Applications of Computer Vision Techniques in an Art Gallery

Abdul Rehman Hamid
205275@studenti.unimore.it

Francesco Lugli
204600@studenti.unimore.it

Andrea Spinazzola
205582@studenti.unimore.it

1. Introduction

The main aim of this project was to find a good approach to recognize paintings hanging on a wall of an art gallery from an image or a video, to then being able to rectify them and finally to match each one to the paintings in a given database. The material we worked on is a set of videos taken from the Galleria Estense in Modena. The videos were taken with different lighting conditions and from different points of view, so it has been complex to find a good approach to detect all the paintings correctly. Other objects can also be present in the scene, such as furniture, statues and paintings' description labels near them. We experimented our approach in various videos with different environments, luminance conditions and shapes of paintings.

2. The approach

Since we started to work on the project, we decided to try two different approaches for paintings recognition and segmentation. The first approach was based on image processing techniques while the second one was based on deep learning and on CNNs in particular. In the end we compared the results obtained from the two methods and we acknowledged the CNNs approach to be superior. We also made a large use of CNNs to detect people, busts and to estimate the head pose of a person.

2.1 Paintings detection

As stated above we started working on the project using a more classical approach. After some image processing transformations to reduce the noise, for example Gaussian blur, we used adaptive thresholding and some morphological transformations to remove the background and distinguish the paintings. After these transformations we were able to identify the contours, but we had a lot of false positives, such as the paintings' description labels near them or some areas with a completely different lighting in respect to the scene.

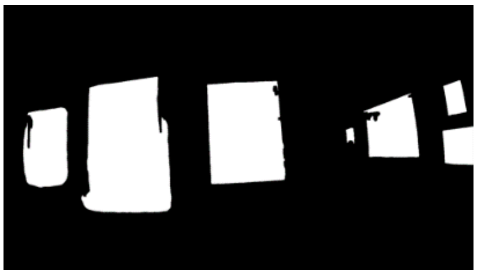

Fig 1. Image after background removal using Adaptive Thresolding

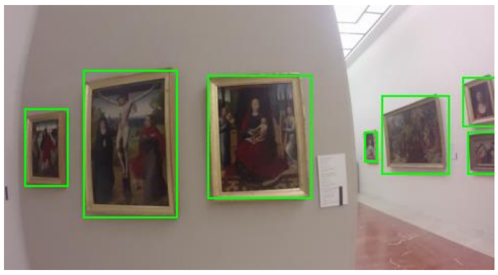

Fig 2. Original frame with ROIs

Seeing these drawbacks, we decided then to move to the world of deep learning by training and integrating Detectron2 [1] in our project. Detectron2 is the open-source object detection and segmentation system from Facebook AI Research. We mainly used the Mask R-CNN implementation of Detectron2 for segmentation since it is way faster than other implementations as shown in the benchmarks on the documentation website of Detectron2 [2]. The implementation of Mask R-CNN we used had a backbone of ResNet-50-FPN.

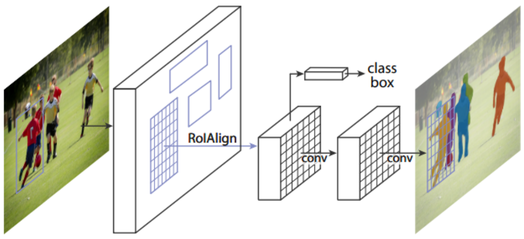

Fig 3. Mask R-CNN framework for instance segmentation

To do the training we started by building a dataset of 400 images containing paintings in various art galleries downloaded from Flickr. We annotated the paintings inside the frames using the tool Labelme. Once the dataset was ready, we trained the network for 50000 iterations splitting the dataset in 80% for training and

20% for validation and we obtained quite a good result for painting segmentation and detection as we can see in the figure below.

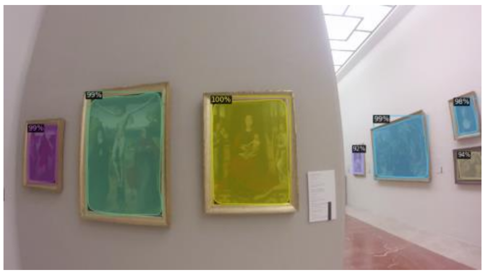

Fig 4. Example of painting segmentation using Detectron2

2.2 Painting rectification

2.2.1 Using image processing

To correct the perspective distortion of the paintings we decided to make use of the *warpPerspective()* function of the OpenCV library. This function takes as parameters the source image, the transformation matrix and the size of the destination image. This means that to have a successful perspective correction a good estimation of the transformation matrix is of paramount importance. OpenCV provides the *getPerspectiveTransform()* function, which allows us to estimate the transformation matrix using a set of four source points and four corresponding destination points which will be part of the rectified image. Hence, the main challenge of this part lies in finding the correct source points in the distorted image.

