Computer Vision - Summer Project

Human hands detection and segmentation

Marco Mustacchi, Nicola Rizzetto
Federico Violin

1 Introduction

Object detection and segmentation are computer vision techniques that work to identify and locate objects within an image or video, in our case human hands. Our goal was to implement a system that first detects any hand in an image, denoting them with bounding boxes, and then segmenting them from the background.

2 Dataset

For the training of the Neural Network we used the given two datasets, HandOverFace and EgoHands. The dataset structure was different throughout the project, for example during the MRCNN development it was just one folder with the whole images and another with the xml files containing the bounding boxes coordinates, while during the ResNet model development it was divided into hand images (squares containing only the hands) and no hand images (the image is divided into 224x224 blocks, where only the blocks that don't contain a hand get classified as no hand). In the end, besides the images from the final test set, some images from HandOverFace were removed, because the "hollow" background gave problems in the division of the image. For the extraction of the bounding boxes (or splitting into hand and no hand samples) we transformed the original files that contained the bounding box coordinates into xml files built like the following, with xmin, xmax, ymin, ymax that denote the corners of the bounding box:

```
▼<annotation>
   <folder>image</folder>
   <filename>190.jpg</filename>
   <path>...</path>
  ▼<source>
     <database>Unknown</database>
   </source>
  ▼<size>
     <width>852</width>
     <height>480</height>
     <depth>3</depth>
   </size>
   <segmented>0</segmented>
  ▼<object>
     <name>hand</name>
     <pose>Unspecified</pose>
     <truncated>0</truncated>
     <difficult>0</difficult>
    ▼<bndbox>
       <xmin>208</xmin>
       <ymin>243</ymin>
       <xmax>502</xmax>
       <ymax>466</ymax>
     </bndbox>
   </object>
</annotation>
```

Figure 1: Image 190.jpg of HandOverFace corresponding xml file

3 Detection

3.1 Discarded approaches

3.1.1 Viola-Jones cascade classifier

The first idea we had for the hand detection was using the Viola-Jones cascade classifier and Haar features. This method was not implemented in the end because there was a problem of compatibility with our version of OpenCV. This problem didn't permit to execute the commands `TrainCascade` and `CreateSamples`, the main commands of the method, and so we decided to discard it.

3.1.2 HOG features

After the problem with the implementation of Viola-Jones we tested the performance of HOG (Histogram of Oriented Gradients) algorithm, usually used together with a SVM to make the predictor. This method had two main benefits with respect to Viola-Jones, the first being the division of the image into blocks of the same dimension, where we have a reduction of time of analysis of the image; the second benefit is obtaining a better accuracy with a reduction of false positive with respect to the Viola-Jones algorithm. Since we were simultaneously working on HOG and Neural Networks approach, in the end we first got a working Neural Network, and so this method was discarded.

3.1.3 Mask Region-Based CNN

The first Deep Learning approach we tried was using a MRCNN, which is basically a Convolutional Neural Network that first does semantic and instance segmentation to the image, then divides the image with bounding boxes, so defining some "regions" and then processing them through the CNN. First of all, this method needed xml files built like explained in the Dataset chapter, so we converted the files from the original dataset ones into the xmls we needed. Our approach starts with some premade code imported from GitHub (repository in the Test_RCNN.ipynb code), then imports all images from the dataset and extracts the bounding boxes from the xml files described in the Dataset chapter. After this the code trains the model, and with the obtained weights it builds the inference model, a model suited for prediction of the labels in an image. Unfortunately, due to various problems regarding the imported code during the conversion from .h5 format to .pb format, this approach was discarded.

3.2 Final approach

3.2.1 Neural Network training

The final working approach for the detection was using ResNet50V2 as the main structure of our Neural Network. ResNet50V2 is a residual Neural Network, meaning that it can perform skip connection, skipping over one or more layers (it can also have another flow of information to the skipped layers), useful to reduce the vanishing gradients (very low change in the weights) and spread information to more layers. This approach uses the hand/no hand dataset, together with some preprocessing from `ImageDataGenerator()`. This method didn't necessarily need the transformed xml files, but since we already got them from the MRCNN method we still used them. After this we added some particular layers to the ResNet: an AveragePooling, to slim the Neural Network, and a double Dropout (with a certain probability ignore some values), to better avoid overfitting. We then imported the ResNet50V2 with pretrained weights from ImageNet, and made sure that the ResNet layers are not trainable, then started with 20 epochs (using only train and validation set). Since we implemented Early Stopping (if accuracy doesnt't increase much in some epochs, training stops), it only did 8 epochs. To evaluate the model we ran some predictions on the untouched test set, ending up with an accuracy of 0.725. Here follows a plot with loss and accuracy throughout the training (plot corresponding to another model that uses the same training but did 13 epochs before stopping):

