Report Final Project CV

GROUP MEMBERS: Bresolin Paolo, Gonella Giacomo, Picardi Pietro

Introduction

Our group is composed by: Paolo Bresolin, Giacomo Gonella and Pietro Picardi.
Our solution for the project comes from a constant dialogue, where each one of us has brought an equally valuable contribution in terms of creative ideas and practical implementations at every step of the process. Therefore we firmly believe that a strong division of tasks does not represent the actual way we worked: everyone took part in the decision process, solving anomalies and errors or pointing out better strategies to perform a specific assignment.
However if we must specify the main contributions and ideas brought up by every member of the group, the following rough division emerges:

- Paolo: import of the NN and display of the bounding boxes, implementation of region growing for image segmentation;
- Giacomo: selection of the NN model for image detection and implementation of the training procedure to obtain new weights, implementation of region growing for image segmentation;
- Pietro: use of different color spaces and filters to optimize the starting point for region growing, implementation of evaluation metrics and their display on the final image, random coloring of masks and bounding boxes, display of segmentation masks.

Moreover we created a specific dataset collecting images from both the internet and already existing datasets too (Egohands, HandsOverFace); this task was equally performed by every member of the group, since the total number of images has been divided into three parts.
Finally we did not assiduously keep track of the time spent by each one of us in researching and implementing our solution, nevertheless we believe that a good and honest estimate is about 75 hours of work per member.
According to our programming experience, we decided to group all the functions we wrote in different libraries. Each library is related to a particular task and contains the functions written to solve it. Since different sections of functions have been written by different members of the group, we wrote before each section itself, as a comment, the name of the person who wrote it. In any case, we would like to point out that every one of us contributed to the whole project, finding errors and fixing issues in any function and part of code. As already pointed out, we cooperated very well.

Our idea is the following: we use a neural network to find the bounding boxes containing, possibly, all the hands in the image, then we apply some techniques studied in the computer vision course to deal with hand segmentation, considering each bounding box separately. Finally we show all results in a single image.
As long as segmentation is concerned, we used OpenCV in C++, whereas for retraining the neural network we used Python and Colab.
We also considered using a mask R-CNN to directly segment the hands and then, starting from the segmentation mask itself, extracting the bounding boxes. This approach, probably, could have led to better results (especially in terms of segmentation and pixel accuracy), but we decided not to follow such framework for two main reasons:

- differentiating hands in an image, given the segmentation mask, can be impossible in some common situations (for example, overlapping hands);
- most important reason: in this way our use of OpenCV would be limited to just find the bounding boxes, while most of the effort would be done by the NN. We believe it is more fair trying to use studied techniques rather than an automatic process, regardless of the possibility of better results.

Detection with YoloV3

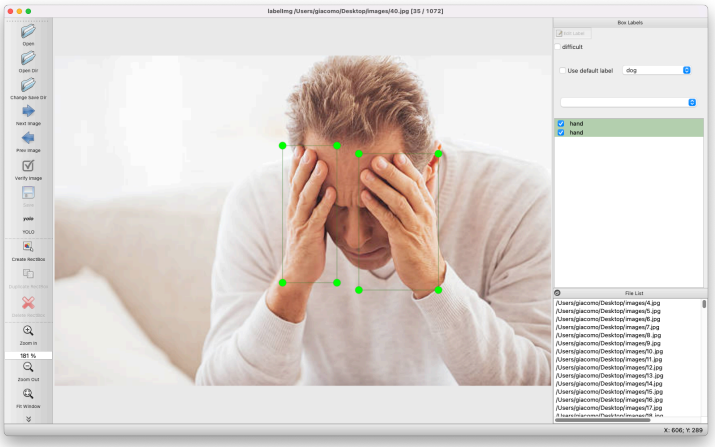
We decided to detect the bounding boxes with a neural network, the YoloV3 (You Only Look Once) of the Darkness Neural Network Framework.
To train the YOLO we used Google Colab, a cloud computing service that gives us the computing power of a GPU.
Before starting with this solution, we also tried to develop our own neural network using Keras, but in the end we chose to use an existing architecture. This is mainly because the network we need to detect the bounding boxes is quite complex. In any case, we were able to understand how to build a bounding box regression neural network model, however we could not solve the following problem: in an image there can be any number of hands, whereas we were able to deal only with images containing at most one hand. Thus, we thought about how to improve our model and finally came out with the idea of using YoloV3 with Darknet.

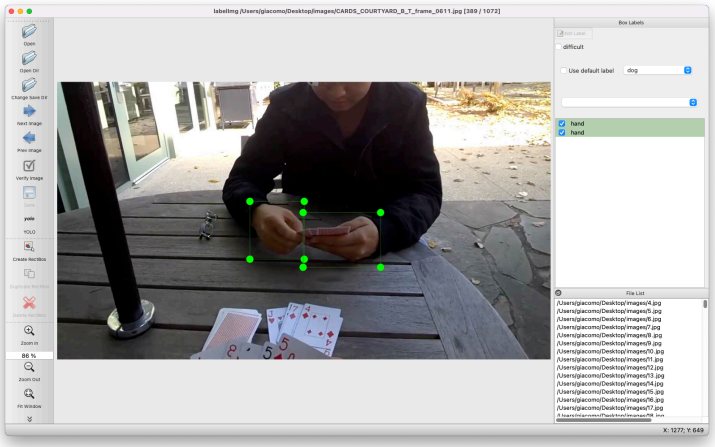
Dataset

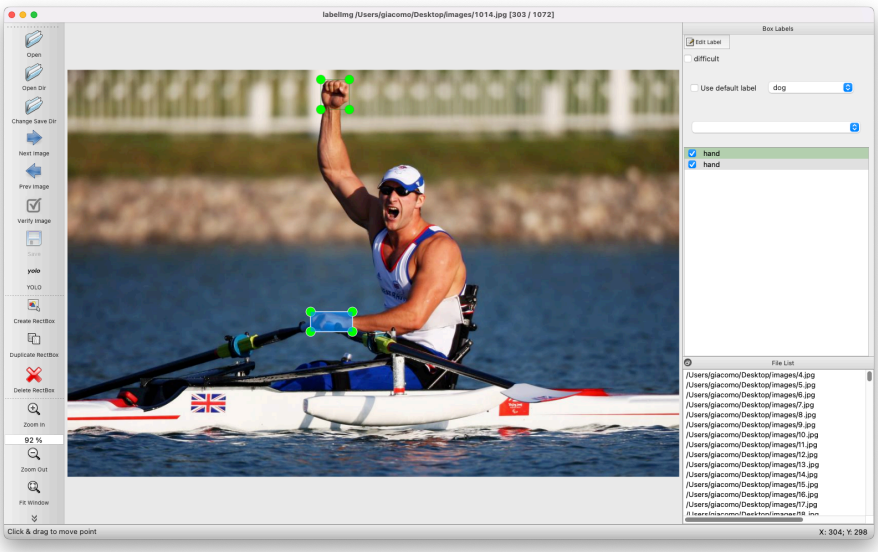
To build the ground truth, we put together more than 1,000 images, many taken directly from the EgoHands and handsOverFace datasets and many others taken from the Internet. We tried to collect as many scenarios as possible, here is a list of a few particular situations:

