Computer Vision - Summer Project 2022

Bastasin Simone, Lotta Alessandro, Varotto Elisa

July 29, 2022

1 Our approach

The goal of this project is to design and develop a system capable of detecting and segmenting human hands in an input image.
We decided to develop our project using C++ based on the OpenCV library with a small implementation of a network in Python.
We searched for the most suitable and reliable network and we found YOLOv5 (Version 5).
YOLO (You Only Look Once) is a real-time object detection algorithm that identifies specific objects in videos or images.
YOLO proposes the use of an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once. YOLO's architecture has a total of 24 convolutional layers with 2 fully connected layers at the end.
The YOLO algorithm works by dividing the image into N grids, each having an equal dimensional region of SxS. Each of these N grids is responsible for the detection and localization of the object it contains. These grids predict B bounding box coordinates relative to their cell coordinates, along with the object label and probability of the object being present in the cell.
YOLO has the inherent advantage of speed, an increasing accuracy in predictions and a better Intersection over Union in bounding boxes (compared to real-time object detectors).
We started with a pre-trained network and then we obtained our model with the use of other databases.
Firstly we implemented and trained the network on a quite large set of image called EgoHands, with high quality image, then for a more accurate training we also used HandOverFace(HOF) dataset.
The complete dataset used for training with the related ground truth is available at the following link: cv_dataset
We used YOLOv5 in the medium version (YOLOv5m).
We decided to use 500 epochs for the training set. The number of epochs define the number of times that the learning algorithm will work through the entire training dataset.
We chose to use a validation set during the training of the network.
YOLO creates some graph that are useful for the creation of a good model. We used the graphs of the mAP (mean Average Precision), Figure (1), to evaluate the best number of epochs, which is 425. We obtained this number by looking at the highest number that the validation mAP reaches.
OpenCV provides a DNN (Deep Neural Network) module. This module lets you use pre-trained neural networks from popular frameworks like TensorFlow, PyTorch etc, and use those models directly in OpenCV. It means that it is possible to train models using a popular framework like Tensorflow and then do predictions with just OpenCV.

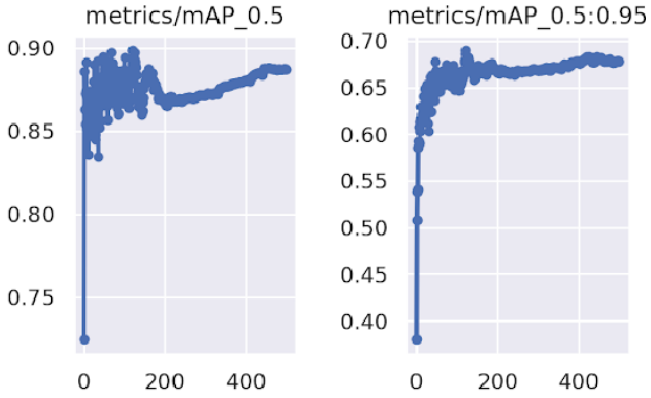

Figure 1: mAP graph from YOLO

From OpenCV DNN module we used the function *readNet* to read the netwok.
At this stage the system read an input image and in such image it will perform the hands detection and segmentation.
We implemented a function called Pre-Process where we computed the blob of the input image and we used it as an input of the model we obtained from the network training.
We ran the OpenCV function *forward* to compute the output of the classifier.
The following step is the Post-Process.
We started by deleting all the detection boxes' scores, called confidences, that are above a definite threshold, equal to 0.45. Then we performed the Non-Maximum Suppression, where we kept the detection boxes with highest confidence score. The remaining boxes were drawn on the image with their confidence score.
For the hand segmentation we decided to use GrabCut Algorithm develop for OpenCV, this algorithm is able to extract the foreground with minimal user interface.
GrabCut Algorithm considers a detection box as a rectangle containing the foreground region, everything outside this rectangle will be taken as sure background. Then the algorithm tries to label the foreground and background pixels inside the rectangle and a Gaussian Mixture Model (GMM) is used to model the foreground and background. A graph is built from this pixel distribution and every foreground pixel is connected to the Source node and every background pixel is connected to the Sink node. The weights are defined by the probability of a pixel being foreground/background and a mincut algorithm is used to segment the graph. GrabCut Algorithm cuts the graph into two separating source node and sink node with minimum cost function. The process continues until the classification converges.
At the beginning we tried to use different methods to obtain a good segmentation result, for example the Watershed Algorithm, Otsu's thresholding and the K-means Cluster. But at the end we decided to use the GrabCut Algorithm because it gave us an higher pixel accuracy.