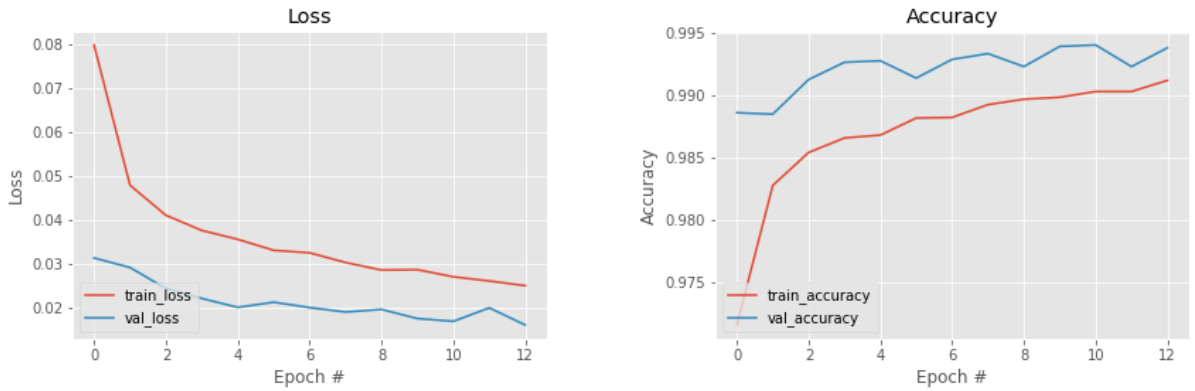


Figure 2: Evolution of the loss and accuracy

The obtained model from the training was converted from .h5 format to .pb, to adapt it for the C++ use.

3.2.2 Adaptation to C++ code

For the Detection in C++ we used a Sliding Window approach, with different window sizes. Initially, it was too slow, because it needed to scan the whole image, so we decided to add a skin detection first: through the `inRange()` method we obtained a mask of the image that only contained the skin regions (with a wide margin, to include all types of skins). To fasten it even more, we removed the outliers, in order to keep only the four white labels with the bigger area.

The values used for skin detection were obtained by starting with those at the following link, and then adding them in order to detect the skin in every possible image. In case this did not happen, we implemented a control based on the maximum area of the skin blob detected. If this was below a certain threshold, the sliding window is performed on the entire image.

At this point we ran the Sliding Window, that ran the detection model only if there was a white pixel (a skin piece) in the window, but it didn't work for grayscale images. So we first check if the image is grayscale, invert the image to remove the white spots and then apply an adaptive threshold (inRange couldn't be used with the values for grayscale images).

The model returns a confidence value based on the scanned region, with lower values meaning a higher probability that the region has a hand. Setting an appropriate threshold for the upper bound, we save the coordinates of the regions below it, but these coordinates do not correspond to the ones from the starting image ROI, since we cropped it. So we reverted back to the original image coordinates, and saved them as XYWH (X, Y, Width, Height).

In the end we applied a Non Maximum Suppression, to keep only the lowest confidence regions (so highest probability they are hands).

The detection method was finally improved by also taking into account the maximum area found as a result of skin detection. In fact, if the maximum area was sufficiently small, only the sliding window method with a smaller window was performed. This made it possible not only to remove any inaccurate detection, but also to speed up the execution of the algorithm.

The biggest problem encountered in detection was the large number of bounding boxes found with the sliding window approach. The final implementation is to eliminate all less likely bounding boxes through the `deleteRedundantBB()` function, keeping in mind that the maximum number of hands in a single image was 4. Following this, first the function `joinBoundingBoxesHorizontally()` joins me any bounding boxes that are joined and aligned horizontally. Finally, the `deleteAlignBB()` function removes any bounding boxes that are joined and aligned vertically, eliminating those with fewer probabilities.