- hands in front of the face;
- hands on top of other hands;
- different skin colors;
- grayscale images;
- hands only partially present (i.e. only the palm or a few fingers are actually visible).

We used an open source software called LabelImage, through which we were able to create the ground truth for the detection task.







Training

As previously mentioned, we used Google Colab for training the NN. This service allowed us to run a Python script by taking advantage of the power of a video card for free.
To use the images we created and linked a drive folder to the script in Colab.
Once we click "Run all" the script will begin the training: due to system limitations the program can run for a maximum of 12 hours. The script will automatically save every last weights created and, as a backup, the weights after every 1000 epochs. Obviously using the last weights we will get the best result, at least in terms of training error.
We repeated the training process four times, in order to better understand how to improve the final results. Based on the results of the previous trainings, we modified the dataset, in order to get better performances. For example, at the beginning we did not include grayscale images, but when we saw that our model was not able to recognize hands in such images, we expanded the training set by including a suitable (in our opinion) subset of grayscale images. Again, for example, we added also a consistent number of overlapping hands and images with low sharpness. Moreover, we experimented training of different time lengths, in order to compare the results and understand how the training time practically affects the quality of predictions.
Notice that each member of the group did at least one entire training experience in Colab.

Import the network in OpenCV

Now that we have obtained the network weights for bounding box detection, we just have to import them into our source code in C++ using the libraries and the appropriate functions provided by OpenCV.

```
#include <opencv2/dnn.hpp>
```

```
//Load the network
Net net = readNetFromDarknet(modelConfiguration, modelWeights);
net.setPreferableBackend(DNN_BACKEND_OPENCV);
net.setPreferableTarget(DNN_TARGET_CPU);
```

OpenCV

After importing YoloV3 with the respective weights, we can use its predictions to find the bounding boxes of the hands in the test set.
For each hand we will read the respective ground truth (bounding box and mask) with which we will later evaluate the results of detection and segmentation.
On the one hand, in order to read the masks, it will be sufficient to use the "*imread*" function (since they are simple images) provided by OpenCV.
On the other hand, the procedure for bounding boxes is a bit more complex: the coordinates are provided in a .txt file, so we used an auxiliary function to read the text file and insert these coordinates into an ad hoc structure.

```
vector<SCORE::Box> true_bb;
getTrueBoundingBoxes(filenames_det[i], &true_bb, &: img);
```

```
void getTrueBoundingBoxes(string path, vector<SCORE::Box>* true_bb, Mat& img) {
    FILE* inDet;
    int x, y, width, height;
    inDet = fopen((path).c_str(), "r");
    if (inDet == NULL) {
        cerr << "Error: file could not be opened" << endl;
        exit(1);
    }
    while (!feof(inDet)) {
        SCORE::Box b;
        fscanf(inDet, "%d", &x);
        fscanf(inDet, "%d", &y);
        fscanf(inDet, "%d", &width);
        fscanf(inDet, "%d", &height);
        b.p1 = SCORE::Vec2<double>(x, y);
        b.p2 = SCORE::Vec2<double>(x + width, y);
        b.p3 = SCORE::Vec2<double>(x + width, y + height);
        b.p4 = SCORE::Vec2<double>(x, y + height);
        (*true_bb).push_back(b);
    }
    fclose(inDet);
}
```

After obtaining the ground truth for the evaluation, we proceed to procure the predictions of the neural network.
To do this we use the function "*blobFromImage*" to obtain a characteristic input for our net. With the "*forward*" function applied to our net we get the output, which is a vector containing the predicted bounding boxes.
Now we have to process the data that we gathered.

Eliminate low confidence Bounding Boxes

We populate the vector of bounding boxes, now in the more convenient "starting point-dimension" format than before, and the confidence vector for each one of them.
We apply a Non Maxima Suppression on the boxes thanks to the "*NMSBoxes*" function provided by OpenCV, using the confidences.

```
NMSBoxes( bboxes: boxes, scores: confidences, score_threshold: 0.5, nms_threshold: 0.4, &: indices);
```

Bounding Boxes on the image

To distinguish the various hands, we used a randomized color to characterize each one of them both for the detection box and the segmentation mask.

```
colors.push_back(Scalar(
        v0: (double) rand() / RAND_MAX * 255,
        v1: (double) rand() / RAND_MAX * 255,
        v2: (double) rand() / RAND_MAX * 255
));
```

We enter the coordinates of the four vertices of the bounding box into an ad hoc structure (the same one previously used to store the “true” bounding box) that we will use to get an assessment of how good the detection accuracy is.

```
SCORE::Box b = true_bb.at(true_bb.size() - 1 - i), b_predicted;
Rect box = boxes[idx];

b_predicted.p1 = SCORE::Vec2<double>(box.x, box.y);
b_predicted.p2 = SCORE::Vec2<double>(box.x + box.width, box.y);
b_predicted.p3 = SCORE::Vec2<double>(box.x + box.width, box.y + box.height);
b_predicted.p4 = SCORE::Vec2<double>(box.x, box.y + box.height);
iou_score.push_back(computeIou(b, b_predicted));
```

Next in the code we will draw the predicted bounding box using the “*rectangle*" function of OpenCV.

