79.0833	82.2733
5	84.5143	85.7441	15	85.0345	85.7721
6	86.8651	86.098	16	69.0977	71.4703
7	87.8114	86.7018	17	89.6253	90.979
8	79.5342	78.3213	18	75.2157	75.4019
9	70.7277	69.8085	19	68.3168	63.0391
10	83.8387	87.7882	20	86.7193	84.983

Table 2: Pixel accuracy for both detected and ground truth boxes for first 20 images

Image	Ground truth [%]	Detected [%]
21	96.3081	87.3327
22	68.5857	80.1653
23	63.387	68.3094
24	84.901	78.6199
25	43.64	40.9119
26	77.25	66.6576
27	77.6642	77.3179
28	82.1106	80.6298
29	71.6644	67.9378
30	85.081	80.5583

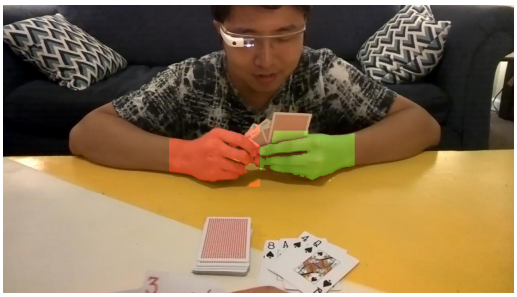
Table 3: Pixel accuracy for both detected and ground truth boxes for the last 10 images

3.2 Segmentation accuracy

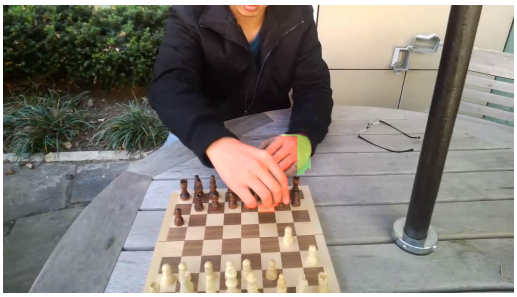
The performance is measured by pixel accuracy, the percentage of pixels correctly classified in the image. It was decided to only consider the pixels inside the bounding boxes to measure more correctly by avoiding the "inflation" from all the pixels in the image not considered for segmentation. Pixel accuracy was calculated by dividing the amount of correctly labeled pixels, the sum of true positives and negatives, by the total amount of pixels inside the segmented bounding boxes. The mean pixel accuracy with the ground truth bounding boxes was 79.2384%, while the mean pixel accuracy with the detected bounding boxes was 78.6644%. The pixel accuracy for each individual image is shown in Table 2 and Table 3

$$P_A = \frac{\#T_P + \#T_N}{\#P} 100\%$$

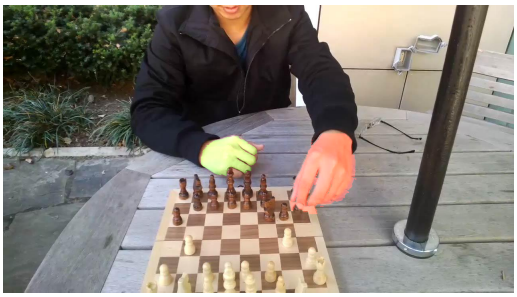
The segmented images using the detected boxes is found in Figure 4, Figure 5 and Figure 6.