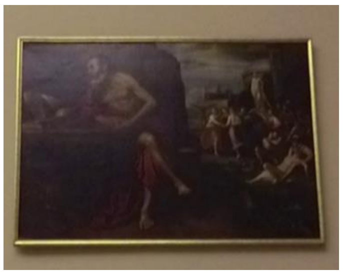

Figure 5. Original image

To find the correct source points used to estimate the transformation matrix we need to do a bunch of image processing steps. First of all, after the image has been converted to grayscale, a Gaussian filter is applied to reduce noise and facilitate edge recognition for the successive steps.

Following these first denoising steps, to detect the edges of the paintings and to separate the background from the paintings, it is necessary to employ a thresholding algorithm on the image. Through testing a fixed threshold (taken with or without Otsu's method [2]) was proven to be unsatisfactory, and thus, a method which employs adaptive thresholding was chosen. There are some drawbacks with the use of an adaptive threshold: shadows are often highlighted and frequently a number of isolated points can be found outside the frame, so it is of crucial importance to filter and exclude them from the contour of the frame. Also, to further reduce noise before corner detection, a dilation and an erosion are applied to the binary image.

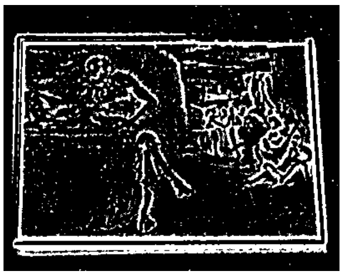

Fig 6. Image after adaptive thresholding

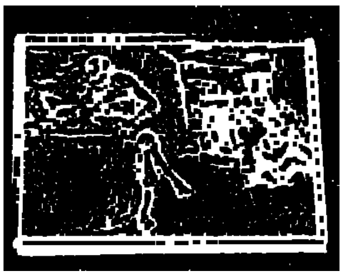

Fig 7. Result of dilation and erosion

After a suitable binary image of the painting has been obtained, the next step is to find the corners of the frame, which will become the source points for the *getPerspectiveTransform()* function. To do so we have chosen to find the contours of the image with the *findContours()* function of the OpenCV library. This function uses an implementation of Suzuki [3] algorithm to find all the contours of the image. However, in our case, to find the four corner points of the frame, we need to find the largest convex contour among them. To obtain a convex contour we have chosen to employ the *convexHull()* function, which is an implementation of Sklansky's [4] algorithm.

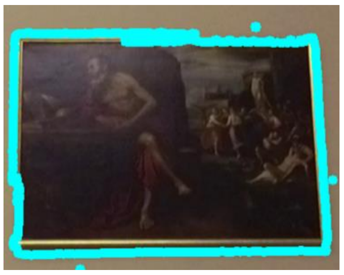

Fig 8. Detected contours

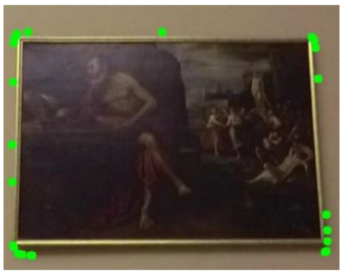

Fig 9. Result of convexHull()

After a convex hull has been found, with a second approximation we can find the best polygon that follows the shape of our hull. This can be done with an implementation of the Ramer - Douglas - Peucker algorithm [5] [6], which is used by OpenCV's *approxPolyDP()* function. This function returns the corner points of the approximate polygonal curve which should be the four corners of the painting's frame, provided of course that the contours have been identified as the frame's contours.

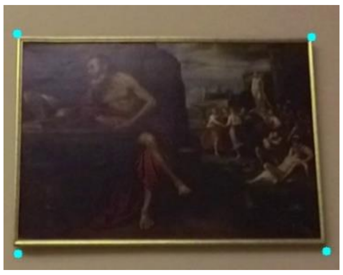

Fig 10. Identified source points

After the four source points have been found, the final step is to find the respective destination points so that the transformation matrix can be created. To do so first we sorted the points in clockwise direction starting from the top left, then we calculated the maximum width and height of the distorted painting's frame so that we can use them to maintain the correct proportions in the rectified painting. The new rectangle will start from the coordinates (0, 0) and will have as width and height the maximum width and height of the distorted frame. Our four destination points will be the corners of the found rectangle and the found width and height will also be used as parameters for the *getPerspectiveTransform()* function.