2 Performance measurement

The Intersection over Union (IoU) is the intersecting area between the bounding boxes for a particular prediction and the ground truth bounding boxes of the same area.
The Intersection over Union (IoU) provides a good estimate of how close the bounding box is to the original prediction.
We noticed that in some images there were more than one hand. So we decided to calculate the number of predicted boxes that have their corresponding ground truth boxes, for each of these we calculated the IoU and then we summed all the IoU together. Finally, we divided the

final result of the previous calculation for the number of the total predicted boxes.
Pixel Accuracy (PA) is the percentage of pixels in the image that are classified correctly.
Pixel Accuracy:

$$PA = \frac{TP + TN}{TN + TP + FP + FN} \tag{1}$$

where:

- TP : True Positives, number of correct matches;
- TN : True Negatives, number of correct non-matches;
- FP : False Positives, number of non-matches that were wrongly matched;
- FN : False Negatives, number of matches that were wrongly missed.

The results we obtained are shown below, both for the hand detection and hand segmentation (output images and metrics values) for the test images in the provided benchmark dataset.

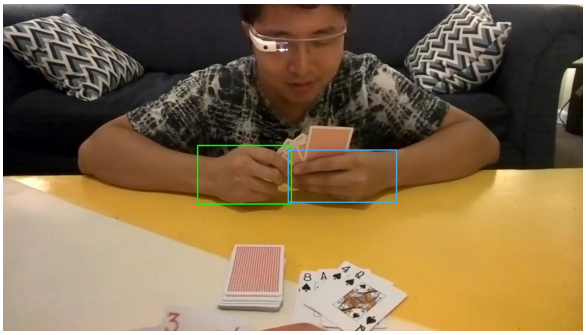

Figure 2: Detection

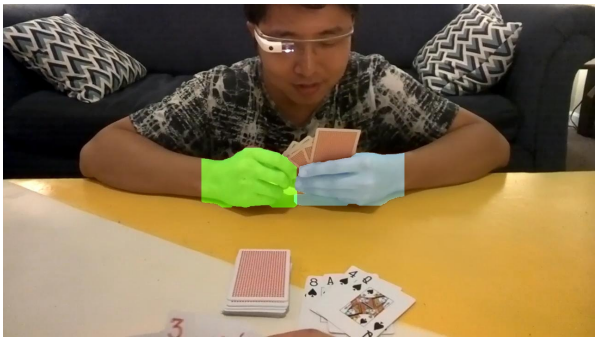

Figure 3: Segmentation

Image 01 - IoU: 0.860466 , Pixel Accuracy: 0.987832

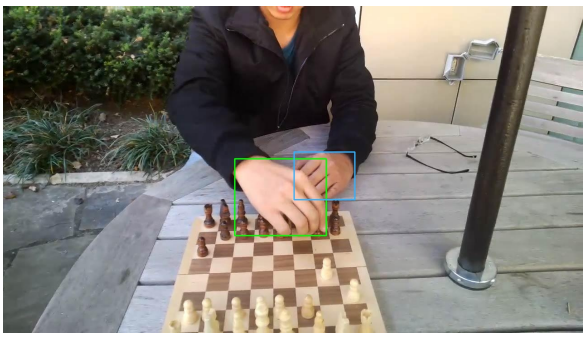

Figure 4: Detection

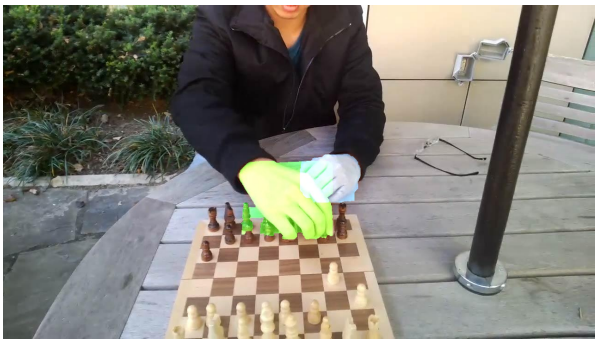

Figure 5: Segmentation

Image 02 - IoU: 0.940299 , Pixel Accuracy: 0.993400

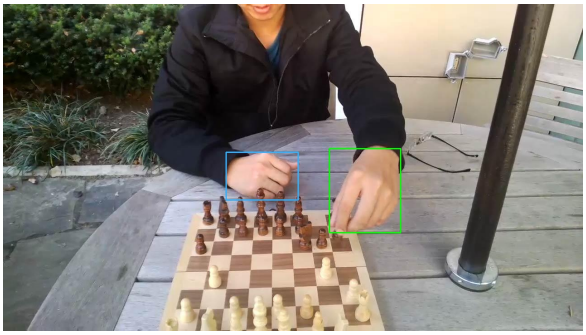

Figure 6: Detection

Figure 7: Segmentation

Image 03 - IoU: 0.941618 , Pixel Accuracy: 0.995490