Segmented image 1

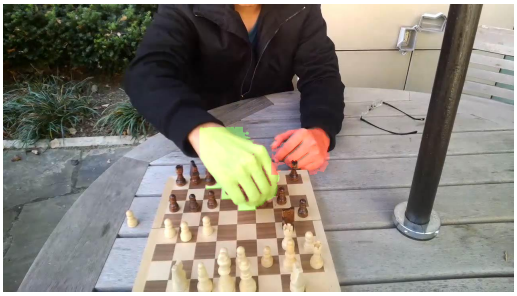


Segmented image 2

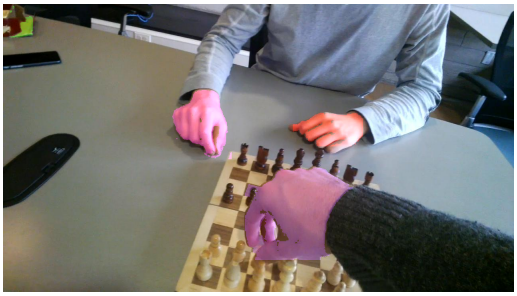


Segmented image 3

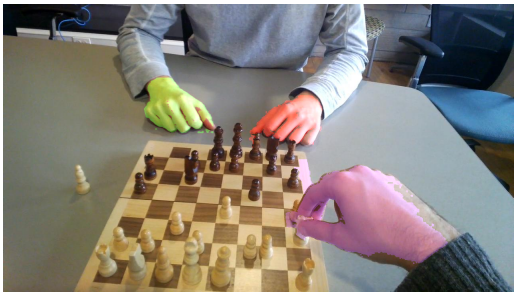
Segmented image 4



Segmented image 5

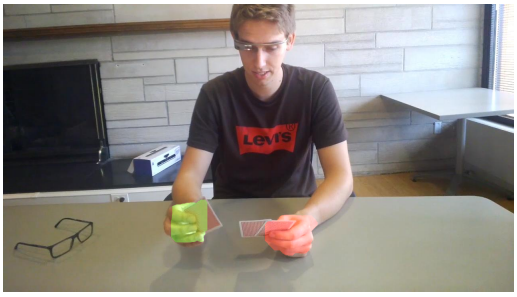


Segmented image 6



Segmented image 7

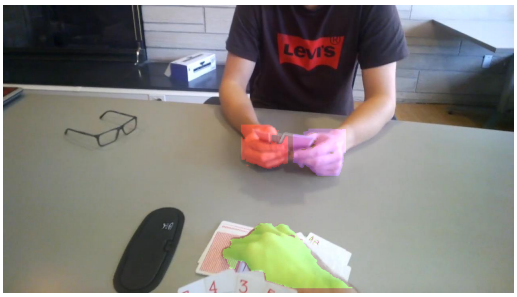
Segmented image 8



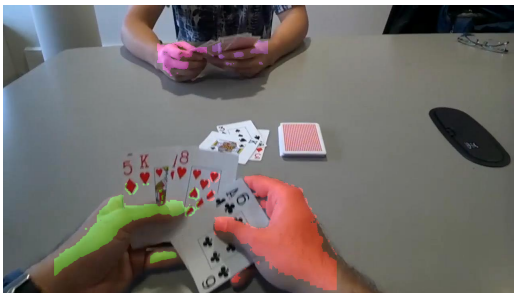
Segmented image 9

Segmented image 10

Figure 4: Output of segmentation module, first 10 images

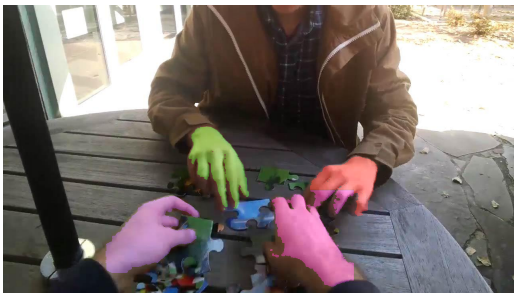


Segmented image 11



Segmented image 12

Segmented image 13



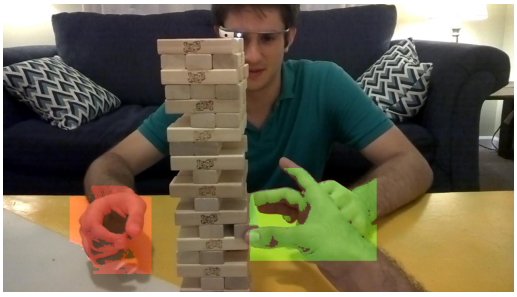
Segmented image 14

Segmented image 15

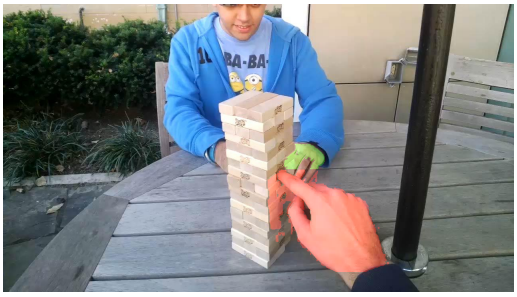
Segmented image 16

Segmented image 17

Segmented image 18



Segmented image 19



Segmented image 20

Figure 5: Output of segmentation module, image 11-20

Segmented image 21

Segmented image 22

Segmented image 23

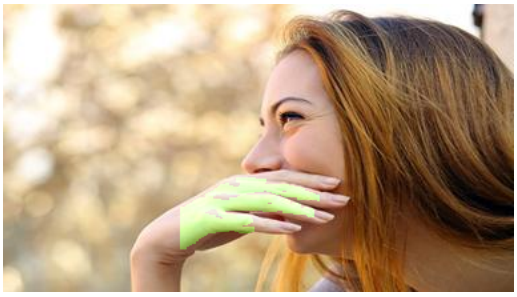
Segmented image 24

Segmented image 25

Segmented image 26

Segmented image 27

Segmented image 28



Segmented image 29

Segmented image 30

Figure 6: Output of segmentation module, last 10 images

4 Discussion

4.1 Detection

The performance metric swings between 0.437% and 0.830% for each test sample. It is a popular opinion that achieving a high accuracy is very challenging for the detection tasks using bounding boxes. This is because the borders of the bounding boxes can be changed to make the box bigger and smaller, and yet still the detection might preserve its validity. This effect is also seen in our detections. For each image, the detection hands seem to be pretty impressive, there are only 2 false positives in all test samples and overall there is no hand in any image that is completely missed by the boxes. Yet, the highest IoU score could not make it above 85%. We also notice that in some images, our model and the ground truth sometimes disagree that from which point a human hand starts, and this causes a lower IoU score, even though the entire hand is captured perfectly. Issues like this lower the IoU score but the detection has still succeeded in finding the hand in the image.

When we look closer at the detections, one can say that the detection accuracy in two different samples, say sample #07 and sample #08, it is very tricky to tell the difference. Yet, these two images had been processed with a difference of almost 10% accuracy. However, for some other cases, say sample #22 or #26, the low IoU score is spot on and the mistakes are easily recognizable. However, this kind of cases seem to be just a few rare occurrences. Therefore, these detections on the test samples, in our opinion, can be considered sufficient and pleasing.

4.2 Segmentation

The pixel accuracy varies between 43% and 96%, indicating that the method's precision differs from image to image. In some cases, like the one with the lowest accuracy, none of the methods give a good result. Furthermore, choosing the right subroutine, or combination of subroutines, for the right image it is not an easy task. The mean pixel accuracies are at a solid level, demonstrating that most of the pixels are correctly classified. It is also clear from Table 3 that the 43% accuracy picture is an outlier.