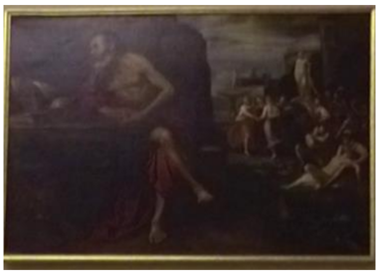

Fig 11. Rectified painting

This method for perspective correction takes in consideration only paintings where the frame is fully visible and fully contained into the image, since the difference in contrast and color of the frame makes it an easy target for the *findContours()* function. Another problem is the presence of shadows which are highlighted using adaptive thresholding and they can be wrongly detected as external contours.

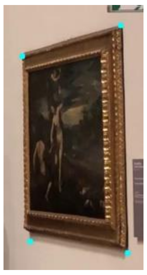

Fig 12. Example of erroneous corners detection due to presence of shadows

The main issue of this method although is that it only works as intended for rectangular paintings or at least paintings where the frame possesses a rectangular shape. Still, considering the problem at hand this method has its merits since the majority of the paintings possesses a rectangular frame.

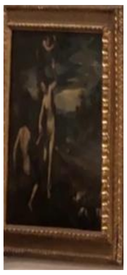

Fig 13. Example of wrongly fixed painting

2.2.2 Using Detectron2 segmentation masks

Since the image processing methods used for rectification and detection of paintings had their drawbacks, we decided to integrate Detectron2 into our project and use its segmentation masks to get more precise points for paintings rectification and for their detection. This helped us with the problem of shadows being detected as part of the paintings' frames or other cases where the paintings' contours included those of nearby paintings or the descriptions boards. An example of mask generated by Detectron2 can be seen in the figure below.

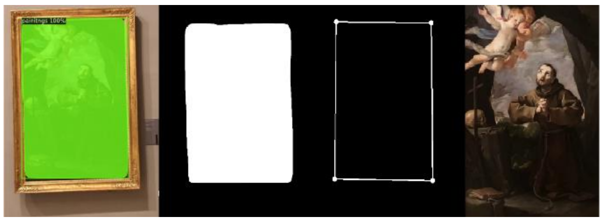

Fig 14. Rectification example of a rectangular shaped painting using the segmentation mask obtained by Detectron2

Once the mask is obtained we get the contours of the mask using the *findContours()* function. The number of points of the contour are then reduced using *approxPolyDP().* The next steps are done differently if a rectangular contour or a non-rectangular shaped contour is found. If the approximated contour is a rectangle then the four corners of this polygon are used for painting rectification. If the contour is detected to not to be a rectangle then, first of all, we compute the rotated rectangle of the minimum area enclosing the contour using the function *minAreaRect()*. Afterward we search for upper, bottom, left and right midpoints of the sides of this rotated rectangle enclosing the contour. In the next step we check if these midpoints are on the contour using the function *pointPolygonTest()* and if they are not on the contour we move them accordingly, so they intersect the contour. An example of the result can be seen in the figure below.

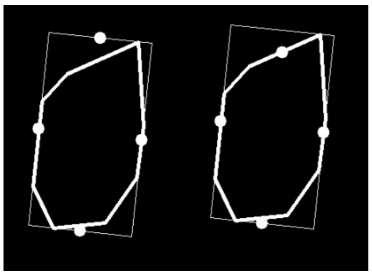

Fig 15. Example of detected midpoints[left] and after they are moved on the contour[right]

Once the midpoints or the corner points of a rectangular contour are ready we build the perspective matrix using *getPerspectiveTransform()* function. The rectified image is then built using the *warpPerspective()* function. If we are rectifying using the midpoints, then the height and the width of the rectified image are respectively the Euclidian distance between the upper and the bottom midpoints and the distance between the left and right midpoints. In case of a rectangular shape the height and the width of the rectified image are respectively the Euclidian distance between the upper left corner point and the bottom left corner point and the distance between the upper left corner point and the upper right corner point. Obviously using the midpoints for non-rectangular shaped paintings is not a good solution for their rectification. Acknowledging this problem, we decided to integrate the usage of vanishing points to detect better points to use for non-rectangular shaped paintings. To do so we utilized an implementation of the Vanishing Point algorithm by Xiaohu Lu et al. [7] called lu_vp_detect [8]. The function *find_vps()* of this library does the vanishing points detection while *create_debug_VP_image()* of this library draws lines of a certain length on an image and clusters them based on which vanishing point they contribute to. For example, if a line contributes to the right vanishing point then its color is red, if it contributes to the left vanishing point then its color is green and blue is used for lines contributing to the vertical vanishing point. An example of the output of this function can be seen in the figure below.