4 Segmentation

For the Segmentation, the first step was extracting the ROI of the corresponding hand from the ground truth ROIs, and then using various computer vision techniques to accurately segment the hand with respect to the background.

4.1 Implemented methods

4.1.1 Otsu and inRange

The first method we implemented was a color based segmentation starting from Otsu algorithm with optimal threshold. We first tried to segment on BRG channels, then we switched to HSV and YCbCr channels.

The first main problem were the outliers, little regions of white pixels outside the hand region. To erase them we segmented the image in all three types of channels and multiplied the masks together, in order to limit the outliers. While it quite effectively solved the outliers problem, this method created holes inside the hand region, because if one of the three segmentations didn't work well it damaged the result of the other two. To counter this last problem we used the function `Floodfill()`, that fills the holes surrounded by only white pixels.

Later on we found another method to remove the outliers: the function `removeOutliers()`, that only keeps the major white region. Being the multiplication of the masks not necessary anymore, we only kept the HSV mask, the one that gave better results.

Regarding the `inRange()` method, initially the same values used in detection were used. Then the values were changed using those at the following link that provided better results.

This first approach worked well with the first 20 images, but had quite some problems in the last 10 images, because hand and faces usually merged.

4.1.2 K-means clustering

The first approach used to attempt instance segmentation was the K-means algorithm. The main idea was to check two bounding boxes overlapped, then give the K-means algorithm a 3 label set (the two hands and the background), otherwise just 2 labels (hand and background). Unfortunatelly, this method, which was still colour based, was discarded because it didn't improve much the segmentation.

4.1.3 Canny based segmentation

Another method was a segmentation based on edge detection, mainly using Canny. The main steps were the edge detection using Canny and then finding and drawing the hand contours using `findCountours()` and `drawContours()`, in order to color white the inside of the hand. This initial approach detected too much edges, and also had double edges feature, so we modified the edge detection in the following way:

1. Convert to grayscale;
2. Threshold (as mask);
3. Dilate the thresholded image;
4. Compute the absolute difference;
5. Invert its polarity as the edge image.

and we ended up with less and clearer edges.

4.1.4 Region growing

The region growing algorithm first starts with a check on the image to see if it's grayscale. Then we perform adaptive thresholding to the grayscaled image and add any edge detected to the original ROI. We then equalize the grayscaled image and run Canny algorithm, also drawing these edges into the original ROI. Now, to limit the seed regions, we perform a skin mask to the original image, with values found on these links, [1] [2] [3] or, if already grayscale, we chose the value 160. The function `seed_finder()` scans the gray-masked regions for compatible seeds, then performs erosion to limit them, and the `regionGrowing()` function merges all similar pixels. Here we also perform outliers removal and filling of the holes of the mask, because otherwise the similarity threshold would be too high and segment too much image.

4.2 Final setup

After various methods, we observed that every approach depended too much on the ROI it ran on. So we decided to keep the best performing methods (Otsu, inRange, Canny based and Region growing) and, based on their performance (pixel accuracy) on the current image, return only the segmentation of the best performing method. The best approach was initially chosen with the `chooseBestMask()` method, that evaluates the segmentation with respect to the ground truth. This method was then improved by considering the pixel accuracy of every single obtained ROI, named `chooseBestROI()`, and not the whole segmentation.

Having segmented all ROIs if an image, we then inserted them in a full black image, similar to the masks in the final test dataset, and then applied a different color to every segmented hand. The final mask was then applied to the original image, applying the colored segmentation to the hands.

[1] https://www.schemecolor.com/real-skin-tones-color-palette.php
[2] https://www.schemecolor.com/rich-skin.php (similar, not all colors)
[3] https://www.schemecolor.com/beautiful-peach-skin-tone-colors.php (similar, not all colors)

