Computer vision and Cognitive Systems project for the Galleria Estense Museum

Arman Arnautovic
227078@studenti.unimore.it

Vipul Kumar
228630@studenti.unimore.it

Lorenzo Schiroli
231849@studenti.unimore.it

Abstract

The following report is a detailed description of the Computer Vision and Cognitive Systems course's software project, which provides for the creation of a pipeline capable of processing images and videos of paintings relating to the Galleria Estense Museum in Modena.
The characteristics of the project are not explicitly linked to hardware constraints and aren't based on particular performance analyzes.

1. Introduction

The project's purpose is to describe, implement and test an application capable to recognize and operate to museum's painting objects, given a video or an image. In particular, the main functionalities are:

1. *Painting Detection*: using a neural network based on *Faster RCNN*, this step provides to recognize paintings with an eterogeneous shape and to compute and draw on them correctly bounding-boxes.
2. *Painting Rectification*: Depending on the shape of the painting, this phase must be able to adapt to the distortion of the image and therefore be able to rectify the painting in such a way that it's better recognizable. This works on each bounding box obtained from the previous step.
3. *Painting Retrieval*: Given each recitified painting in their bounding box, this phase must be able to recognize the painting and to associate it in the best possible way the correct name and author, as well as the room to which it belongs.
4. *People Detection*: Similar to the first functionality, the neural network must also be able to recognize any people present in the video or in the image.
5. *People Localization*: Having recognized a potential person in the video / image, the algorithm must be able to place the person in the correct room of the museum.

2. Related Works

In order to learn in the best possible way the modernities that belong to the world of Computer Vision we certainly thought of realizing the project by approaching directly on known and already developed technologies such as the Faser R-CNN neural networks for the identification of paintings and people (trained with images taken from Google and repositories on the web), technologies such as ORB (Oriented FAST and Rotated BRIEF) to identify the keypoints between images useful for the retrieval of the paintings, but also trivially the evaluation of the detection we based on known tools like *"A Survey on Performance Metrics for Object-Detection Algorithms"* [4].

By doing so, not only we could have guaranteed results to make everything work within an acceptable error, but we also have the opportunity to manage things in our own way, by exploring these tools in depth to be able to also understand how we can adapt them to our design needs.

3. Approach

Taking into account the related works, below we illustrate the detailed characteristics of the pipeline introduced a little while following what are the assigned tasks.

3.1. Painting Detection

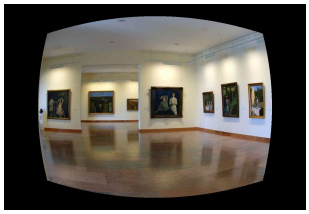
For this task, we used a library called Detecto that uses a Faster R-CNN ResNet-50 FPN [2]. It gives to the developer very basic and high-level functions making simple the detection of objects using a *FastRCNNPredictor box predictor* included in the PyTorch module.
The reason because we have chosen Faster R-CNN is because of its precision in detecting objects but most of all the project's pipeline consists in working with two types of painting shapes: oval and rectangle. This precision looks like very important in order to do a correct *painting rectification* in the next step, and therefore also the *painting retrieval* and so on.

3.1.1 Fetching Images

To create the dataset for bounding boxes we downloaded images from google images of museum paintings with similar patterns like the painting frame or the light color of the wall, in this manner, the convolutive network will learn these common patterns.
To make the detector stronger and to increment the size of the dataset, we have done different kinds of data augmentation:

- *Perspective changes*

- *Horizontal and vertical motion blur* for sudden movements of the camera



- *Brightness changes* with more and less brightness.

- *Barrel effect* to train the detector even for GoPro effect:

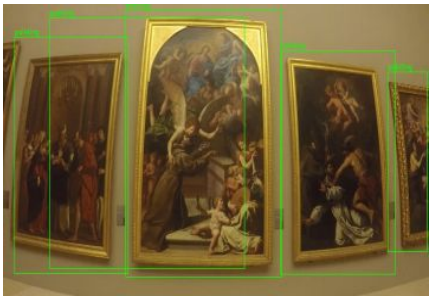
3.1.2 Annotate Labels For Training

For this step was used a tool for label annotation called *labelImg* that includes the Pascal VOC annotation.
Pascal VOC provides standardized image data sets for object detection using XML files. We create a file for each of the image in the dataset with a XML format.