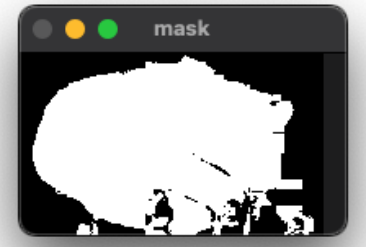
```
//Draw the predicted bounding box
rectangle(img, pt1: Point( x: box.x, y: box.y), pt2: Point( x: box.x + box.width, y: box.y + box.height), color: colors[i], thickness: 2);
```

Masks on the image

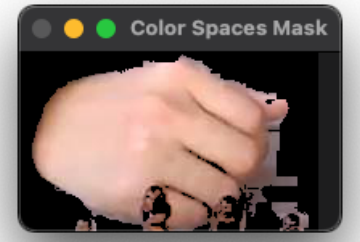
To obtain the segmentation mask, before reaching our final version, we tried several approaches that proved to be unsuccessful. Some of them are listed below:

- after smoothing the sub-image highlighted by the predicted bounding box with a bilateral filter, we applied the *Canny* function *for edge detection*, using then the *closure* operation to close the obtained shape. The problem with this approach was that the actual closure of the figure almost never occurred and so we were not able to obtain a closed shape to be filled by using, for example, the OpenCV function "fillPoly".
- Use of an *adaptive threshold*, here the initial result was good, however it only gave a vague idea of the contours of the hand, often including parts of other objects.
- We also tested the use of snakes, even so we realized that to make this approach efficient we would have to manually enter the initial points.

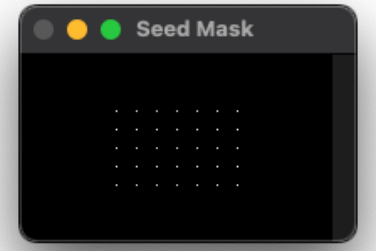
Our final approach is to use different color spaces (RGBA, YCrCb and HSV) to extract a mask extracting the skin from the sub-image, the results are quite good even though not perfect.
To understand how to obtain the skin through this method, we consulted the paper [“Human Skin Detection Using RGB, HSV and YCbCr Color Models”](#).



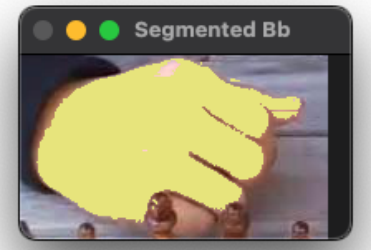
After pulling out the mask, we applied a simple threshold to obtain its binary version.



At this point we opted for region growing as segmentation method.
We also tried to blur the image before applying region growing, especially by using a bilateral filter with different settings of the parameters, but it did not lead to better results.
Thanks to the binary mask, we applied the function "*selectInitialPoints*" to obtain the seed points from which we run our segmentation algorithm.
This function is very simple: it selects a series of points spaced around the central point, based on the pixel values in a neighborhood around the center; if it finds no suitable point, then it returns only the central point.

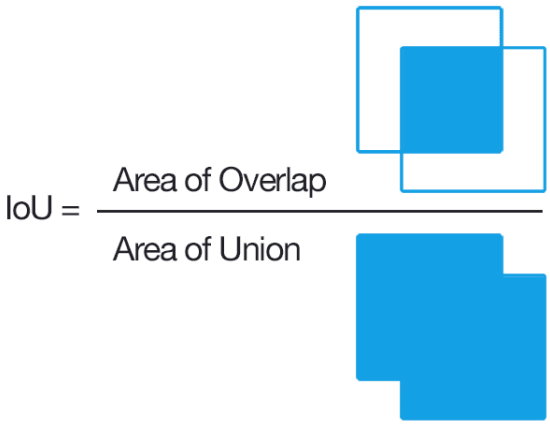


Our region growing is also quite simple: from each initial point we expand by considering points which color is similar to the average of the initial points, but only if there is at least one neighbor that has been already selected as part of the mask (i. e. the region).



Evaluation

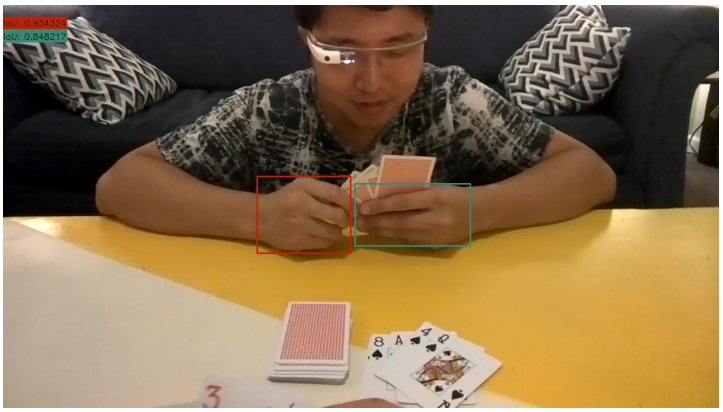
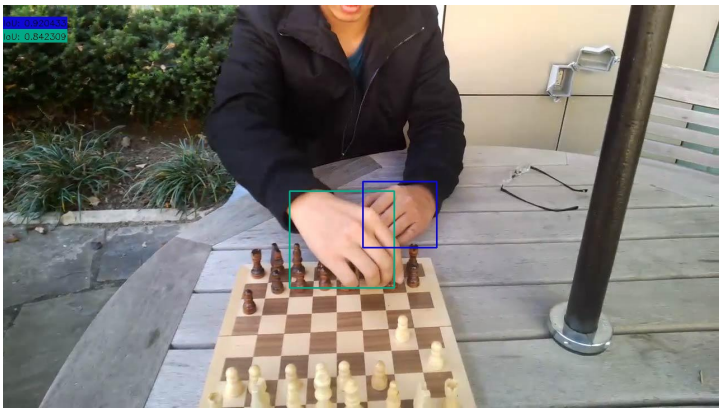
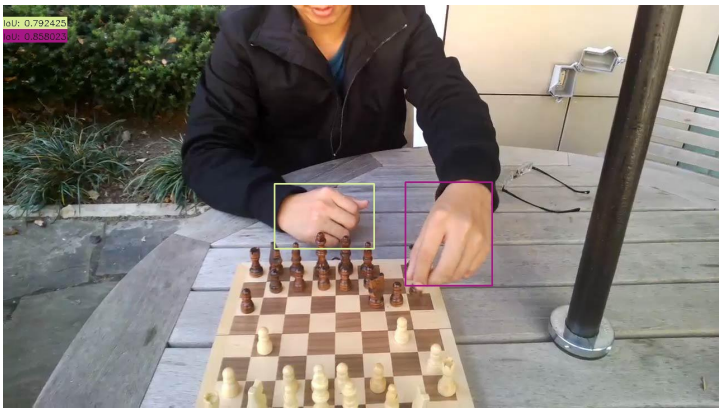
As far as the scoring of the detection results is concerned, the suggested method (intersection over union) required us to create a specific set of structures and methods all of which are contained in the files "*score.cpp*" and "*score.h*" . The predicted bounding boxes are quite accurate, however none of them gets a perfect/nearly perfect score. This is due to the way the ground truth is provided (both in the test set and in the training of the NN): each “true” bounding box is manually selected, therefore errors can occur. Consequently a good result will have a score in the range [0.75, 0.95] .