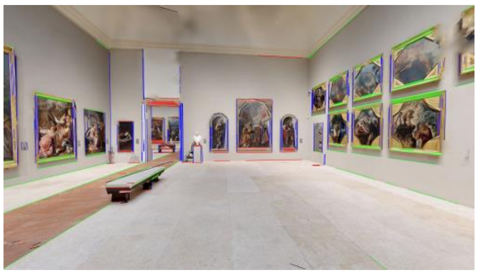

Fig 16. Example of lines drawn by create_debug_VP_image() function of lu_vp_detect library

Once the image with lines oriented towards the vanishing points is obtained, an image processing approach is used to determine to which vanishing point the upper midpoint and lower midpoint of a painting's contour should be connected to. For doing this we first detected the green and red lines in the image then we computed the distance between the upper midpoint and the red lines and the distance between the upper midpoint and the green lines. If the minimum distance between the upper midpoint and the green lines is lower than the minimum distance between the upper midpoint and the red lines, then we assume that the upper midpoint should be connected to the left vanishing point otherwise it should be connected to the right vanishing point. The same procedure is done for the bottom midpoint. For the left and right midpoints on the contour this is not necessary since they should be connected to the vertical vanishing point, but this is true only if we assume that the paintings on the walls are not tilted. After we determined to which vanishing point the upper and bottom midpoints should be connected to, we consider the lines connecting these midpoints to the corresponding vanishing points. The intersection between these lines gives us four corner points to better rectify the painting.

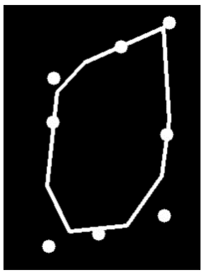

Fig 17. Example of painting with midpoints on the painting's contour and the four points discovered using the vanishing points method

To use the just described method we assumed that our input video frames or images respect the Manhattan World assumption which states that the imaged scene should contain three orthogonal, dominant directions, and this is often satisfied by outdoor or indoor views of man-made structures and environments. We have seen that when the lines oriented toward the vanishing points are correctly drawn and the vanishing points are correctly detected this method works better than using only the midpoints to rectify the image, but this is not true if the input video frame or image is a close up of just a wall where paintings are hanging. So, this method was mainly applied in the rectification of images from the 3D model where the scenes seems to respect the Manhattan World assumption. A better way to determine if the input images respect the Manhattan World assumption is needed and a starting point can be the creation of a null hypothesis model like it is stated in the James Coughlan and Alan Yuille's paper [9].

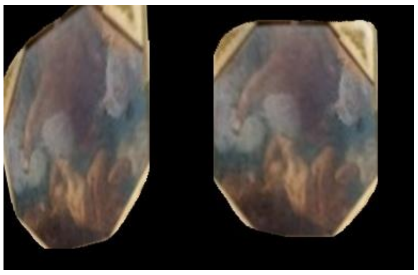

Fig 18. Example of a painting wrongly fixed using only midpoints[left] and the same painting correctly rectified using points found employing the vanishing points method [right]

2.3 Paintings retrieval

For what concerns the retrieval of paintings from the given paintings database, we mainly implemented three methods of which we selected the best one after a benchmark in terms of the speed taken by each method and its accuracy. The best method resulted to be ORB in combination with a brute force matcher and using the KNNMatch. Each rectified painting found in the video's frame was matched against each image of the given paintings database. A dictionary having as key the title of the matched image and as value the number of good matches was created. This dictionary was then sorted in descending order by the keys. The images of the given paintings database were loaded into memory to speed up the comparison.