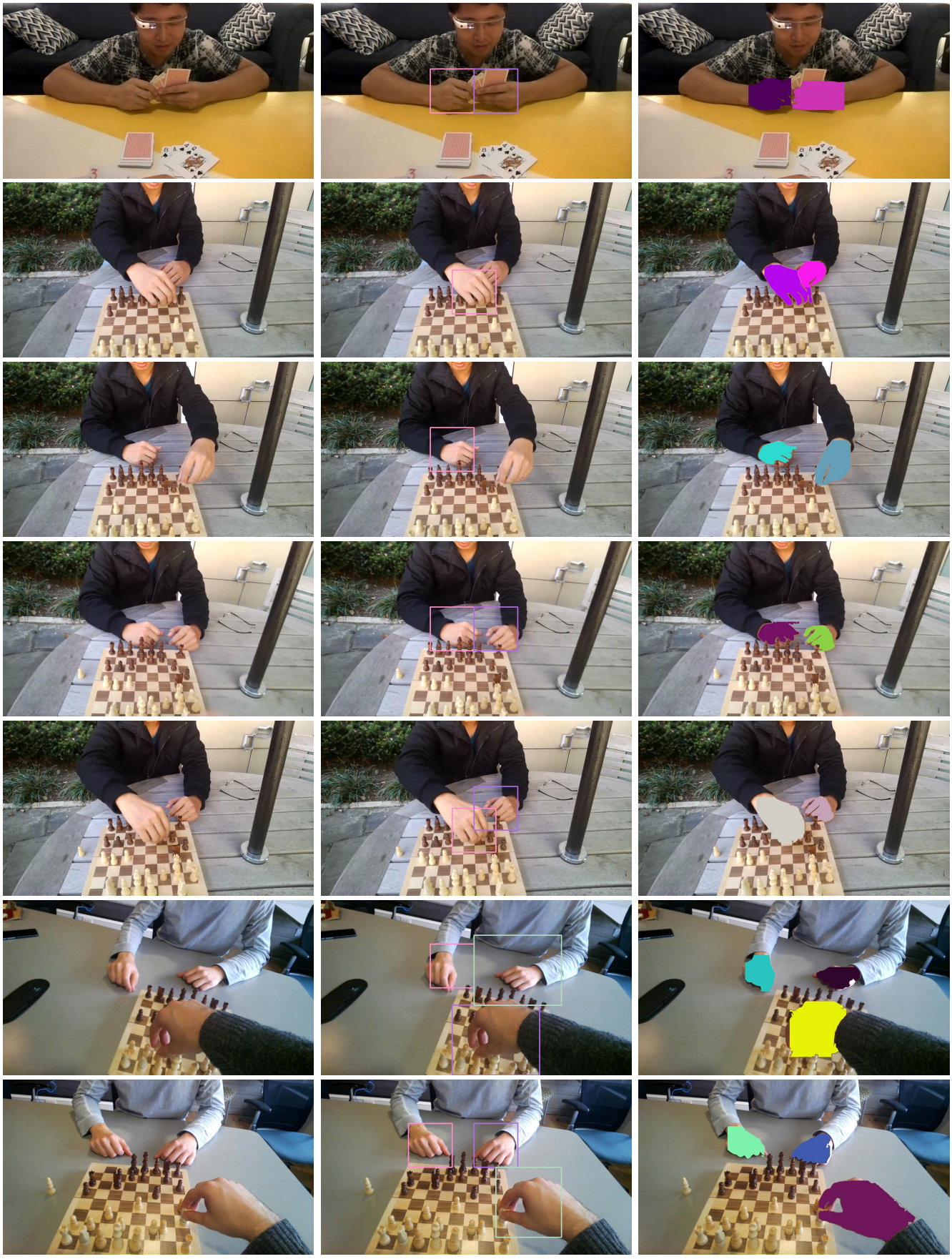
5 Performances

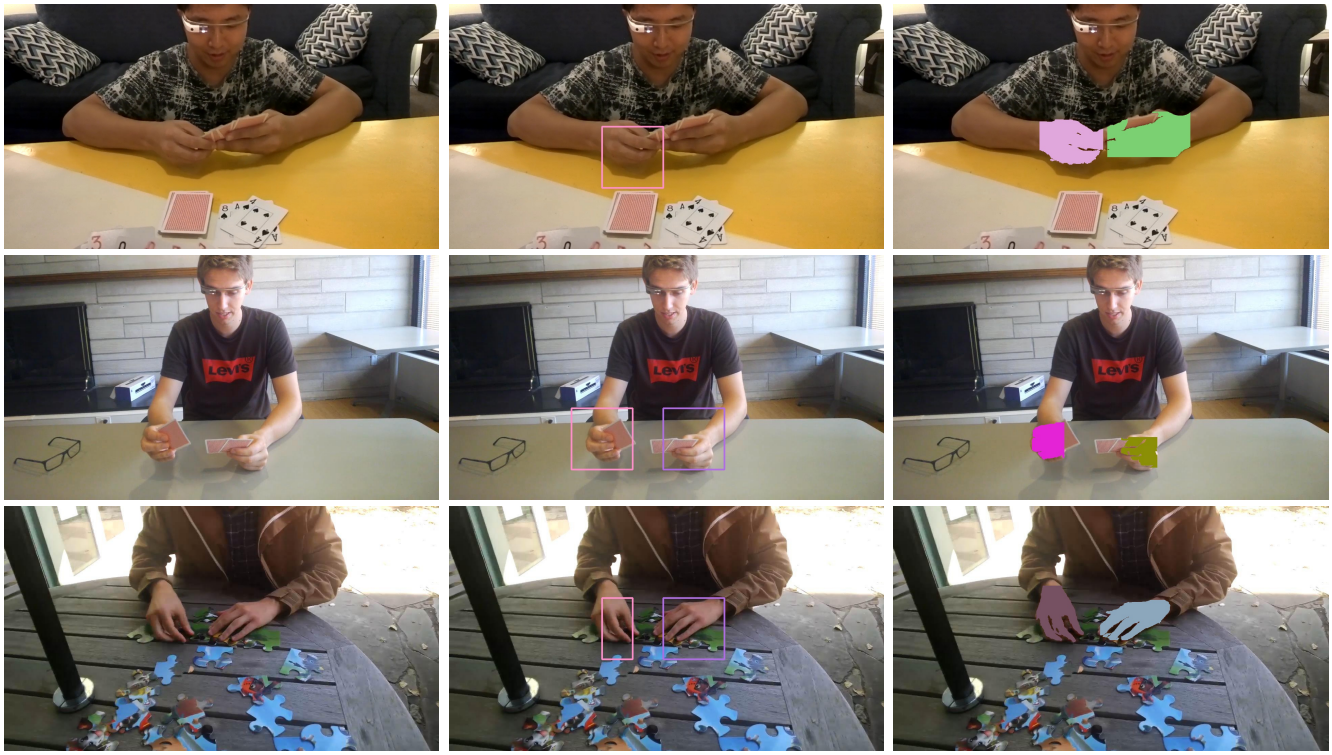
With regard to performance analysis, some difficulties were encountered in the Intersection over Union method. In fact, the problem in this case was to correctly associate the bounding box predicted by our model with that of the ground truth. Initially, the method was implemented by sorting the coordinates with respect to the first column (i.e. the position of x). However, this approach presented problems especially when the number of hands detected did not match the number of hands in the ground truth. Indeed, considering for example the detection result obtained in image 11.jpg (first image of Dataset 2), using this approach our predicted bounding box was compared with a different bounding box of the ground truth. The method was then changed by first doing a sort as before, and then associating for each predicted bounding box the corresponding ground truth bounding box that provided the greater IoU. In this way, going back to the example in image 11.jpg, the association was correct. The new method, however, makes a wrong association in image 20.jpg (the last of Dataset 2), as it associates the pink bounding box with that of the hand in the foreground and therefore the purple bounding box is no longer associated with it but with the one slightly above. We nevertheless decided to keep this solution because unless the predicted bounding box is wrong (as in image 20.jpg), the method works correctly.

Pixel Accuracy					
Image	**Accuracy**	**Image**	**Accuracy**	**Image**	**Accuracy**
01.jpg	0.991	11.jpg	0.987	21.jpg	0.963
02.jpg	0.996	12.jpg	0.938	22.jpg	0.984
03.jpg	0.995	13.jpg	0.982	23.jpg	0.982
04.jpg	0.993	14.jpg	0.978	24.jpg	0.981
05.jpg	0.993	15.jpg	0.987	25.jpg	0.948
06.jpg	0.991	16.jpg	0.955	26.jpg	0.968
07.jpg	0.991	17.jpg	0.982	27.jpg	0.965
08.jpg	0.990	18.jpg	0.928	28.jpg	0.989
09.jpg	0.993	19.jpg	0.965	29.jpg	0.975
10.jpg	0.993	20.jpg	0.994	30.jpg	0.991

