Hand detection and segmentation for egocentric images

Computer Vision project

Boldrin Cristian[a], Bellan Riccardo[a]

[a] *DEI, University of Padua, Italy*

July, 2022

Abstract

A large number of works in egocentric vision have concentrated on action and object recognition. Human hands play a very important role when people interact with each other or with objects and tools. Therefore, a reliable detection of human hands from images is a very attractive result for applications of human-robot interaction, gesture recognition or human activity analysis. Here, we take an in-depth look at the hand detection problem, utilizing the state of the art of deep learning and training existing datasets on standard object detection models. After that we illustrate and discuss the different approaches that can be exploited to perform pixel-wise hand segmentation, starting from the data provided by the detection module.

1. Introduction

The growing usage of wearable devices such as - Go Pro and Google Glasses has made egocentric research in computer vision a rapidly growing area. We need computer vision approaches to handle the challenges of egocentric, i.e. in first-person, videos, including camera motion and poor imaging conditions.
The goal of this work is to develop a system capable of:

- detecting human hands in an input image
- segmenting human hands in the image from the background (i.e., binary segmentation)

In this project we focus on a very fundamental entity in egocentric situations: *hands*.
Hands are ubiquitous in our daily routine: their position, orientation and configuration are essential information telling about what the person is planning to do or what is currently doing, and what is paying attention to.
That said, hand detection and hand segmentation are crucial problems in egocentric computer vision in order to extract the hand region in the frame.

A large number of works have addressed this problem, also in third-person or surveillance scenes; however, we focused our task in first-person videos. We base our analysis on Bambach et al. [1], in which a new hand dataset is introduced alongside with deep learning strategies for hand detection and semantic segmentation.

We recurred to deep learning techniques, the state of the art for detection, performing the training of already existing and labelled dataset with powerful object detection model, making some preprocessing and post-processing of the frame in order to guarantee a better precision on bounding boxes. We also trained the model for hand classification task, making a strong assumption and constraint that the framed scene is part of an egocentric video, so we are able to classify only the main and the interacting person's left and right hand. In our work the detection and classification tasks also play the role of giving a prior to the segmentation module, in order to improve its precision and robustness.

The hand segmentation process is mainly based on the application of the GrabCut algorithm on the portions of the source image contained inside the bounding boxes individualized by the detection module. GrabCut is a an energy-minimization algorithm that allows to obtain foreground extraction, possibly from an approximate initial mask.

In this way, we were able to develop a strong and robust model for hand detection and segmentation in egocentric videos, that does not make assumption on skin color, viewpoint, scale, presence of clothes and objects (e.g., shirts, rings), brightness and contrast, and it is independent on the presence or not of the main and interacting person, so it can detect and classify a minimum of 0 and a maximum of 4 hands in a frame.
We provide a standard C++ executable that makes detection and segmentation given an input image. Such program has been developed exploiting the methods and tools provided by the OpenCV C++ library.

2. EgoHands dataset

Since our project is only focused on egocentric situations and relies on the classification of the type of hands in the image in order to segment with a different color a different hand, we decided to use only the EgoHands dataset since we were able to obtain better performances on the test mini-batch of the first 20 images. However, with this type of approach, we lose a bit of generality when the hands are not positioned and oriented in egocentric ways. EgoHands [1] includes 48 videos recorded with a google glass; each video has two actors doing one of the 4 activities: *playing puzzle, cards, jenga or chess*. Each video has been quantized into 100 manually annotated frames, resulting into 4800 frames in total, including 15,053 ground-truth labeled hands.

Related works [1], [2] use a random split into training, validation and test sets with the ratio of 75 %, 8 % and 17 % respectively.
In our work, we decided to use the respective ratio of 80 %, 10 % and 10 %. Before proceeding with the training part, we first of all carefully removed the 20 evaluation's test images from the training and validation sets.