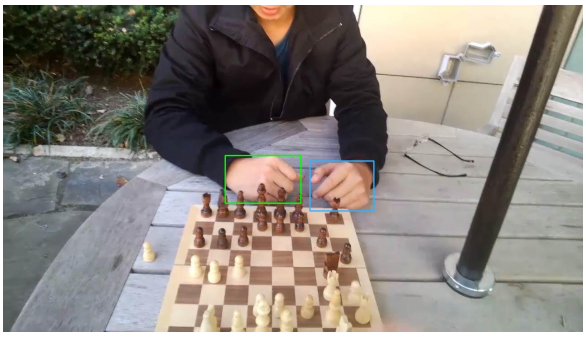

Figure 8: Detection

Figure 9: Segmentation

Image 04 - IoU: 0.969308 , Pixel Accuracy: 0.994996

Figure 10: Detection

Figure 11: Segmentation

Image 05 - IoU: 0.917267 , Pixel Accuracy: 0.987582

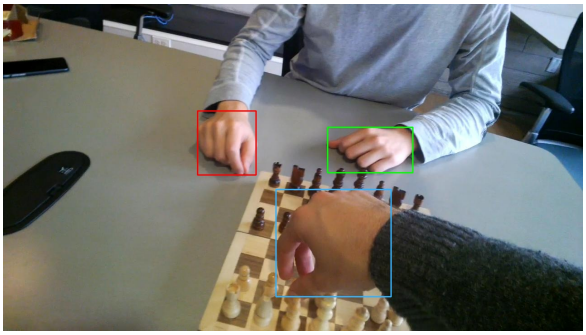

Figure 12: Detection

Figure 13: Segmentation

Image 06 - IoU: 0.880801 , Pixel Accuracy: 0.984575

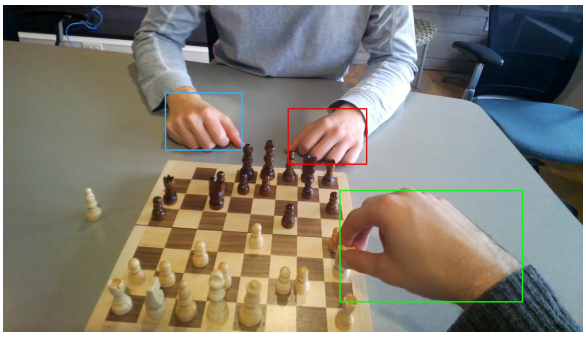

Figure 14: Detection

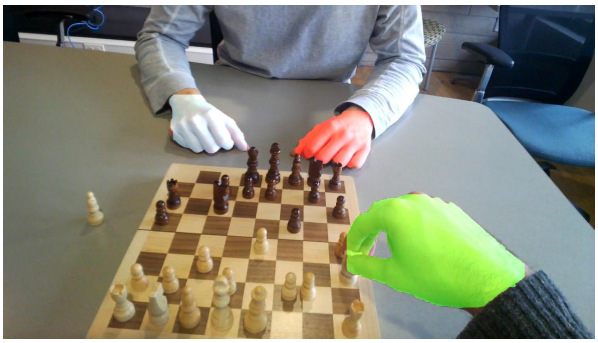

Figure 15: Segmentation

Image 07 - IoU: 0.889847 , Pixel Accuracy: 0.988800

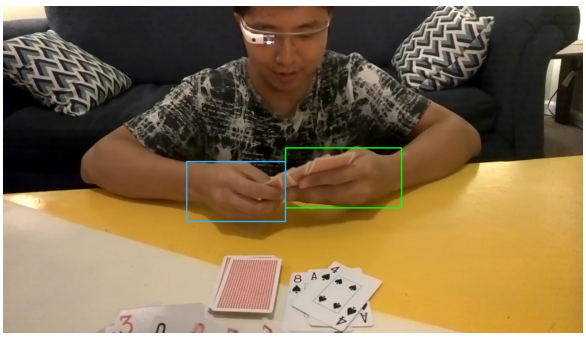

Figure 16: Detection

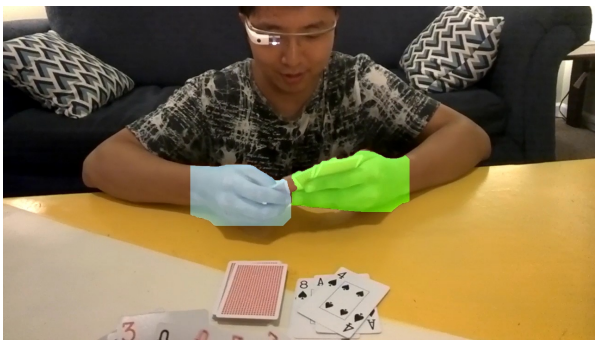

Figure 17: Segmentation

Image 08 - IoU: 0.893077 , Pixel Accuracy: 0.983980

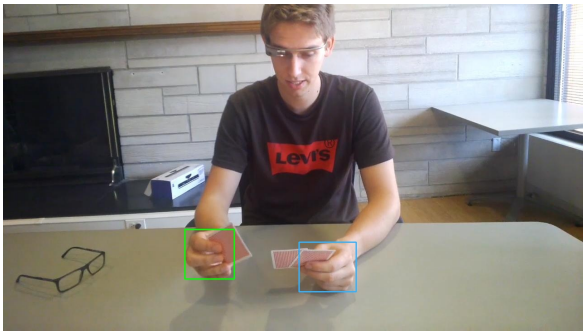

Figure 18: Detection

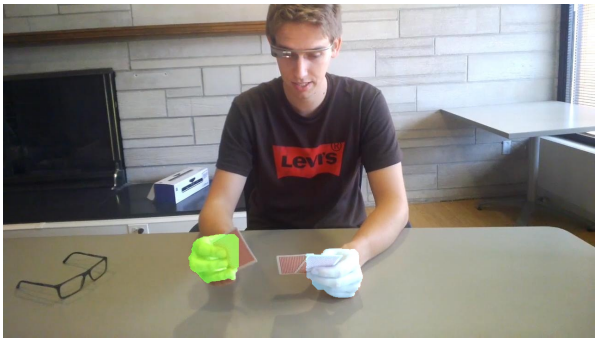

Figure 19: Segmentation