LabelImg generates XML files (in Pascal VOC data format) after the user has created manually the bounding box for rectangular paintings and for oval paintings. In order to make the rectification work correctly, the bounding boxes are taken bigger, including a small part of the wall. These files contain coordinates and other information about bounding boxes.

All these images and xml files form a custom train set for the training of Detecto.
To train the neural network we used 5 epochs because experimentally the results are good such as the compromise between accuracies and time of training.

3.1.3 Correcting Errors



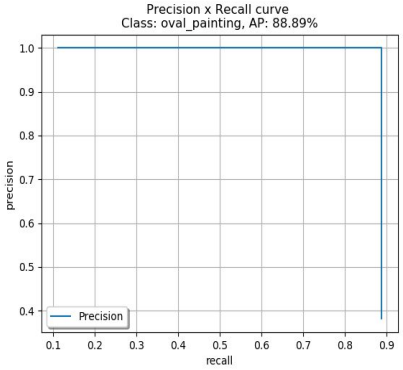
For this step we decided to establish a threshold of acceptance score of 30%, obtained experimentally, in order to remove detections with lower scores because those are wrong where most of them are false positives, to be more precise it means that they are detected as paintings but actually it's not true.

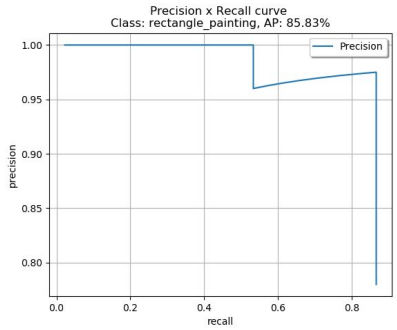
3.1.4 Painting Model Evaluation

For painting detector evaluation we have used the **mAP** (mean average precision) with a threshold for IOU (Intersection on Union) of about 80%.
The average precision of the two painting classes are:

- rectangle_painting: 85,83%
- oval_painting: 88,89%

So the mean average precision of the painting detector is the mean between the two APs: **87,36%**.
So, according to this result, we can say that choosing Faster RCNN was a good choice.





For the computations and plots we used a shared tool on GitHub [4].
The formulas used by the Python module are:
With the first formula we calculate the AP for every class considering the maximum precision for every recall layer (in this case are 11).

$$\mathrm{AP}_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots, 0.9, 1\}} P_{\mathrm{interp}}(R),$$

$$\mathrm{AP}_{\mathrm{all}} = \sum_{n} (R_{n+1} - R_n) P_{\mathrm{interp}}(R_{n+1}),$$

$$P_{\mathrm{interp}}(R_{n+1}) = \max_{\tilde{R}: \tilde{R} \geq R_{n+1}} P(\tilde{R}).$$

The mAP of the object detector is calculated as the mean of the APs.

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i,$$

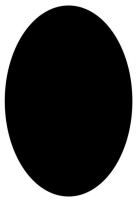
3.2. Painting Rectification

For the Painting Rectification task we consider two kind of modus operandi, one for each type of painting, according to the detection results: oval painting rectification and rectangle painting rectification.
Of course this task works on the single frame's bounding box portion and the final results depends significantly on the quality of the previously detection.

In both cases we decided to operate using the class IV perspective transformation known as homography or collineation.
In particular we used two common image patterns, each one for each kind of rectification, in order to use the openCV's function *getPersectiveTransform*(src, dst, mapMatrix), after a quite similar image pre-processing pipeline in which the aim was reaching a qualitatively and quantitatively sufficient number of points which are, as is well known, necessary to do this kind of operation.

3.2.1 Oval Painting Rectification

As far as the rectification of oval-shaped paintings is concerned, we have decided to base ourselves on the following approximate pattern.