Then, we decided to perform some data augmentation of the original dataset; with the help of Roboflow, we were able to obtained a x3 augmented dataset of 12.408 images in total, using augmentation techniques including the random enhance and lower of brightness, exposure, scale and horizontal flipping. In Fig. 1 we can see the heat map and class distribution of the above cited augmented dataset (plots from our training).

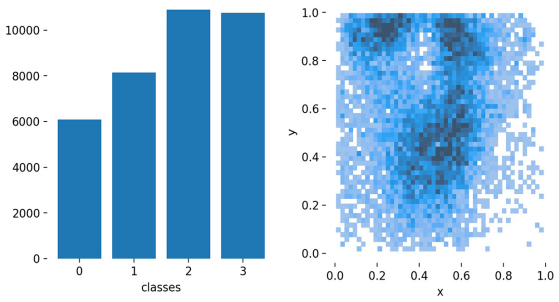


Figure 1: Heat map of labels of EgoHands dataset

The class label - id mapping that we used is the following: *0 = myleft, 1 = myright, 2 = yourleft, 3 = youright*. We can note that the interacting person's hands are the most present. On top of that, we can see that the heat map concentrates the hand distribution towards the top of the image, where the interacting person is placed, and the bottom/center of the image, where the filming person is situated.

3. Hand Detection

3.1. Training on YOLOv5

Once we have precisely splitted and removed the benchmark from the starting EgoHands dataset and created the augmented model, we started the training phase for object detection.

In principle, finding hands in first-person video frames is simply an instantiation of one particular object detection task. To this end, we chose YOLOv5 as one of the fastest and best object detector and classifier available at the moment, other than supporting different exporting formats in order to guarantee interoperability between various platforms and system, including our project developed in C++. It is based on the PyTorch framework, which has a larger community than other version, e.g., Yolo v4 Darknet, such as mosaic data augmentation and auto learning bounding box anchors. YOLOv5 is basically a Fully Connected Neural Network, comprising of:

- Backbone, extracting the essential features of the image (CSPDarknet53 focus structure)
- Head, containing the output layers with final detections. (Number of bounding boxes x (5 + class score))
- Neck, connecting backbone and head, creating feature pyramids and collecting features map of different stages. (PANet)

As loss function, YOLOv5 uses the binary cross entropy and logit loss.

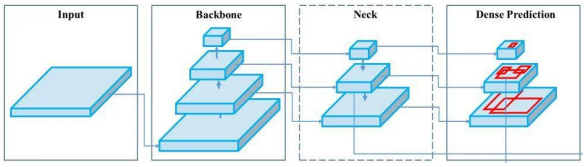


Figure 2: YOLO architecture overview

YOLOv5 was released with four models at first: Small, Medium, Large and Extra large. Recently, the Nano model and the support for OpenCV DNN were introduced. Currently, each model has two versions:

- P5: three output layers, trained on 640x640 images.
- P6: four output layers, trained on 1280x1280 images.

For the training of our custom model, we used the YOLOv5 small P5 version: we performed the so-called *transfer learning* task using Google Colab's hosted environment, including processors and memory, since we do not own CUDA GPU. Once we have specified our custom model configuration (with the 4 classes) through the .yaml YOLOv5 files, we trained the model starting from randomly initialized weights for 300 epochs, with a batch size of 64 and using the image caching for faster training.

The metrics used for the training are the standard in nowadays object detection tasks, that are the COCO metrics: AP and AR are respectively the average precision and recall, evaluated at different threshold of IoU (specifically, 10 thresholds of .50:.05:.95), where the precision is defined as the number of true positives over true positives + false positives and the recall is the number of true positives over the number of true positives + false negatives.

In Fig. 3 we report some train images batches and in Fig. 4 some test images batches.