Image 09 - IoU: 0.822655 , Pixel Accuracy: 0.992240

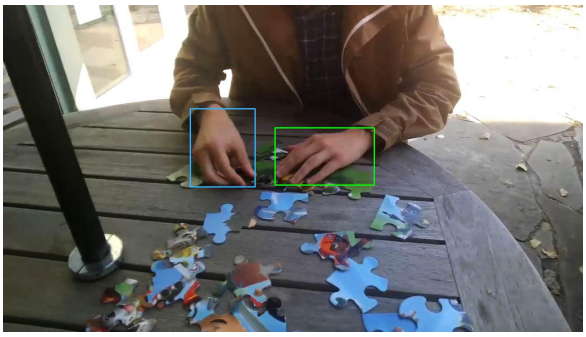

Figure 20: Detection

Figure 21: Segmentation

Image 10 - IoU: 0.889210 , Pixel Accuracy: 0.988985

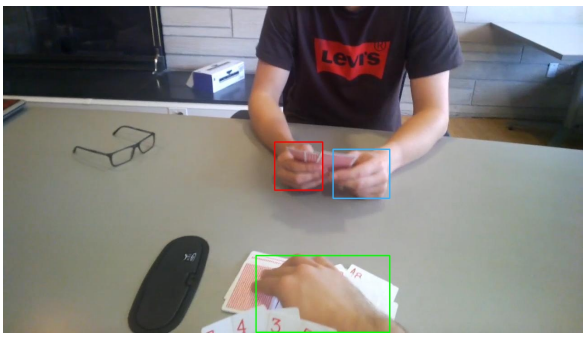

Figure 22: Detection

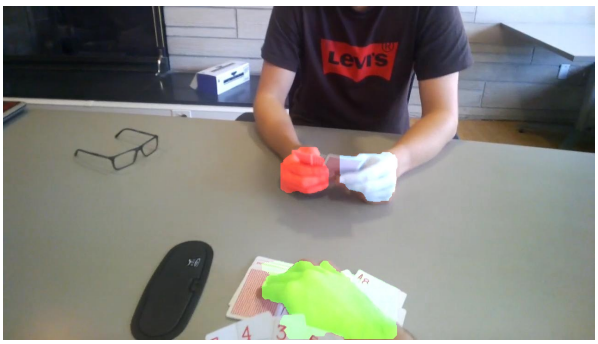

Figure 23: Segmentation

Image 11 - IoU: 0.875268 , Pixel Accuracy: 0.977142

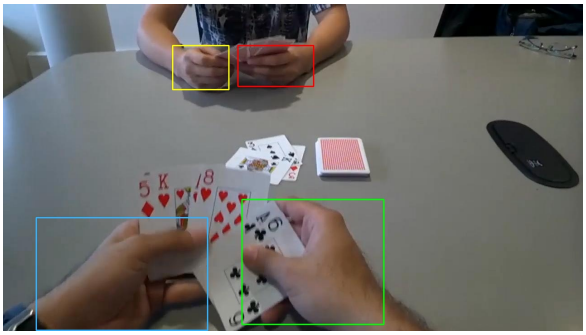

Figure 24: Detection

Figure 25: Segmentation

Image 12 - IoU: 0.911366 , Pixel Accuracy: 0.958416

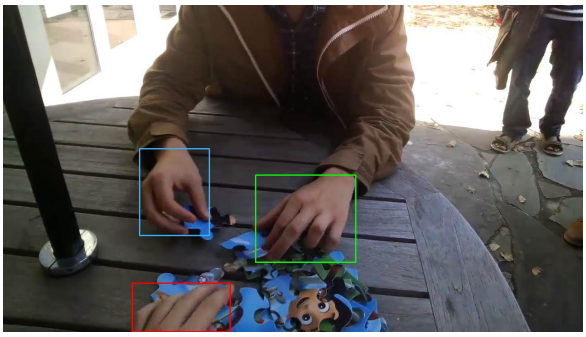

Figure 26: Detection

Figure 27: Segmentation

Image 13 - IoU: 0.938042 , Pixel Accuracy: 0.975809

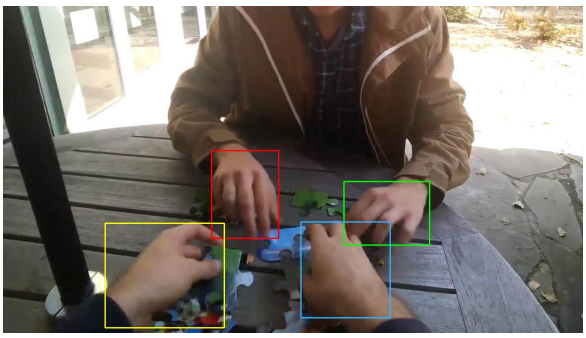

Figure 28: Detection

Figure 29: Segmentation

Image 14 - IoU: 0.914361 , Pixel Accuracy: 0.965629

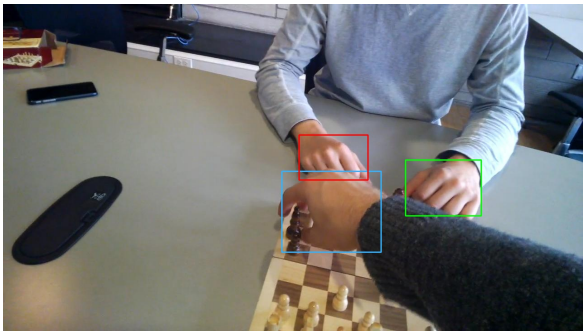

Figure 30: Detection

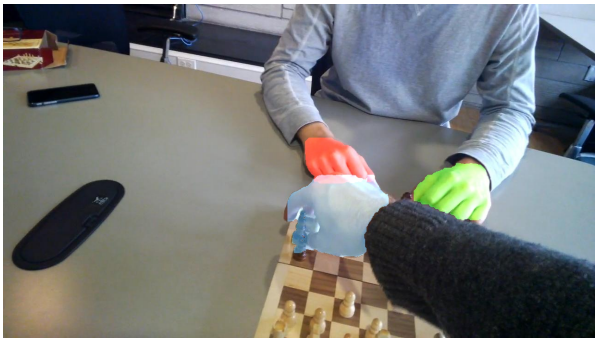