Since we are talking about a pattern common to all, the points we decided to use to make the transformation are, intuitively, the well-known maximum and minimum vertices (with respect to the

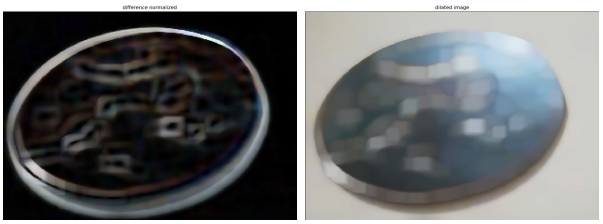
x-axis and the y-axis) of the oval itself.
At this point the challenge we set ourselves was to figure out how to get the corresponding points from a picture identified as oval. Considering one of the paintings that we considered to be one of the most complex to rectify, below will be listed the steps necessary to obtain the best possible result, as appropriate.

3.2.1.1 Filtering

Considering the complexity of the task we thought it necessary to operate on the painting with an appropriate filter, the bilateral filter in this case, with the aim to remove as much noise as possible and, especially, to preserve the contours of the painting, implementing the so-called edge-preserving smoothing.
Although the bilateral filter is one of the slowest filters (compared to averaging, for example) we thought it was the most suitable because the situations that characterize the video footage of paintings can be very heterogeneous and this can be a hard obstacle in the case of static filters such as Gaussian Blur and Median Blur.
Taking note of this heterogeneity, the bilateral filter is the most adaptive because it considers the similarity and the distance between the neighbouring pixels and the one considered.

3.2.1.2 Edge Highlighting and Shadow Suppression

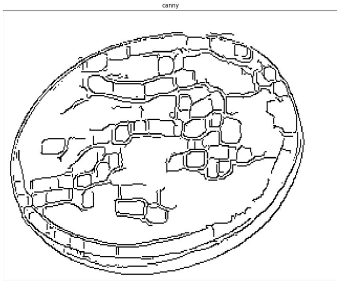
At this point the goal is to highlight the areas of interest in order to obtain the candidate points. The actual operation will be formalized with edge detection, but to facilitate the identification of the edges the idea would be to clean the image by normalizing the difference between the image and its dilated version.
Dilation (15x15) is useful because the difference with the undilated image can preserve and highlight the contents of the painting and its edges.



After this operation of highlighting the edges, the next step is to erode the image in order to remove as much as possible the shadow. The removal of the shadows may not affect the rectification much more, but there may be cases in which they take on such ambitious dimensions that even a sufficient suppression could significantly improve the identification of the points.

3.2.1.3 Edge Detection

For obtaining in a clear way (and also for a matter of speed) the pixels candidates to be part of the points we are looking for, an additional operation to the edge highlighting obtained by the Bilateral Filter and the previous cleaning is the edge detection.
In particular it has been decided to use Canny Edge Detector, as it provides not only a concrete edge detection but also a non-maximum suppression to remove possible extraneous edges and a hysteresis thresholding based on a fixed minimum threshold of 10 and a fixed maximum threshold of 50.
It is clear that a pair of low-value thresholds like this one can also highlight the content of the painting and detect so many edges but, being that the ultimate goal is to extract the maximum and minimum points, in any case these edges will be discarded.
Considering these hyperparameters we could visualize more edges than those that can actually serve, but on the other hand it reduces the risk that those interesting for the warp perspective are not suppressed.



3.2.1.4 Source Point Extraction

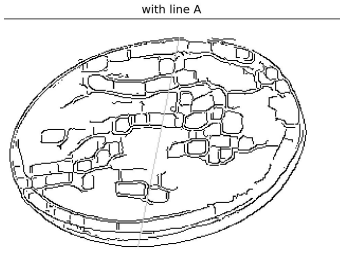
In order to achieve the interesting points to make the

homography it's assumed that the vertex points with respect to the y-axis are always respectively in the maximum and minimum points's range of the entire image, so the assumption can be explained, in other words, as the fact that the picture is taken in an acceptable perspective to be able to say that.
It's already possible to guess how fundamental it's to be able to highlight the edges and to remove as much as possible the shadows and noises from the original image in order to avoid that possible outliers are considered in this phase of the pipeline.
The first operation is to obtain the coordinates of the maximum and minimum pixels with respect to the y-axis, obviously of the same color of the edges obtained from the edge detection.
More precisely, considering the fact that there could be more than one pixel with the same y-coordinates, we simply take the mode bewteen them. This choice may not be perfectly precise but we considered it a good compromise to generalize at best according to the case, in terms of perspectives and resolution.

$$top_{point} = mode\,\{argmin_y((x,y)\ \in\ img\ \mid\ v(x,y) = 0\}$$

$$bottom_{point} = mode\,\{argmin_y((x,y)\ \in\ img\ \mid\ v(x,y) = 0\}$$

Once these points have been obtained, actually a straight line (we call it *line A*) that connects them is drawn and this fact allows us to extract useful information for obtaining the left and right extreme points, in our case the **slope**.