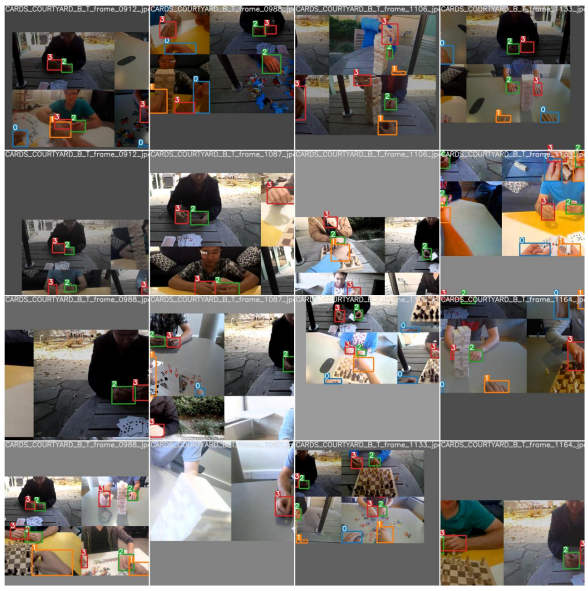


Figure 3: Train batch images

For more precise and complete plots, we refer to Fig. 5, that displays the confusion matrix and the precision and recall curves over the confidence with which the bounding boxes are detected and classified.

Finally, we exported the trained model into the ONNX (Open Neural Network Exchange) format, that is the open standard for machine learning interoperability, in order to make inference with OpenCV in our C++ program. Before doing that, we carefully inspected the network, we selected the correct opset version and we manually removed

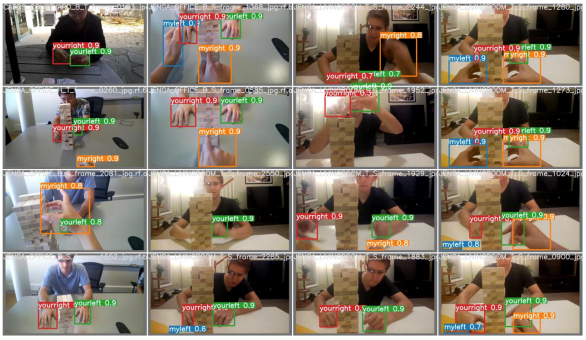


Figure 4: Test batch images

the slicing layers that cause troubles with OpenCV DNN libraries' version of Taliercio Virtual Machine (specifically, OpenCV 4.5.2).

3.2. OpenCV Inference

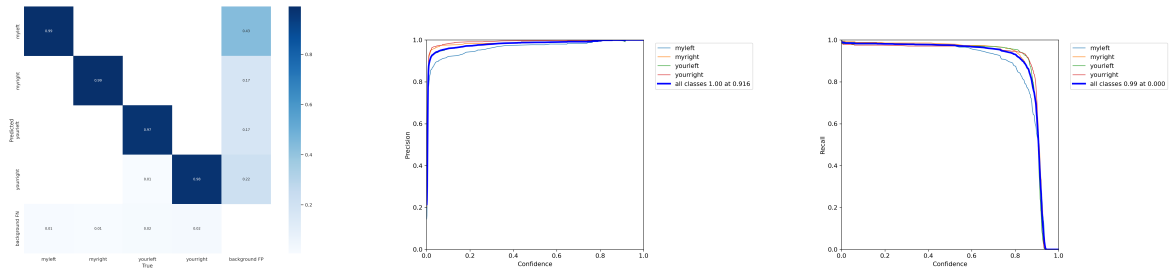
The chart in Fig. 6 describes the workflow of the inference using OpenCV. Briefly, the program works as follows:

- Convert the image into a blob (specifically, a 4D array object with the shape (1, 3, 640, 640)), in order to be processed by the network.
- Get the detection results as an object and unwrap it.
- Filter out good detection using confidence threshold
- Get the index of the best class score (for classification task) and discard detections with class scores lower than a fixed score threshold.
- Perform a Non-Maximum Suppression: this function takes a list of boxes, computes the IoU and decides to keep the bounding box depending on a fixed NMS threshold. This is done, for example, to remove overlapping and close detections.

We wrote the program in C++ using OpenCV libraries, in particular OpenCV DNN to load the model in the previously exported ONNX format, and make the predictions.