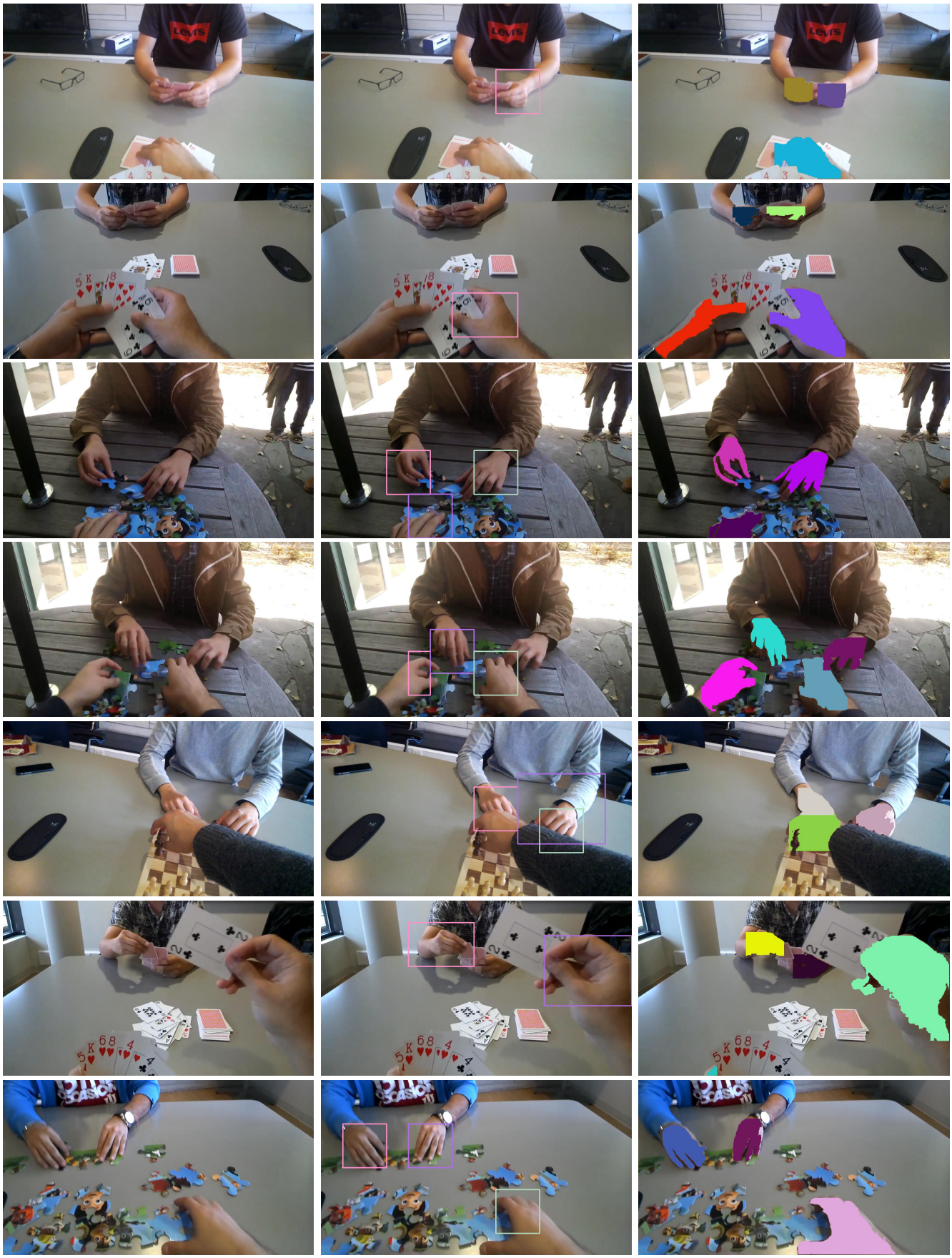
Intersection over Union					
Image	**IoU**	**Image**	**IoU**	**Image**	**IoU**
01.jpg	0.72 0.59	11.jpg	0.40	21.jpg	0.33
02.jpg	0.65	12.jpg	0.54	22.jpg	0.23 0.20
03.jpg	0.40	13.jpg	0.53 0.41 0.51	23.jpg	0.36
04.jpg	0.42 0.39	14.jpg	0.25 0.59 0.44	24.jpg	0.35
05.jpg	0.42 0.43	15.jpg	0.54 0.20 0	25.jpg	0.64
06.jpg	0.42 0.47 0.16	16.jpg	0.33 0.58	26.jpg	0.52
07.jpg	0.65 0.61 0.50	17.jpg	0.90 0.53 0.25	27.jpg	0.46
08.jpg	0.49	18.jpg	0.53 0.45 0.37	28.jpg	0.47
09.jpg	0.35 0.31	19.jpg	0.76 0.37	29.jpg	0.58
10.jpg	0.43 0.59	20.jpg	0.41 0.01	30.jpg	0.41