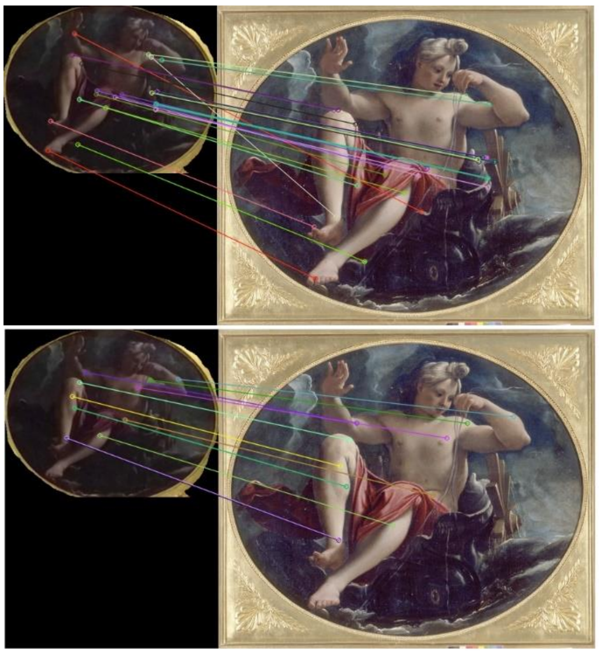

Fig 19. Example of a painting with its best match in the database and its good matches found using ORB[above] or AKAZE[below] in combination with a brute force matcher and KnnMatch

2.4 People detection

For people detection we decided to use YOLOv5 [10], the last version of the famous fast compact object detection model. We first tried to detect people using the pretrained weights based on the 80 classes of the COCO dataset and showing only the detected ROIs with the label "person". However, we found out that we had a lot of false negatives, for example the statues/busts and the people drawn in the paintings. Hence, we decided to avoid this problem by building up our own dataset with two classes (people and statues) and training YOLOv5 from it. We started by collecting 2000 pre-labeled images of people or statutes/busts for each class from Google Open Images [11]. When the dataset was ready, we trained YOLOv5 on Colab using the yolov5s model. We took advantage of the transfer learning technique by starting the training from pretrained weights. At the end of the training we obtained a good detection result for statues/busts but we had a lot of false negatives in people detection. Hence, we decided to improve our dataset of people's images and, to do so, we downloaded from Flickr 1500 other images containing persons walking in museums and people in art galleries. We labeled the bounding boxes in these images manually using the tool *Yolo_label*. Finally, we retrained YOLOv5 using this dataset and we had a good result in detection of both people and statues/busts. The previous problem of people drawn in paintings being detected as real people was resolved. To train YOLOv5 the images were resized to 460 x 460, we used a batch size of 64 and we trained the network for 500 epochs.

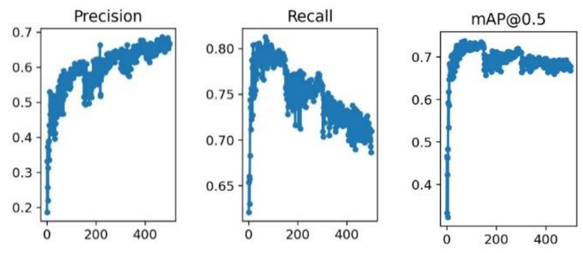

Fig 20. Precision, recall and mAP during the training

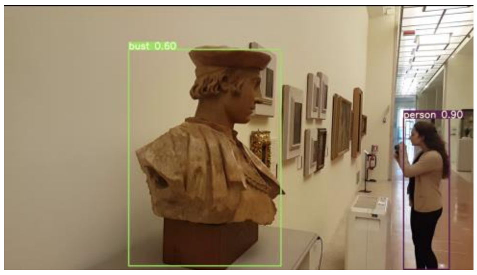

Fig 21. Example of detection of people and busts

2.5 People localization

First of all, we processed the map image provided using image processing methods to find the coordinates of each room on the map. This is achieved by firstly thresholding the map image using the *threshold()* method of OpenCV, where a combination of THRESH_OTSU and THRESH_BINARY flags are employed obtaining a Otsu's binary image. Afterward the function *morphologyEx()* is used with a kernel of (15,3) to detect horizontal lines in the binary map image and then the same function is employed with a kernel of (3,14) to detect vertical lines. Following, the contours of the horizontal and vertical lines are detected and these lines are then removed from the map image using the *drawContours()* method of OpenCV. Once the vertical and horizontal lines are removed we searched for the contours of the digits on the map and we created a ROI for each digit using the method *boundingRect()*. Once the single digits are localized, we searched for double digits by simply checking if a single digit has another digit near it. Once the double digits are found we simply used pytesseract's *image_to_string()* method to recognize the digits. Once each room number is detected and recognized we save their ROI's location to be able to quickly find them later.