3.3. Other attempts

Before coming up with this final version of our work, we tried others pre-trained model and Deep Learning techniques. In particular, we started from a baseline model trained on YoloV5 as well, without data augmentation and trained for 100 epochs. Then, we tried using the pre-trained EfficientDet,



(a) Confusion matrix (b) Precision curve (c) Recall curve

Figure 5: Training plots

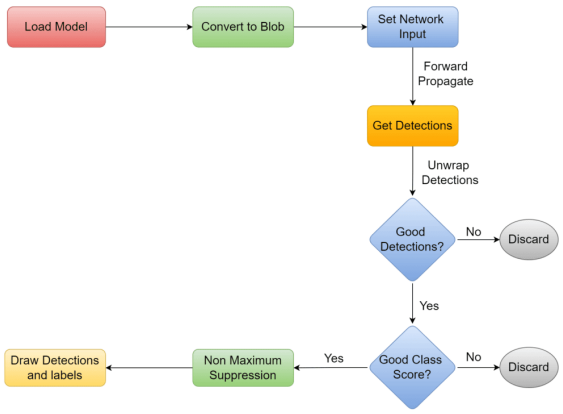


Figure 6: Workflow of C++ code

that is a scalable and efficient object detector compatible with Tensorflow. Furthermore, we tried to change the hyperparameters and do some fine-tuning to the network. Besides that, we trained several different versions of our EgoHands dataset, including different train-valid-test splits, different types of data augmentations and also gathering other different datasets found on the net. Anyway, we decided to keep the model analyzed in the previous sections since it is the best one regarding evaluation and performance on our 20 images benchmark. Our empirical results are described in section 5.

4. Hand Segmentation

4.1. K-means algorithm

One of the first attempts that we made to solve the segmentation task was the application of the K-means algorithm, due to its simplicity and speed of convergence. The use of a well known and studied algorithm like K-means gave us also the possibility to understand better the characteristics and, somehow, quantify the difficulty of the hand segmentation problems that we were trying to solve. In particular, K-means was run separately on the portions of the source image contained inside each bounding box with the number of cluster set to 2. The results obtained for each hand were then merged into a unique segmentation mask. For this segmentation problem we made the algorithm work considering three-dimensional vectors, one for each pixel inside the bounding boxes. The three elements that composed each vector were: the Cb and Cr channels of the corresponding pixel of the source image (previously converted into the YCbCr color space); the euclidean distance between the corresponding pixel and the central pixel of the bounding box. The last element (properly weighted with respect to the other two) seemed to be a good choice since the hands often appears as some sort of almost circular or elliptical blobs (especially if they are seen with the finger very close) and its presence induces the resulting clusters to assume a more circular shape.

In Fig. 7 it is possible to notice the result of the application of K-means on a frame taken from the test dataset. As we can see the algorithm succeeded into recognizing and following the countours of the interacting person hands (even if the segmentation is still irregular on the internal part) while it fails to distinguish the bottom-left hand from the wooden chessboard (that has indeed a color quite similar to the skin one). This algorithm, however was not capable of segmenting several hands that appeared in a slightly more complex position, returning not acceptable results. For this reason we decided to move to different and possibly more complex segmentation approaches.

4.2. The GrabCut algorithm

Among the several approaches that we have considered to solve the segmentation problem, the use of the GrabCut algorithm revealed to be the most effective solution. This algorithm was proposed in 2004 by Rother, Kolmogorov and Blake [3] for foreground/background segmentation and it essentially models the segmentation task as an energy minimization problem in the form of a Markov