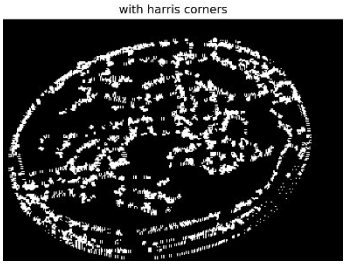
Given those points the *line A* which connects them together, the next operation we implemented is to obtain the **best approximation to the exact perpendicular line** w.r.t. the one we achieved, with the only constraint of having to go through the middle point of the straight *line A*.
We mentioned the *slope* before because the aim now is to achieve two surrogate lines *B* and *C* in order to obtain the *perpendicular line* w.r.t. the *line A*, which passes through the middle point obtained with the following formula:

$$middle_{point} = \{(x,y)\ \in\ img \mid\ v(x,y) = (200,0,0)\ \wedge\ y \simeq \mid y_{top_{point}} - y_{bottom_{point}}\ \mid\}$$

where (200,0,0) = line A's color

The middle point explained above is another approximation of reality: in fact it's not said that this point is the exact center of the painting (with respect to the y axis), however through various tests we've believed that this assumption can adapt sufficiently as appropriate.

The next step in order to obtain the best approximation to the perpendicular line is to achieve the **extreme corners** of the object, using the **Harris Corner Detection**. More precisely, we decided to identify as many points as possible, therefore with a very low threshold (in the order of 10 ^ -3), in order to obtain the points that we need in the most accurate way.



Line A, more precisely the y-coordinates of the two top and bottom points, will act as separators of the corners in the left and right side of the processed painting. This division will be used to consider separately the search for the lines B and C mentioned above.

Considering the **slope m** of *line A*, the modus operandi of this phase will be to search for the best straight line that connects the *left corners* with the middle point (analogous also on the right side). Note that this division has been applied to force the passage of the final perpendicular line to the central point.
Limiting ourselves to the left area (the right is analogous), the straight *line B* we are looking for will be the one that will present, among many others, the most similar *slope* to that of the optimal perpendicular straight line (-1/m).
The research carried out is exhaustive and is done on every possible combination of lines between the *left corners* and the middle point.
The main problem that can occur with this approach is that the corners inside the painting can also be considered, which therefore form part of the content, distorting the final result. To overcome this, we simply opted for a search for a set of straight candidates to be the final one, using the following criterion

To avoid choosing the internal points of the painting, simply select those that have the component x that is farthest from that of the central point.

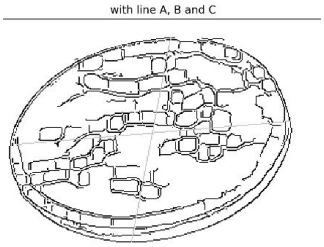
$$candidates_{left} = \left\{(x_l,y_l),(x_{middle},y_{middle})\ \in\ img\ \mid\ v(x_l,y_l) = 255\ \wedge\ \mid slope_{(x_l,y_l),(x_{middle},y_{middle}))} + \frac{1}{m}\mid < 0.4\right\}$$

$$where\ slope = \frac{y_l - y_{middle}}{x_l - x_{middle}} \quad and$$

$$m = \frac{y_{top} - y_{bottom}}{x_{top} - x_{bottom}}$$

$$left_{point} = argmax_{(x,y)} \{D((x,y),(x_{middle}, y_{middle})) \mid (x,y) \in candidates\}$$

It's analogous to the right side.