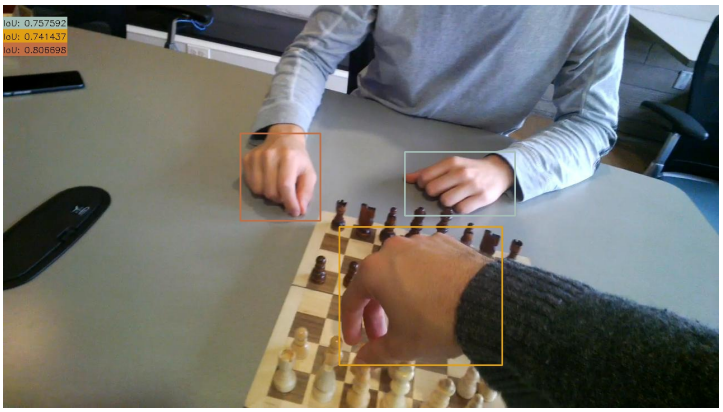
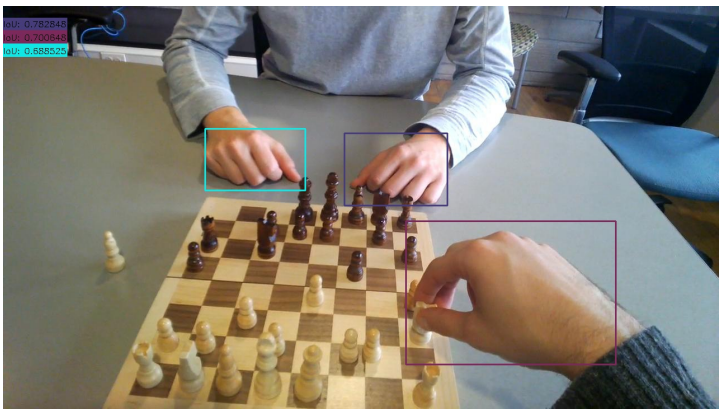
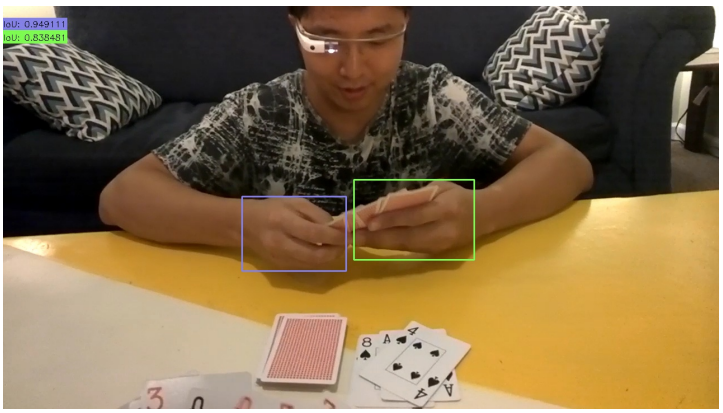


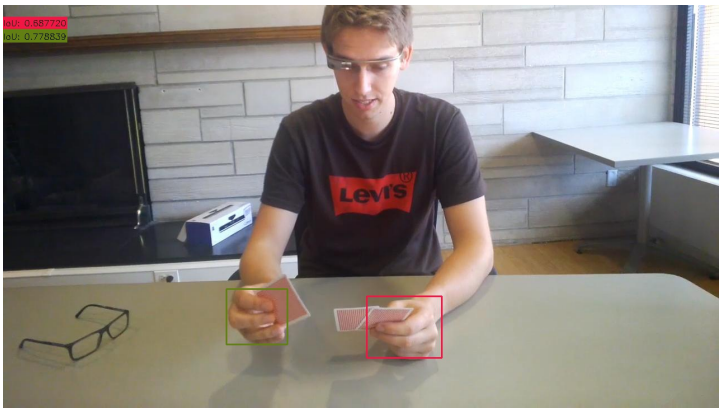
On the other hand, the evaluations of the segmentation results happens via pixel accuracy. This method required us to compute the ratio of common pixels between the predicted mask and its ground truth over the total amount of pixels in the whole image.
As previously done with detection, an annotation about scores is necessary. Pixel accuracy will always provide a high score (above 0.9), nevertheless this does not mean that segmentation was excellently performed. This is due to the nature of masks: they are binary image mostly black with few small white areas (corresponding to hands). Therefore high scores come from black common pixels, rather than white ones. We noticed, however, that a good segmentation provides a score > 0.96 , while a mediocre/approximate result will return a ~0,95 .
Scores are displayed on the top left corner of the images returned by our program.