Even though the percentage of correctly classified pixels alternate depending on if ground truth boxes or detected boxes are used, the mean pixel accuracy is slightly higher for ground truth boxes. This is expected as the methods were developed and tuned according to the performance demonstrated by using these boxes. Moreover, the detected boxes are not as precise.

In Figure 1 it can be seen that the segmentations are good. This is backed by the first half of Table 2 where the first 10 pictures have high percentages. This is to be expected as these images are considered easier to segment, they have similar backgrounds. In Figure 5 and Figure 6 the segmentation is not as consistent, but still mostly satisfactory. This is also expected as these images are harder to segment: the backgrounds have more variations and along with differing skin tone and gender. It can be seen that segmentation will low pixel accuracy also has visibly bad segmentation, like image 19, image 23 and image 25.

5 Conclusion

For the detection module, we believe that the current model is quite satisfactory, we also claim that the current performance could be pushed further with the help of a larger and more expressive data set, combined with a higher computational capacity available entirely. Besides, other architectures could be examined under these facilities too, and perhaps the fusion could be performed among models with different architectures. As this new structure, containing different models learned from more challenging examples, might have a better and more generalized understanding of the shape of a hand, perhaps could be on the level of a day-to-day usable product.

The segmentation module is also quite satisfactory on average, but not on every image. A more consistent segmentation could be achieved by including more models, or choosing a more high-level approach like deep learning. That would require a large dataset with ground truths. Still we are satisfied by the results achieved by a combination of low- and mid-level techniques used.

Bibliography

[1] Tsung-Yi Lin et al. 'Microsoft coco: Common objects in context'. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[2] OpenCv. *Canny Edge Detector*. URL: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html (visited on 24th July 2022).

[3] OpenCv. *Interactive Foreground Extraction using GrabCut Algorithm*. URL: https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html (visited on 24th July 2022).

[4] OpenCv. *Miscellaneous Image Transformations*. URL: https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#gaf1f55a048f8a45bc3383586e80b1f0d0 (visited on 24th July 2022).

[5] Joseph Redmon and Ali Farhadi. 'Yolov3: An incremental improvement'. In: *arXiv preprint arXiv:1804.02767* (2018).

[6] Joseph Redmon et al. 'You only look once: Unified, real-time object detection'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[7] J.M Wayakule S. D. Patravali and Apurva D Katre. 'Skin Segmentation Using YCBCR and RGB Color Models'. In: *International Journal of Advanced Research in Computer Science and Software Engineering* 4 (2014). URL: https://www.researchgate.net/publication/290440563_Skin_Segmentation_Using_YCBCR_and_RGB_Color_Models (visited on 23rd July 2022).

Appendix

A Workload

The workload was divded as follows: Onur did the detection module as he had prior experience with deep learning. The segmentation module was designed and implemented by Henrik and Matthes. Testing, performance measurement, and report writing was done evenly by the group members. The total amount of work hours per member was about 100 hours.