Figure 31: Segmentation

Image 15 - IoU: 0.899281 , Pixel Accuracy: 0.985834

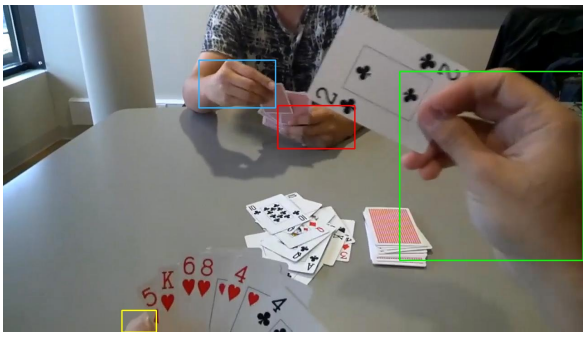

Figure 32: Detection

Figure 33: Segmentation

Image 16 - IoU: 0.855415 , Pixel Accuracy: 0.947317

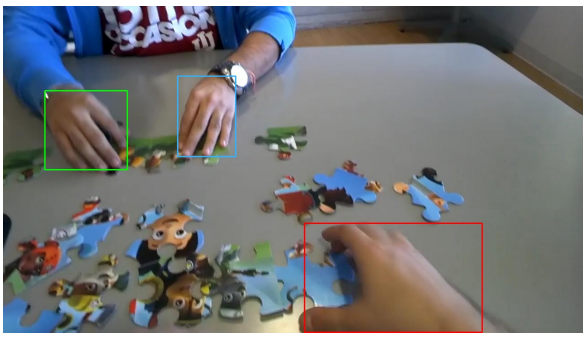

Figure 34: Detection

Figure 35: Segmentation

Image 17 - IoU: 0.929661 , Pixel Accuracy: 0.972319

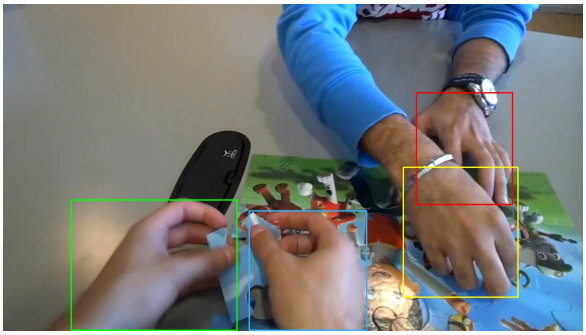

Figure 36: Detection

Figure 37: Segmentation

Image 18 - IoU: 0.967216 , Pixel Accuracy: 0.957868

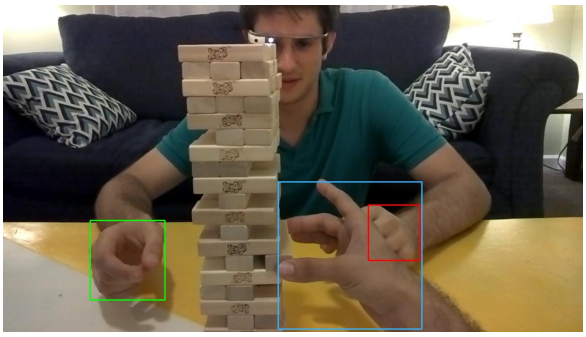

Figure 38: Detection

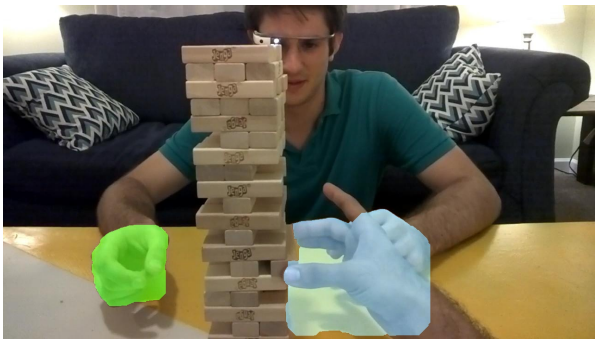

Figure 39: Segmentation

Image 19 - IoU: 0.841524 , Pixel Accuracy: 0.957613

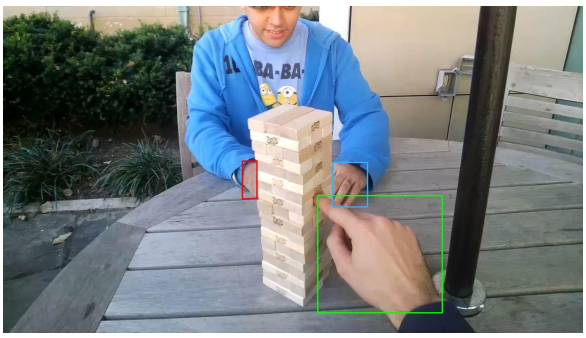

Figure 40: Detection

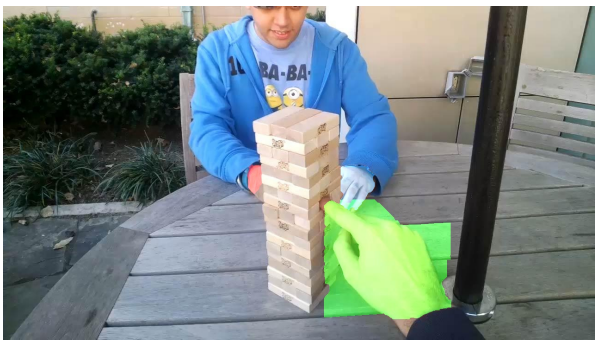

Figure 41: Segmentation

Image 20 - IoU: 0.762129 , Pixel Accuracy: 0.977726

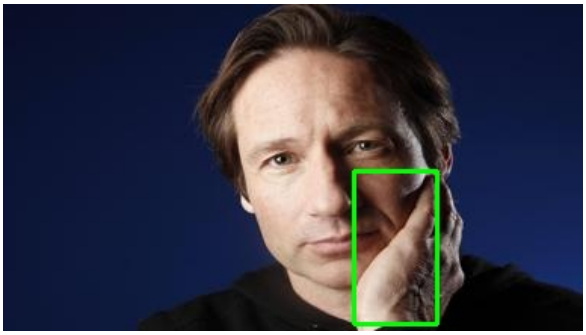