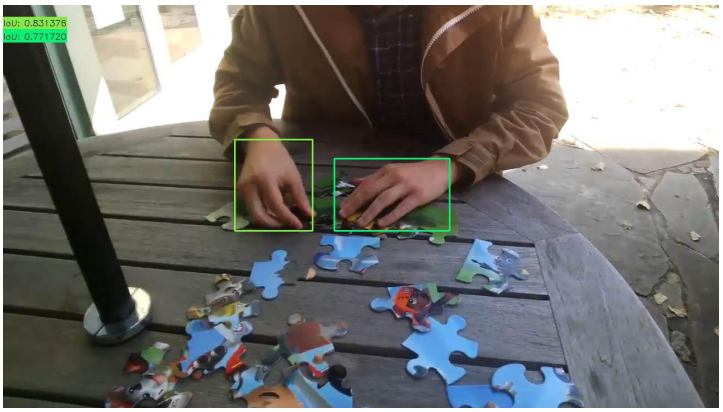
Results

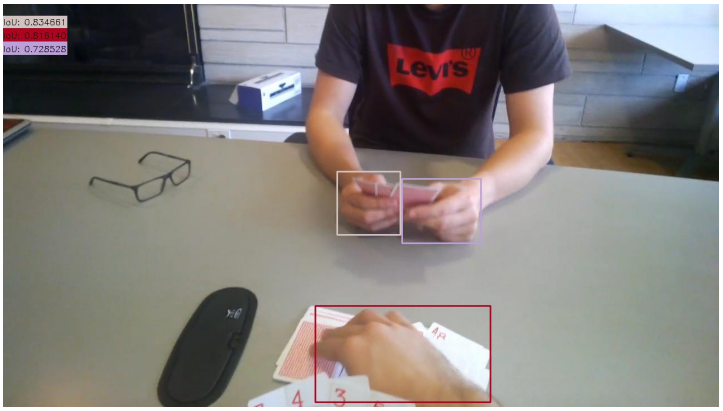
Detection

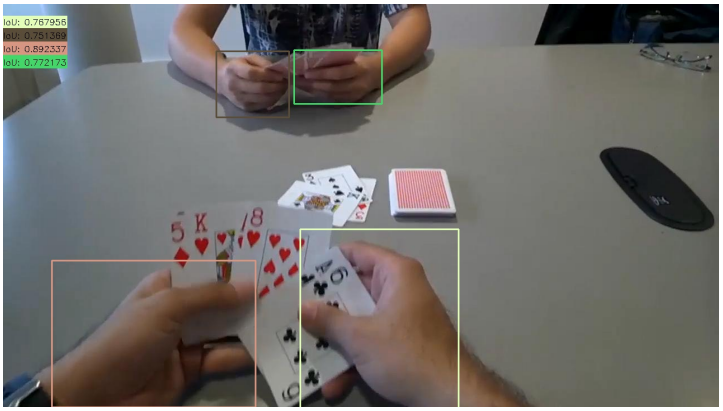




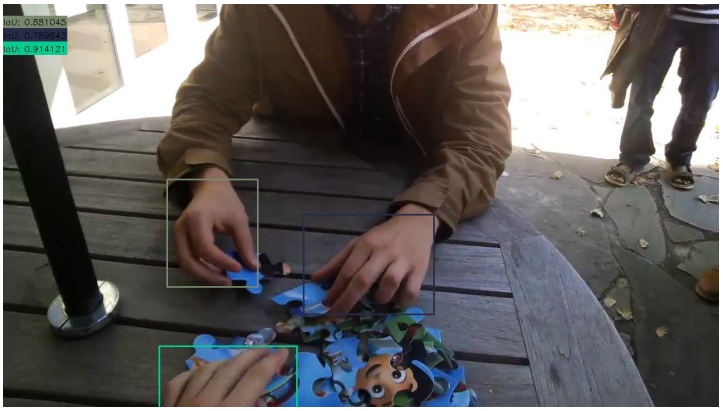





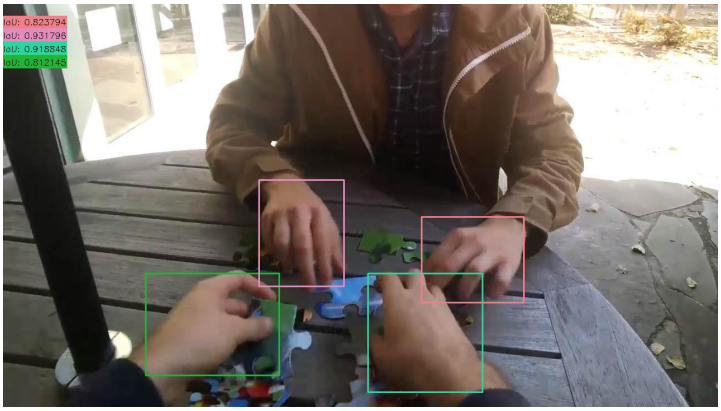

IoU: 0.687720
IoU: 0.778839

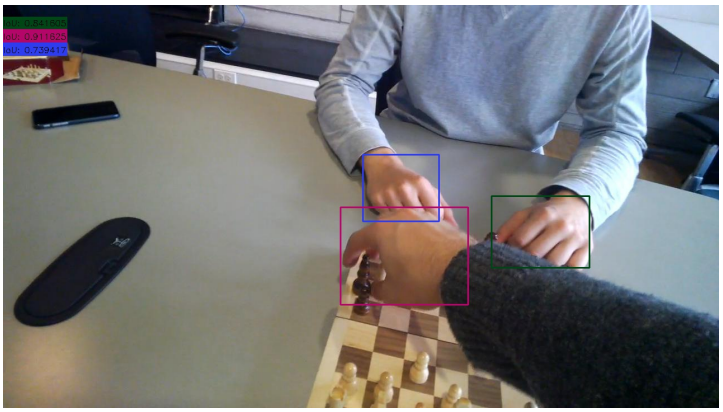

IoU: 0.831375
IoU: 0.771720

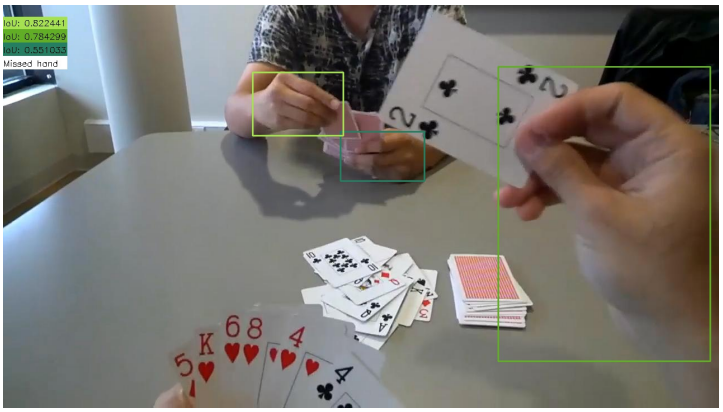

IoU: 0.834661
IoU: 0.728528


IoU: 0.767956
IoU: 0.892337
IoU: 0.772173


IoU: 0.881045
IoU: 0.914121

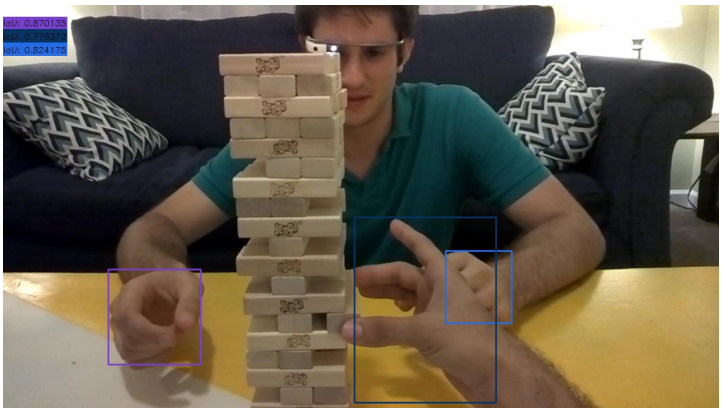

IoU: 0.823794
IoU: 0.931796
IoU: 0.915848
IoU: 0.812145

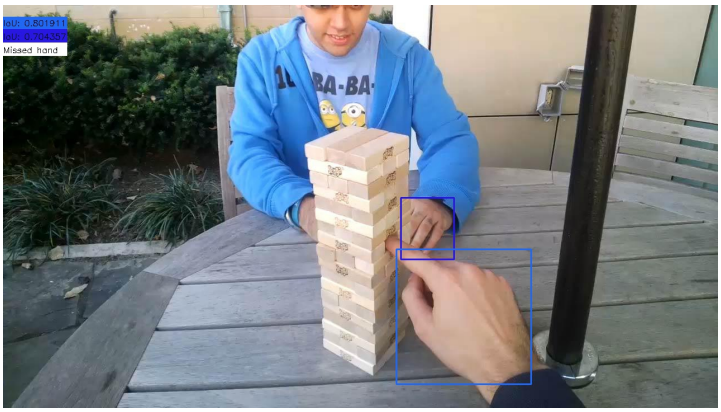

IoU: 0.911625


IoU: 0.822441
IoU: 0.784299
Missed hand


IoU: 0.857449
IoU: 0.702205
IoU: 0.901503


IoU: 0.942165
IoU: 0.785023
IoU: 0.852813
IoU: 0.498467

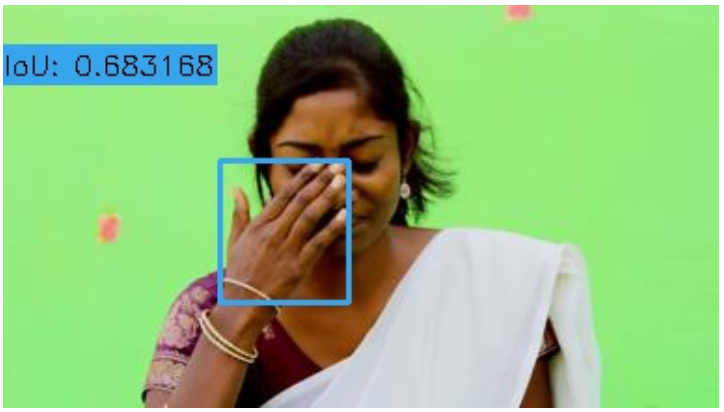



Missed hand


IoU: 0.753738


IoU: 0.710611
Missed hand

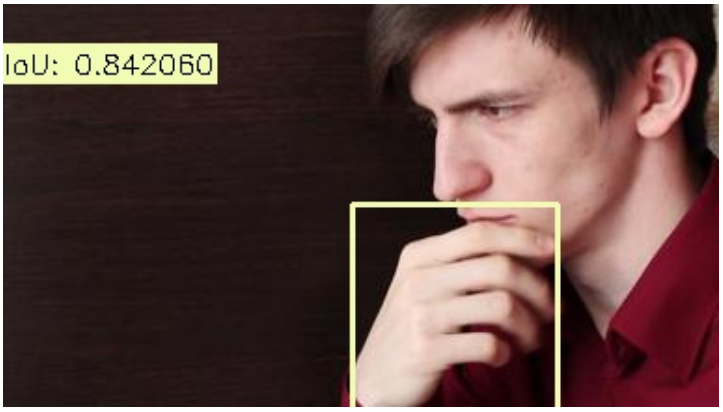

IoU: 0.628564

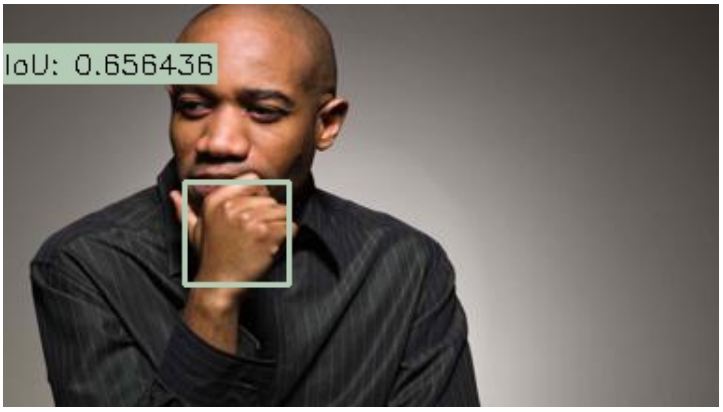

IoU: 0.683168


Missed hand
Missed hand


IoU: 0.850319


IoU: 0.756221
IoU: 0.463540

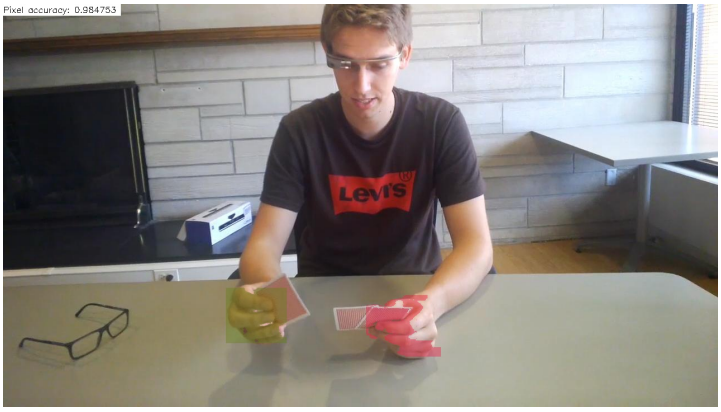

IoU: 0.842060



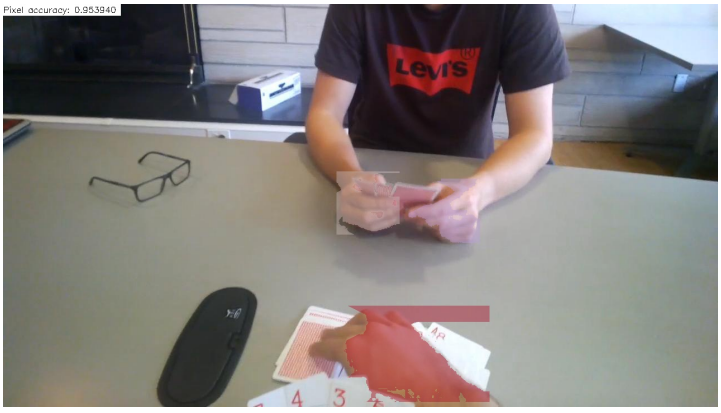
Segmentation


Pixel accuracy: 0.905264

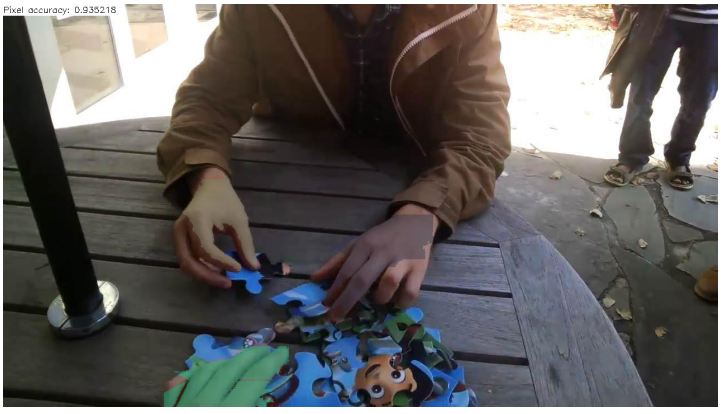

Pixel accuracy: 0.956598

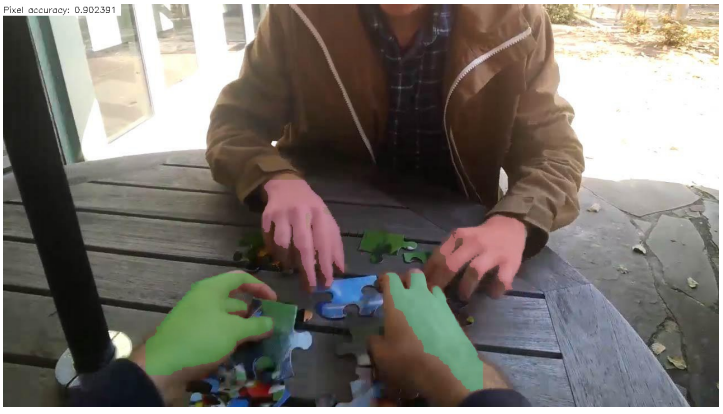

Pixel accuracy: 0.984753


Pixel accuracy: 0.968404


Pixel accuracy: 0.953940


Pixel accuracy: 0.876414


Pixel accuracy: 0.935218

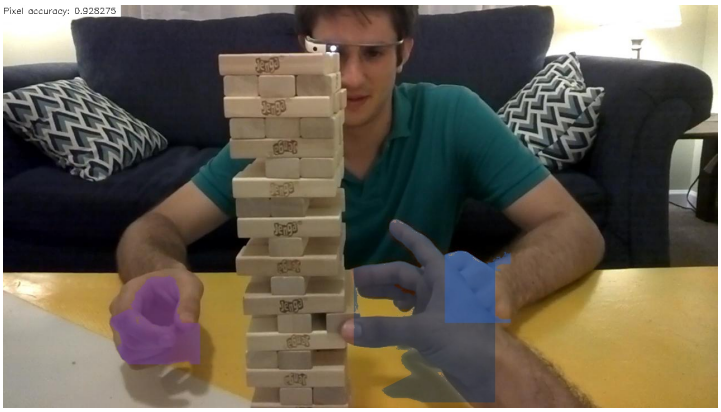

Pixel accuracy: 0.902391

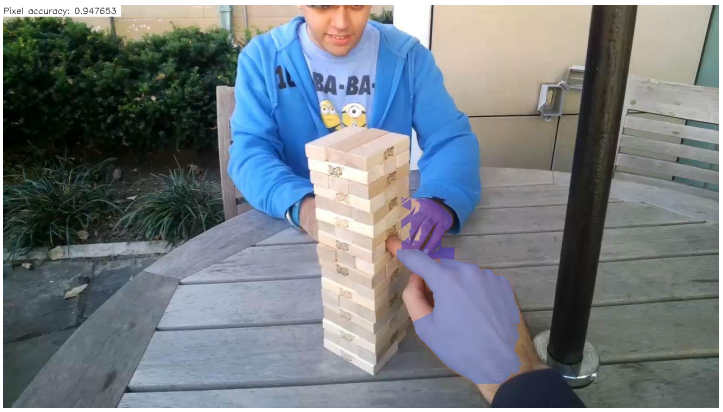

Pixel accuracy: 0.954390


Pixel accuracy: 0.847613


Pixel accuracy: 0.908575

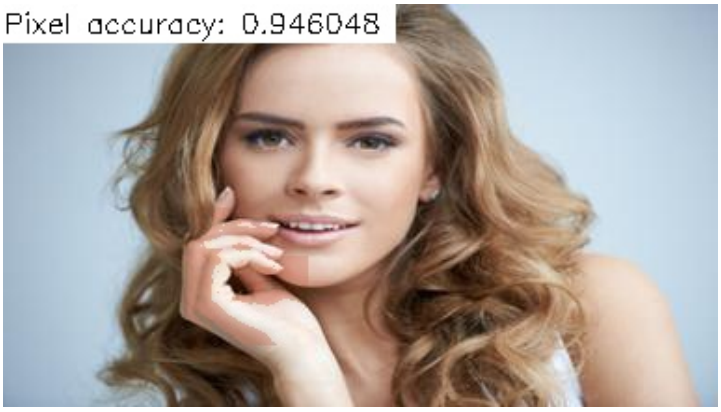

Pixel accuracy: 0.813388


Pixel accuracy: 0.928275


Pixel accuracy: 0.947653


Pixel accuracy: 0.942479

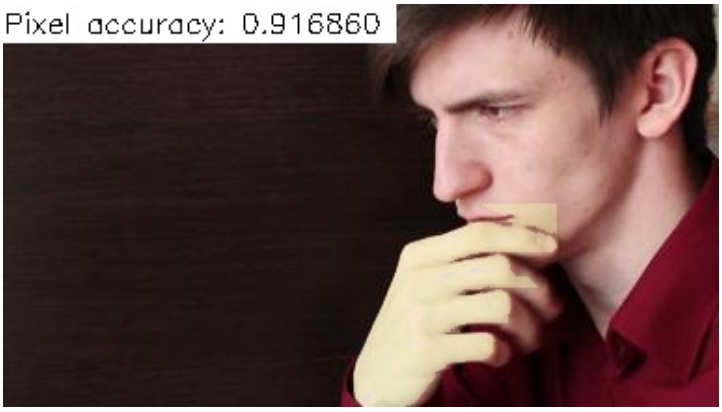

Pixel accuracy: 0.962047