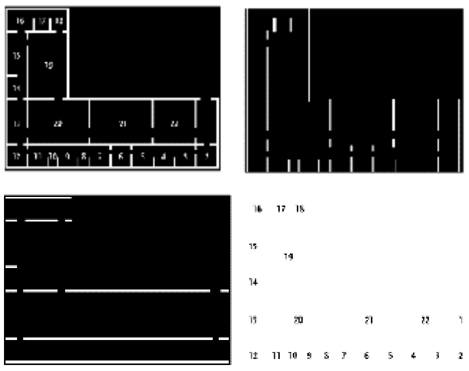

Fig 22. Image representing the initial process of rooms' numbers detection and recognition

Instead, to localize in which room a person is, we used the information from the image retrieval process. Once each painting in the room is detected, rectified and a list of matched paintings, sorted by their score, from the paintings database is retrieved, we proceed to find the room where these paintings might be located. To find the correct room we simply count how many times a room appears to be the one of the best matching painting for each painting. The room with the highest number of occurrences is assumed to be the one in which the paintings can be. Then, for each person in the image, we simply drew a little colored circle on the map on the location of the previously discovered room number for better visualization.

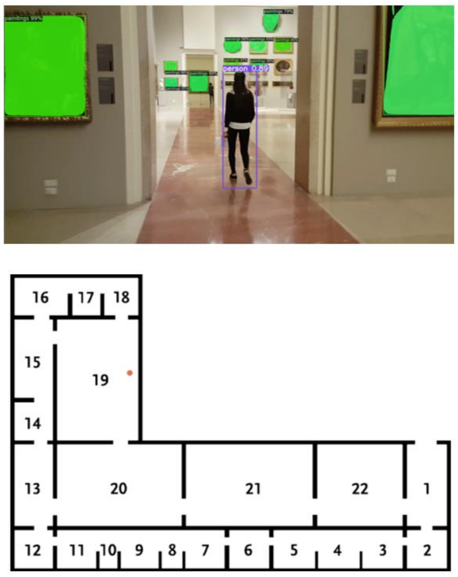

Fig 23. Example of person localization

2.6 Pose estimation

After detecting people, we also tried to estimate which paintings they were looking at. The focus of attention of a person can be approximately estimated by finding the head orientation and this is particularly useful when the eyes are covered or when the user is too far from the camera to grab the eye region with a good resolution. To estimate the head orientation we relied on Deepgaze [12] created by Massimiliano Patacchiola. The aim of this project is to use CNNs to estimate the head pose of a person. In particular it exploits the Viola-Jones object detection framework as a face detector and two CNNs, trained on the AFLW dataset, as head pose estimators which detect the pitch and yawl angles.

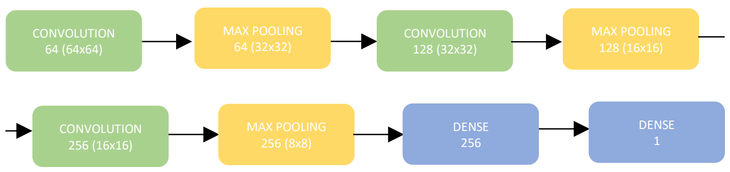

Fig 24. Architecture of CNNs used in Deepgaze

We used the Haar Cascade algorithm implemented in OpenCV to detect the ROIs of the faces, on the people found previously using YOLOv5. If the face is not discovered, we considered that the person is seen from behind. Thus, we implemented the Deepgaze network in our code and we used it on each face to obtain the angles. In particular we used the yaw angle to get the direction the head is pointing to and to understand if the person is looking left, right or if he is looking at the camera. Deepgaze network returns the yaw angle in a range of [-100°, 100°], where an angle of 0° means that the person is looking at the camera. The ROI of the face to be passed to the network must have a dimension larger than 64 x 64. Thus, after acquiring the paintings' ROIs from Detectron2, we determined for each one if it is located in the direction the person is looking at.

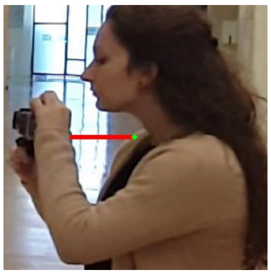

Fig 25. Example of a pose estimation

2.7 Replacing paintings in the 3D model

Once the process of detection, rectification and retrieval of the paintings is finalized then the process of replacing the paintings in the 3D model is quite simple. Each painting in the model is detected, rectified and its best match is loaded from the database using the image retrieval method. The painting is then replaced in the 3D model using the same points used for its rectification. An example of the final result in one of the images obtained from the 3D model can be seen in the figure below.