Figure 42: Detection

Figure 43: Segmentation

Image 21 - IoU: 0.712559 , Pixel Accuracy: 0.976345

Figure 44: Detection

Figure 45: Segmentation

Image 22 - IoU: 0.433200 , Pixel Accuracy: 0.962023

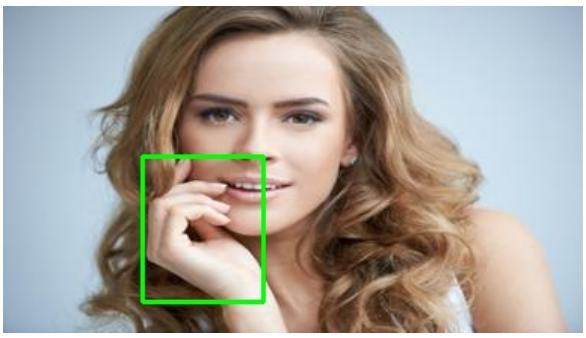

Figure 46: Detection

Figure 47: Segmentation

Image 23 - IoU: 0.953584 , Pixel Accuracy: 0.990777

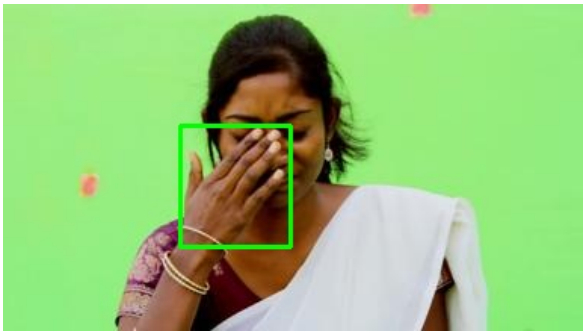

Figure 48: Detection

Figure 49: Segmentation

Image 24 - IoU: 0.707428 , Pixel Accuracy: 0.984218

Figure 50: Detection

Figure 51: Segmentation

Image 25 - IoU: 0.416913 , Pixel Accuracy: 0.958369

Figure 52: Detection

Figure 53: Segmentation

Image 26 - IoU: 0.833532 , Pixel Accuracy: 0.978431

Figure 54: Detection

Figure 55: Segmentation

Image 27 - IoU: 0.402351 , Pixel Accuracy: 0.961323

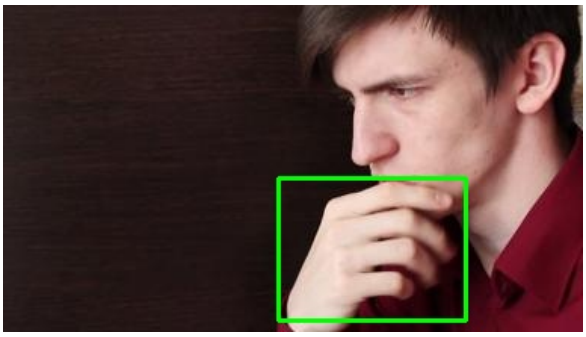

Figure 56: Detection

Figure 57: Segmentation