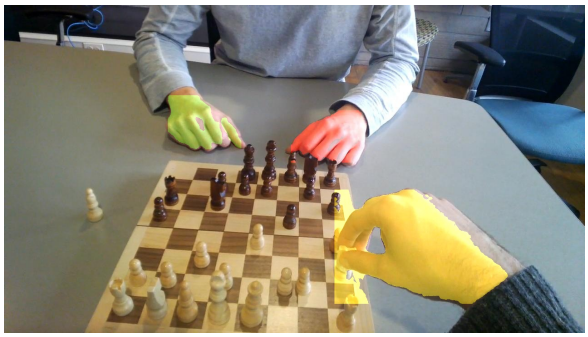


Figure 7: example of K-means segmentation result

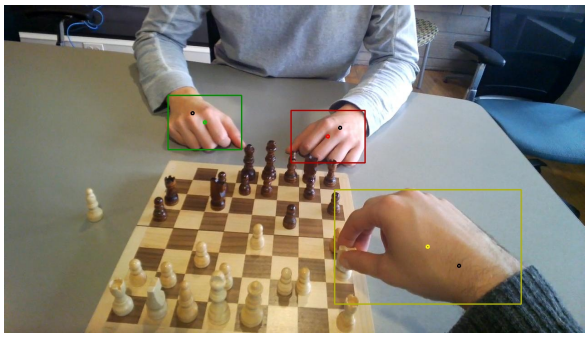


Figure 8: example of color picking points

Random Field: the energy function is composed by a data term, that evaluates how much a given label (foreground or background) fits for each pixel and a smoothness term, that penalizes discontinuity between neighboring pixels. The formulation of the data terms also exploits a Gaussian Mixture Model for modeling the foreground and background. The algorithm tries to perform an iterative minimization of the energy function, in order to obtain a good labeling of the image pixels and so a good segmentation. To start the segmentation process GrabCut needs the user to previously define a rectangle on the source image, that should completely contain the foreground object that needs to be segmented: the algorithm will consider all the pixels outside the rectangle as certain background. It is also possible to label some pixels of the source image as certain/probable foreground or background pixels, in order to provide additional information to GrabCut and so obtain a more precise segmentation.
For the above reasons the GrabCut algorithm seemed to be a good way to exploit part of the information available after the detection process, namely the location of the computed bounding boxes on the source image. Our first attempt was indeed to run the algorithm separately for each detected hand in the image, using its corresponding bounding box as starting rectangle for the segmentation. The obtained results were then merged together to obtain an unique segmentation mask, with different colors for different hands. This solution, if applied on the 20 images of the test-set, resulted to be acceptable on those images where hands were surrounded by an homogeneous background with a quite different color. However in several cases it failed to follow correctly the hands boundary, including in the segmentation parts of the image that were actually background.

4.3. Difference from skin color computation

In order to improve the performances obtained by the use of GrabCut when only a starting rectangle is specified, we tried to find a way for properly labeling the pixels inside the bounding boxes that have an high probability to be foreground or background pixels, before running the algorithm. In this way it was possible to provide additional information to GrabCut, improving the precision of its segmentation. To do so, we decided to exploit an information of the source image that is quite important in an hand segmentation task: the skin color. The idea was to obtain an estimate of the skin color for each hand by picking the color value of the central pixel of the corresponding bounding box (that almost surely will belongs to the hand). In Fig. 8 it is shown a frame taken from the considered test set, with the corresponding bounding box for each hand and the position of central pixel considered for the skin color estimation (identified by a circle of the same color of the bounding box). At this point we could take all the pixels with a color that is close enough to the estimated skin color model and consider them as probable foreground. To implement this procedure we built a gray-scale image for each detected hand, with sizes equals to the width and height of its corresponding bounding box; each pixel of a gray-scale image was set to an intensity value proportional to the difference between the corresponding pixel in the source image and the estimated skin color for the given hand. By applying this procedure, the darkest zones of each gray-scale image will correspond to the portions inside a bounding box that have a color more similar to the computed skin model. To achieve a more robust result the source image was transported into the YCbCr color space and the Y channel was not considered for the difference computation. In Fig. 9 it is possible to notice an example of the three grayscale images resulted from such process, if applied to the same image previously considered (the contrast has been enhanced to for a better visualization).

Each gray-scale image is then thresholded, exploiting the OTSU method, and the obtained binary images will be used as initial (and often coarse) segmentation masks for the GrabCut algorithm. In particular, each pixel with gray value under the computed threshold will be labeled as probable foreground; otherwise it will be labeled as probable background. The result of the threshold applied on the previous images is reported in Fig. 10 (where white is used to identify the pixels considered as probable foreground). Fig. 11, instead, shows the segmentation obtained by using such binary images as initial masks for GrabCut.