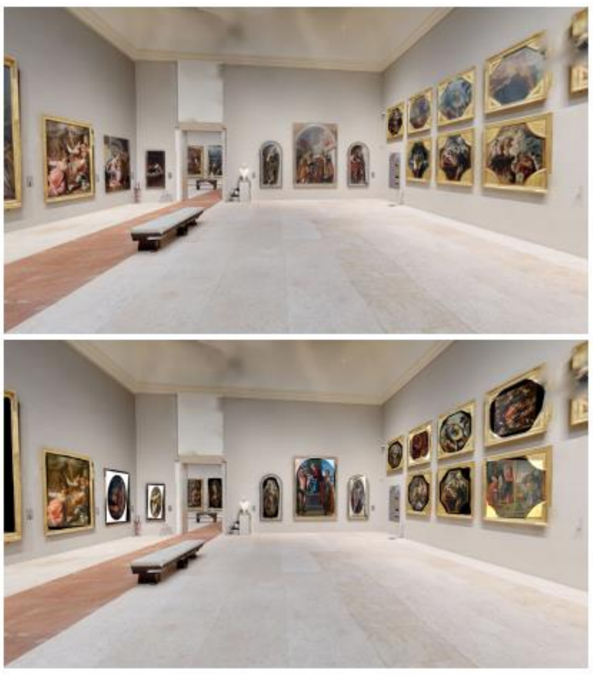

Fig 26. Example of an image of the 3D model with original paintings(above) and the replaced paintings(below)

Another possible, but yet to be tested, approach to replace the paintings in the 3D model images could have been to use feature matching to find a query image, taken from the given paintings database, in the 3D model images. Once the query image is located in the 3D model image we can use the points discovered in both the images to create the transformation matrix using the function *findHomography()* in combination with RANSAC or LEAST_MEDIAN to remove the bad matches which provides bad estimation, or otherwise called the outliers.

3. Results

To evaluate the paintings segmentation method, we used as metric the average precision (AP) and the average recall (AR) over a determined IoU threshold. To build the test dataset we made a script to save 50 random frames from all the videos as JPEG images and we annotated manually over 200 contours of paintings on them. AP can then be defined as the area under the interpolated precision-recall curve, which can be calculated using the following formula:

$$\mathrm{AP} = \sum_{i=1}^{n-1} (r_{i+1} - r_i) P_{interp}(r_{i+1})$$

where $r_1, r_2, \dots, r_n$ are the recall levels (in an ascending order) at which the precision is first interpolated. Instead AR is the recall averaged over all IoU $\in$ [0.5, 1.0] and can be computed as two times the area under the recall-IoU curve:

$$AR = 2\int_{0.5}^{1} recall(x)dx$$

Where x is IoU and recall(x) is the corresponding recall. The Intersection over Union (IoU), also known as the Jaccard index, measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks.

$$IoU = \frac{target \cap prediction}{target \cup preditcion}$$

For example, if we define an IoU threshold of 0.5 all the detected paintings with an IoU over 0.5 are considered true positives.

The results we obtained testing our trained weights using Detectron2 at different IoU thresholds are the followings:

	IoU=0.50:0.05:0.95	IoU=0.5	IoU=0.75
mAP	0.769	0.946	0.885
mAR	0.219	0.785	0.882

To evaluate the YOLOv5's weights for people and statues detection we used the same metrics in addition to the precision (P) and the recall (R) which are defined as:

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

We tested the network on 30 images containing 15 people and 14 statues instances. The results we obtained are the followings:

P	R	mAP IoU=0.5	mAP IoU=0.50:0.05:0.95
0.679	0.808	0.743	0.512

Unfortunately, we could not test the network on more images due to a lack of videos containing people.

Considering the numerical and the qualitative results we obtained, we think that our painting detection and segmentation method work quite well. Almost all the medium and big size paintings are detected and segmented precisely, while we obtain some false negatives with the smaller ones. For people detection the results could have been improved by enlarging or augmenting the dataset before training.

Following we can see the results of the comparison we made between different methods for painting retrieval.

Method	Time(s)	Accuracy	Accuracy/Time
Resnet18	39,909961	0,27027	0,006771994
ResNet50	96,066402	0,378378	0,003938713
ORB-BFM-KNN	45	0,864865	0,019260705
ORB-BFM	47	0,594595	0,012623071
ORB-FLANNM-KNN	48,033017	0,837838	0,01744296
ORB-FLANNM	48,061893	0,027027	0,000562337
AKAZE-BFM-KNN	243,623476	0,864865	0,003550007
AKAZE-BFM	246,296408	0,378378	0,001536271
AKAZE-FLANNM-KNN	254,998496	0,810811	0,00317967
AKAZE-FLANNM	255,861801	0	0

TimeIt was used for timing each method's function execution time. The comparison was done on a test dataset of 37 images of rectified paintings extracted from different videos. Each image of the test dataset was compared using every retrieval method against each image taken from the given painting's dataset. Every image was loaded into memory and each retrieval method's function did only the basic operations of comparison to avoid affecting the benchmark's results. The parameters used for KnnMatch were K=2 and the match ratio for Lowe's ratio test was 0.75. Crosscheck was used in case of the normal matcher. In case of FLANN the parameters of the matcher were the ones suggested on the OpenCV documentation's website [13]. Hamming distance was used for the brute force matcher.