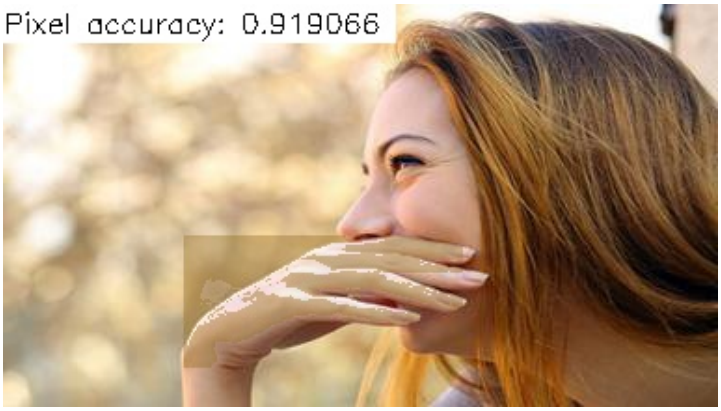

Pixel accuracy: 0.946048

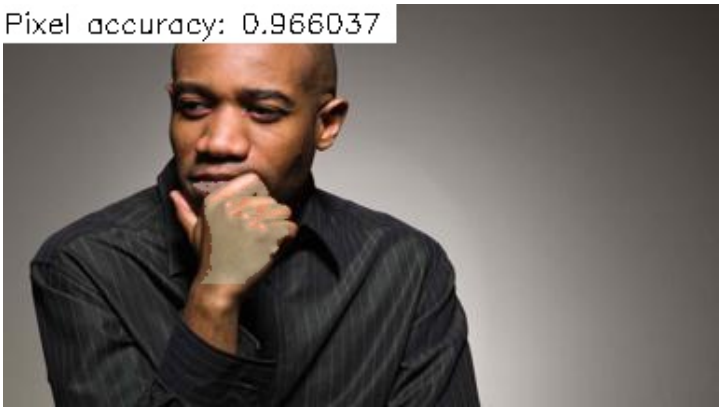

Pixel accuracy: 0.953173


Pixel accuracy: 0.866295


Pixel accuracy: 0.857277







Problems

Detection

We can notice that our detection model works very well, except for some pathological cases:

- overlapping hands: we tried to solve this problem by providing our dataset of a good number of images with overlapped hands. With different versions of the weights we were able to find at least one bounding box containing both the hands. We think that the probability of solving such a problem can increase by performing a longer training, but it remains a very difficult case to deal with.
- Tiny portion of hand: this is very difficult to be solved. The main reason is that such a portion of hand can be interpreted also as something else: background, desk, neck, other parts of the body. This means that to solve this problem we should add images with small portions of hands, but using them to train could mean increasing the number of detected bounding boxes, leading to some possible false positives.
- Black and white images: to solve such a problem, we added some black and white images to the dataset. After having done so, we were able to detect at least one of the two hands. We think that another reason why the other hand is not detected lies on the position of the hand. That hand, in fact, has a closed shape and is located in the left-bottom corner. We tried to add suitable images to recover from this issue, but it did not lead to any improvement. The fact is that the majority of the hands we have in our training set is near to the central portion of the image.

Segmentation