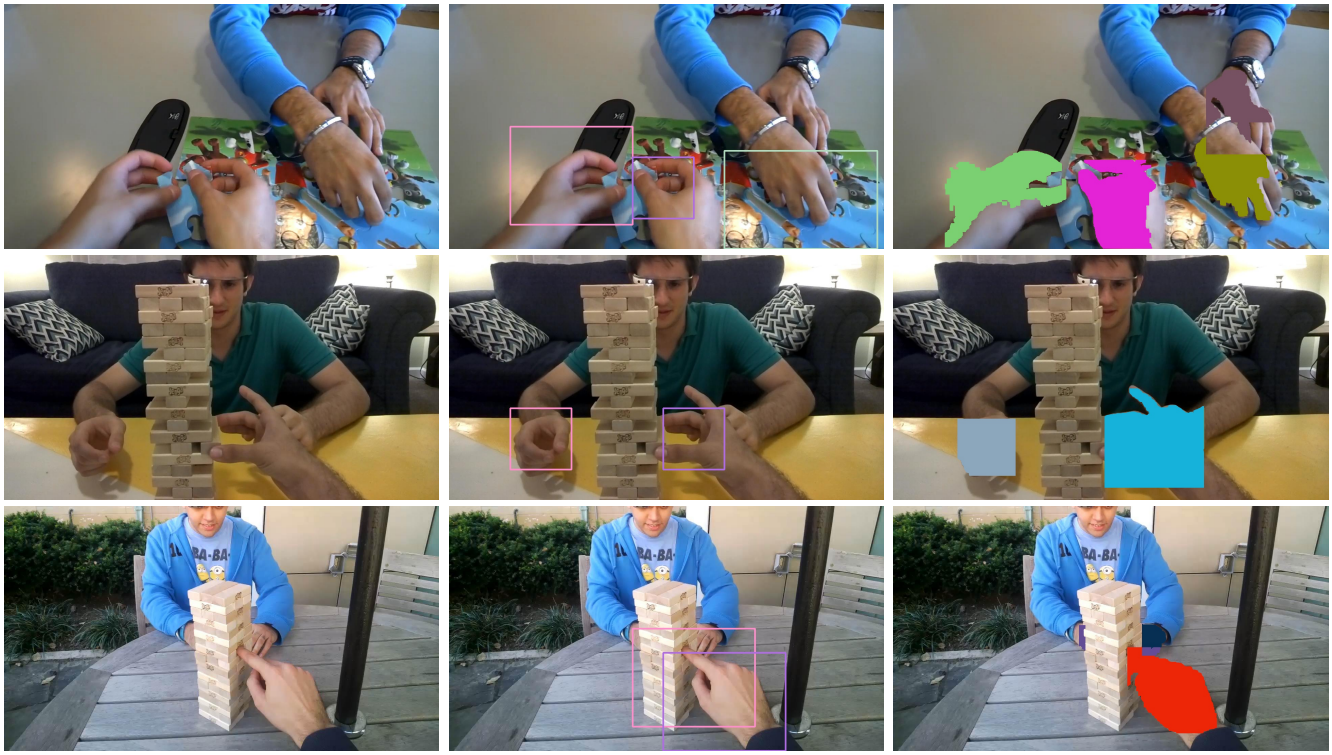
Dataset 1



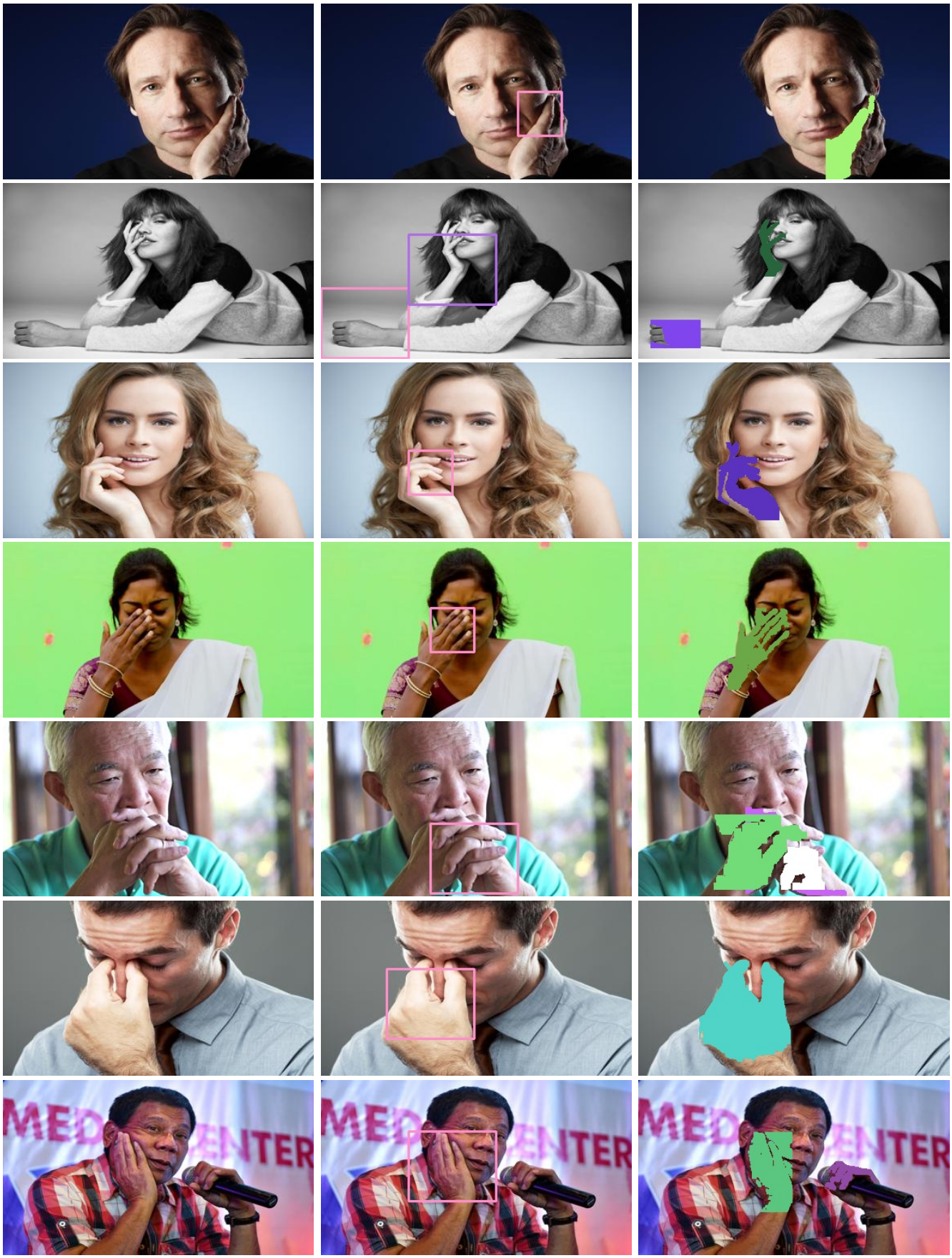


Dataset 2





Dataset 3



6 Contributions

Team member	Contribution
Marco Mustacchi	• */src/main.cpp* • */src/detection.cpp* • */src/segmentationMethods.cpp* • */src/iou.cpp* • */src/pixel_accuracy.cpp* • */lib/Detector.cpp* • */lib/Segmentator.cpp* • */lib/segmentationAlgorithm.cpp* • */lib/fillMaskHoles.cpp* • */lib/removeOutliers.cpp* • */lib/read_sort_BB_matrix.cpp* • */lib/write_to_file.cpp* • */CMakeLists.txt*
Total number of hours: 120h	
Nicola Rizzetto	• */Python_scripts/CNN_Resnet.ipynb* • */Python_scripts/Test_RCNN.ipynb* • */lib/segmentationAlgorithm.cpp (Region Growing algorithm)*
Total number of hours: 65h	
Federico Violin	• */xml_extractor/xml_extractor.cpp* • */Python_scripts/CNN_Resnet.ipynb (generator of positive and negative samples)* • */Python_scripts/Conversion.ipynb* • */Python_scripts/handoverface_bb.ipynb*
Total number of hours: 25h	