The first method was feature extraction from the last hidden layer of ResNet18 and ResNet50 models, pretrained on ImageNet. The approach of ResNet18 and ResNet50 resulted a complete failure both in terms of time (at least using a CPU) and in terms of accuracy. The cause of a very low accuracy in this case might be the huge dissimilarity between the images in ImageNet and the paintings' images we had. It would be nice to test the same approach on a ResNet18 or Resnet50 model trained on paintings' images, but due time concerns this test was not possible. The second tested method was ORB in combination with a brute force matcher or FLANN matcher with KNNMatch or the normal match with crosscheck. The combination of ORB plus brute force matcher using the KNNMatch resulted to be the best one. The FLANN matcher, despite being reported to be faster, resulted to be the slowest, at least with the parameters suggested on the OpenCV documentation's website. The third method was AKAZE in combination with a brute force matcher or FLANN matcher with KNNMatch or the normal match with crosscheck. Even in this case the combination of AKAZE plus brute force matcher using the KNNMatch gave the best result at least in terms of accuracy. In the end it is quite obvious why the brute force method works better in contrast to FLANN matcher in terms of accuracy. But most of all these comparisons prove that KNNMatch is superior to the normal matching method and the reason probably is due to the increase in the number of matches that directly boosts the probability to find a good match. In fact, instead of having only one match being considered good based on the Euclidean distance between the feature points, if we consider two or more matches and we check the goodness of the first match based on the distance between its neighbors, we increase the probability of the first match to be good.

4. Discussion

In this work we analyzed and described different types of methods mainly employed for paintings extraction and detection. With our first approach using image processing we have seen that a good generalization cannot be achieved. Paintings detection itself was satisfactory although there was a significant presence of false positives. With these drawbacks in mind we decided to try to employ a deep learning approach which gave us way better results. Even if the results were great some types of very distorted paintings were still undetected so an augmentation of images of the training dataset would have helped in these cases.

Regarding the rectification task we have seen that after using different methods none of them generalized well for non-rectangular shapes.

For what concerns the retrieval methods we have seen that ORB and AKAZE give very promising results even if they can be very slow when the comparison is done over a very large dataset of images.

Overall, considering the numerical and qualitative results, we can conclude that the employed methods are acceptable even if with more time and research better outcomes could have been achieved.

A working demo of segmentation and detection can be viewed at this link.

5. References

[1] [Online]. Available: https://github.com/facebookresearch/detectron2.

[2] [Online]. Available: https://detectron2.readthedocs.io/notes/benchmarks.html.

[3] S. S. a. K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing,* vol. 30, no. 1, pp. 32-46, 1985.

[4] J. Sklansky, "Finding the convex hull of a simple polygon," *Pattern Recognition Letters,* vol. 1, pp. 79-83, 1982.

[5] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing,* vol. 1, pp. 244-256, 1972.

[6] D. D. &. T. Peucker, "Algoriths for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer,* vol. 10, no. 2, pp. 112-122, 1973.

[7] J. Y. H. L. Y. L. Xiaohu Lu, "2-Line Exhaustive Searching for Real-Time Vanishing Point Estimation in Manhattan World".

[8] [Online]. Available: https://github.com/rayryeng/XiaohuLuVPDetection.

[9] A. Y. James Coughlan, "Manhattan World: Orientation and Outlier Detection by Bayesian Inference," *In Neural Computation,* vol. Vol. 15, no. 5, pp. 1063-1088, 2003.

[10] [Online]. Available: https://github.com/ultralytics/yolov5.

[11] [Online]. Available: https://storage.googleapis.com/openimages/web/index.html.

[12] M. Patacchiola. [Online]. Available: https://github.com/mpatacchiola/deepgaze.

[13] [Online]. Available: https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html.

[14] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Sys. Man. Cyber.* , 1979.

[15] [Online]. Available: https://github.com/facebookresearch/detectron2.

[16] [Online]. Available: https://storage.googleapis.com/openimages/web/index.html.

[17] [Online]. Available: https://github.com/rayryeng/XiaohuLuVPDetection.