Figure 9: result of difference-from-skin computation

Figure 10: threshold of difference-from-skin image

Figure 11: result of segmentation with GrabCut

4.4. Other modifications and possible improvements

The skin color estimation approach described above could be, in some cases, not much effective. In fact the central pixel could be placed in a specific portion of the hand that has a different color or illumination condition with respect to the rest of the skin. This can happens for example in presence of shadows or light reflection phenomena. So we tried to partially overcome this problems by modifying the color picking method: we computed the skin color estimate as the average color value between the central pixel and a second pixel collocated in a specific position inside the bounding box, according to the class label associated to each hand during the detection process. In particular for each hand the second pixel will be placed in a position near the center of the bounding box but slightly moved towards the supposed location of the wrist (for example moved down and right if the hand is classified as my-right or moved up and right if the hand is classified as your-left). The collocation of this second pixel can be seen in Fig. 8 where it is identified by a small black circle inside each bounding box. It is very important to notice that this modified approach has been taken into account as a possible improvement of our project only because we are considering egocentric images of person engaged in manually activities. In these case we can expect that the great majority of the hands that appear are oriented towards the centre of the image, where the object that represents the focus of the first-person view is located. For this reasons, if we are instead considering the problem of hand segmentation in more general situations or context, this modified approach should not been chosen, rather taking into account the original method described in the previous section (that uses only the central pixel for the color picking).

5. Results

In the following we report our results regarding hand detection and hand segmentation on the 20 images of testset. We consider both the assigned metrics, that are IoU for hand detection and pixel accuracy for hand segmentation, and also IoU for segmentation as well. Regarding the detection task, the IoU (Intersection Over Union) is computed, given a ground truth bounding box, as:

$$\frac{\text{Area of overlap of ground truth and predicted bb}}{\text{Area of union of ground truth and predicted bb}}$$

Since the output of the network is not deterministic, we ran the program three times and then averaged the results. As we can note in plot 12, the augmented model is the one that performs better.

Even though the baseline model seems detecting in a very good way as well, we spotted that it misclassified some hands. In particular, 2 hands over 56 in the 20 images were not correctly classified. This fact disappeared when we fine-tuned the model, so we were able to correctly classify all the 56 instances of hands both with the fine tuned and augmented model.

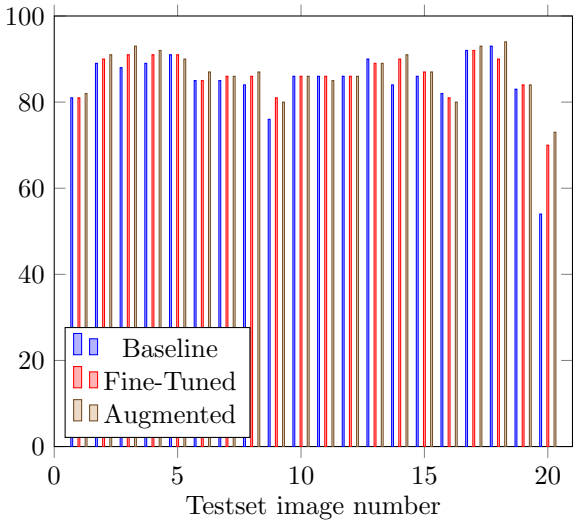


Figure 12: Histogram plot of IoU vs testset input images

The final average IoU performance for detection (not considering misclassifications) are the followings:

- Baseline model: average IoU 84,471 %
- Fine-tuned model: average IoU 86,328 %
- Augmented model (final model): average IoU 86,788 %

As mentioned above, to evaluate the goodness of our segmentation results we considered the pixel accuracy and the Intersection over Union metrics, that for clarity we consider as follows:

- $\text{PA} = \dfrac{(\#\ of\ correctly\ classified\ pixels)}{(total\ \#\ of\ image\ pixels)}$
- $\text{IoU} = \dfrac{1}{2}(\dfrac{hand\ inters.}{hand\ union} + \dfrac{not\ hand\ inters.}{not\ hand\ union})$

where with *hand inters.* we identify the number of pixels that have been correctly classified as hand, while with *hand union* we identify the number of pixels that have been classified as hand or that are labeled as hand in the considered ground truth mask. In Table 1 we report the values of the above defined metrics, obtained with the several segmentation approaches illustrated in section 4: for each

	PA	IoU
GrabCut - mask v.2	0.9835	0.888
GrabCut - mask v.1	0.9834	0.887
GrabCut - rect	0.9776	0.861
K-means	0.9760	0.851

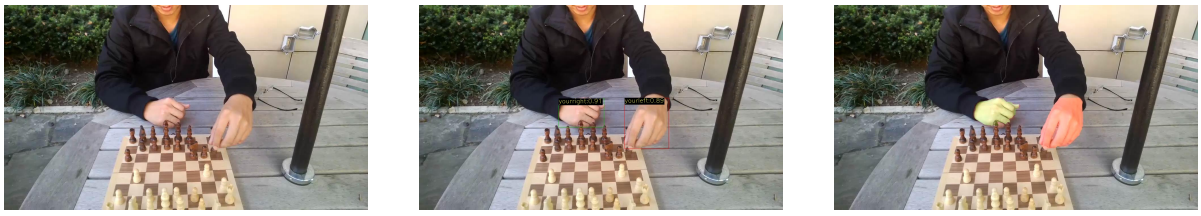
Table 1: hand segmentation performances

approach we indicate the average value obtained for the hand segmentation of the 20 images contained in the considered test-set. The expression "GrabCut - rect" identifies the GrabCut algorithm initialized only with a rectangle corresponding to the bounding box of each hand. The segmentation masks obtained with this method, even if still coarse, revealed to be more precise of the one obtained with the K-means algorithm that, as already mentioned, gave very poor results on several images of the test set. With "GrabCut - mask", instead, we identify the GrabCut algorithm initialized with a binary mask computed by considering the color difference from the estimated skin model (as detailed in section 4.3). In particular "v.1" means that the skin color has been estimated considering only the central pixel, while "v.2" identifies the modified skin color estimation method described in section 4.4. As it can be seen from the table, the introduction of the additional information provided by the initial mask allowed GrabCut to obtain a noticeable improvement of the considered performance metrics; on the other hand, the use of the modified approach for the skin color estimation introduced in "GrabCut - mask v.2" led to a very small increase of both the pixel accuracy and IoU metrics. However, since this last method revealed to be the one that returned the best result in terms of pixel accuracy and IoU values (colored in green), we decided to include it in the final version of our program. Regarding this last result, it is worth noting that it was possible to increase the number of iterations performed by GrabCut in order to obtain a very slight further increase in the PA and IoU values, paying however with a noticeable increment in the time needed to complete the segmentation process. For this reason we decided to keep the number of GrabCut iterations performed to 2 for the segmentation of each hand.

In the Fig 13, 14, 15, 16, 17 we depict some examples of detection and segmentation both of testset images and custom images taken by us.

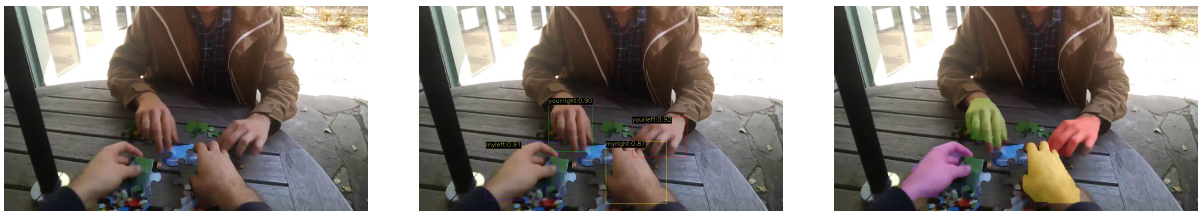
6. Conclusion and further improvements

We analyzed and discussed how to detect, distinguish, classify and segment hands in first-person