Image 28 - IoU: 0.891032 , Pixel Accuracy: 0.979601

Figure 58: Detection

Figure 59: Segmentation

Image 29 - IoU: 0.780488 , Pixel Accuracy: 0.984098

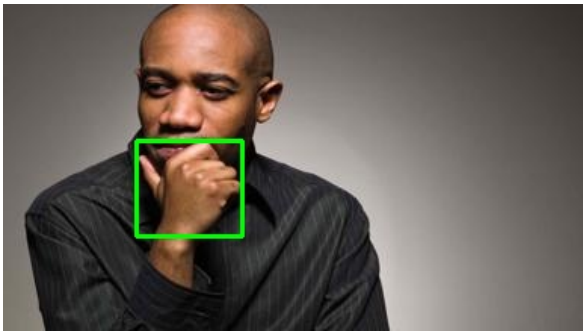

Figure 60: Detection

Figure 61: Segmentation

Image 30 - IoU: 0.942382 , Pixel Accuracy: 0.991440

3 Contribution of each member of the group

We worked as a group and we helped each other to reach the goal. We divided the work approximately like this:

- Bastasin Simone did the implementation of the neural network in Python, detection on C++, segmentation on C++ and implementation of the IoU metric;
- Lotta Alessandro did the implementation of the neural network in Python, detection on C++, segmentation on C++ and implementation of the pixel accuracy metric;
- Varotto Elisa did detection on C++, segmentation on C++ and the report on LATEX.

We approximately dedicate 55-60 hours for each of us.