Please notice that some hands are segmented with colors that are similar to the color of the skin. This is due to the random selection of the colors for hand segmentation. The value of the pixel accuracy for sure gives us an information about the goodness of segmentation, but if you cannot distinguish it, please look at the predicted bounding boxes: the color of a segmented hand is the same as the color of the related bounding box. Moreover, if a bounding box is not found, a label "missed hand" appears in the output image of the previous section.

As we can see, the overall result of segmentation is quite good, but for sure not perfect. We already said that we experienced a great variety of other techniques, but the presented one has the best performances. We also tried at least three different region growing algorithms, each one with a different similarity criterion. Moreover, we tried with different algorithms for selecting the initial seed points and different values of the threshold, but, again, none of those solutions was comparable to the proposed one.
Just to list some of the tried alternatives to the actual region growing algorithm:

- do not consider if the neighbors are labelled as being part of the region, but take into account only the similarity with the seeds of the current point;
- update the mean reference values every time we add a new point in the seed mask;
- visit the points of the image in a different order: the current solution starts from the central seed and visits the matrix from the center to the beginning and to the end;
- use set of visited points instead of searching inside a mask;
- consider similarity to at least one of the starting seeds with a small threshold;
- consider similarity to all the initial seeds with a high threshold;
- use two thresholds: one for the central point and one for the neighbors.

The main problems with our solution for segmentation are:

- it is impossible to isolate a hand when it is in a background with a similar color. The most evident cases are those images containing hands over the face;
- in some cases the region growing algorithm includes too many points, while in others it includes too few of them. This is due to the high generality of situations from which the images are taken: we had to set a threshold value which is the most general possible. Those bounding boxes with too many points selected would have required a lower threshold value; vice versa for bounding boxes with too few selected points;
- sometimes we consider large portions of the bounding box not containing the hand. This mainly happens, in addition to what written above, when we select as initial seeds a great number of points not belonging to the hand. Unfortunately, this means that our algorithm for selecting the initial points can fail and, when it happens, being the reference value the mean of the seed pixels' values, the region growing enlarges a region which is not properly that one of the hand. The problem is that we do not have any a priori information regarding where the hand is in the bounding box, so we could not find any better method than extracting some points in the neighborhood of the center, since we suppose that the hand is fairly centered in the bounding box. We tried to solve this problem, with bad results, by exploiting the Canny algorithm for having an idea of where hand edges are.

Build the project

Once we loaded the directory containing all code in the virtual machine, we opened the terminal, entered the directory through it and typed the following commands:

```
mkdir build
cd build
cmake ..
make
./main
```

Note

All of us contributed to the report, according to the division of tasks explained at the beginning of the report itself. Then, of course, we all edited and revised the latter in its entirety, adding also more information in paragraphs written by other members.