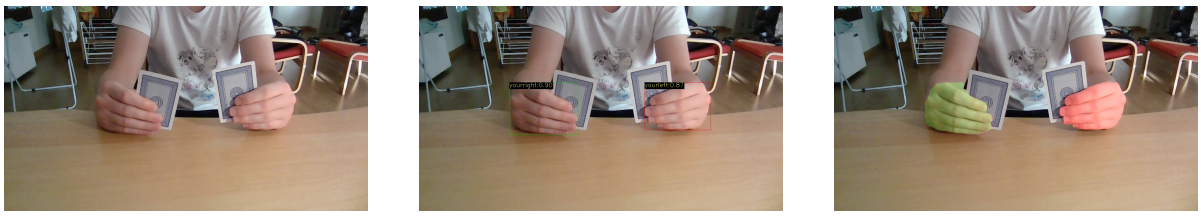
(a) Input image 03.jpg (b) Detection (c) Segmentation

Figure 13: Example 1 of detection a segmentation



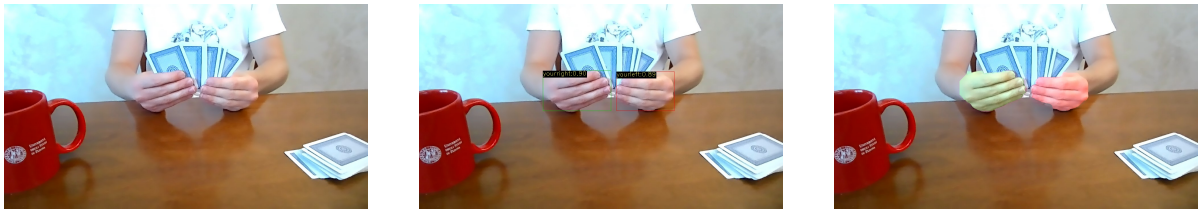
(a) Input image 14.jpg (b) Detection (c) Segmentation

Figure 14: Example 2 of detection a segmentation



(a) Input image (b) Detection (c) Segmentation

Figure 15: Custom example 1 of detection a segmentation



(a) Input image (b) Detection (c) Segmentation

Figure 16: Custom example 2 of detection a segmentation

(a) Input image (b) Detection (c) Segmentation

Figure 17: Custom example 3 of detection a segmentation

video, by combining Deep Learning object detection model and pixel-wise segmentation techniques. Some further improvements could be the generalization of our work to all type of videos, not constraining to be in first person, and the consideration of more challenging social situations, e.g with multiple interacting people moving around the scene. On top of that, another enhancement of our work can be the addition of an activity recognition task given a segmented mask of the hands, i.e., the algorithm should be able to understand what activity the person is performing.
par Finally, we report the following required information:

- **Number of working hours:** around 80-90 hours each
- **Github project link** (make project public)

Acknowledgments: thanks to learnopencv.com for the OpenCV inference tutorial, Indiana University for the dataset, Roboflow for creation and augmentation of custom dataset.

References

[1]S. Bambach, S. Lee, D. J. Crandall, and C. Yu, «Lending a hand: detecting hands and recognizing activities in complex egocentric interactions», in The ieee international conference on computer vision (iccv) (Dec. 2015).

[2]A. U. Khan and A. Borji, *Analysis of hand segmentation in the wild*, Center for Research in Computer Vision, University of Central Florida, (2018) https://arxiv.org/pdf/1803.03317v2.pdf.

[3]C. Rother, V. Kolmogorov, and A. Blake, *Grabcut: interactive foreground extraction using iterated graph cuts*, Microsoft Research Cambridge, UK, (2004) https://www.researchgate.net/publication/220184077_GrabCut_Interactive_Foreground_Extraction_Using_Iterated_Graph_Cuts.