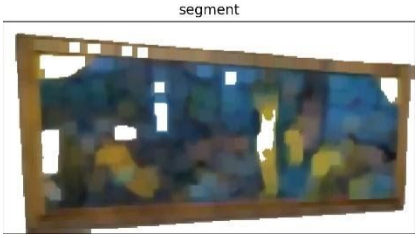


The final result can be represented as follows
Having obtained the four points that interest us, now all that remains is to do **homography** between the **common oval pattern** explained at the beginning and the original image. Obtaining the perspective transformation and applying the *warpPerspective* function of the OpenCV library, the result based on the example shown so far is the following

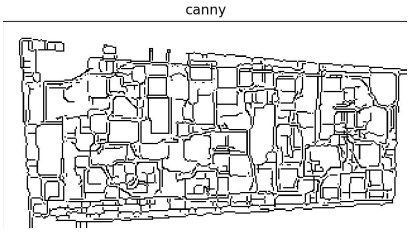
3.2.2 Rectangle Painting Rectification

As for the rectification of paintings classified as rectangular, the pipeline illustrated so far appears to have a modus operandi very similar to that used for this case.

In fact, the image pre-processing step is almost the same. While no **segmentation** had been implemented in the oval, probably because the shape of the painting itself did not allow effective segmentation, in this case it's applied with previous **opening**, all before cleaning. This is simply because the image itself, through some tests, has shown that it can present significant noises such as portions of labels or shadows which, with the oval version, did not occur frequently. The segmentation, by means of the OTSU thresholding, in this case makes the difference because it manages to extract the painting well in most situations, on which the edge extraction procedure turns out to be even more efficient, while the opening with 8x8 kernel is able to very often remove unpleasant shadows and details.

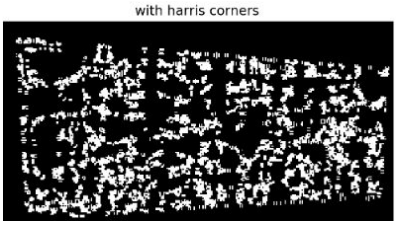


After the same cleaning and edge detection step of the Oval painting case, the result looks like the following one



3.2.2.1 Source Point Extraction

The source point extraction, in this case, turns out to be simpler w.r.t. the previous rectification case. In fact, the objective here is to extract the **corners of the painting** as the points for the homography transformation.
To do this, even in this case, it was decided to use **Harris Corner Detection** again to extract as many points as possible from the pre-processed image, in exactly the same way as previously seen.



Now, obtained those corners, the purpose's to extract only the **corners of the painting**, that is, the points that have coordinates closest to those of the bounding box, like the following

Having obtained these points, with the obvious assumption that they are the points we are looking for (thanks to the preprocessing and corner detection and filtering phase we can approximate them well as appropriate) the rectification is nothing more than the application of the *warpPerspective* function of the library of OpenCv with source points those just obtained, and those of destination the quadruplet obtained in the following manner

Based on the example just illustrated, the final correction will be represented as follows

rectified

3.3. Painting Retrieval

In the step image retrieval, we used ORB [3] (Oriented FAST and Rotated BRIEF). We preferred ORB in place of SIFT since ORB is faster. Moreover, SIFT is a non-free software and is not present in the current versions of OpenCV. There is also to consider that since SIFT is not integrated in OpenCV could lead to compatibility problems between packages if we use it.

Each time there is a new image to analyse we extract its features and compare them with the database images features. To be more precise we have done the following steps:

- Before executing the pipeline, we compute the keypoints and their descriptors for all the database images and then we store them (in order to optimize the process). The keypoints are found using ORB with its default settings.
- During the pipeline execution, when there is an image to analyse (target), we compute its keypoints and descriptors and compare them with the keypoints and descriptors of each database image, obtaining a set of couples of keypoints for each couple of target images – database image. The comparison between keypoins is done using brute-force matcher with Hamming distance.
- After that, we use a threshold to consider only strong couples of keypoints (low distance) and then we count them obtaining the score.
- At the end we have a score for each database image that is proportional to the similarity with the target one, thus the database image with the highest value is the most similar to it. The paintings are ranked with decreasing order of similarity. (example of a rank: {“020”: 32, “010”: 4, “003”: 0, “027”: 0,”083”: 0, “091”: 0 ...})

The first element of the rank is chosen as the retrieved painting, but only if it has at least a score of 3 (threshold 3). This is done because, doing several tests, we noticed that usually a retrieved painting with a score lower than 3 is not the correct one, therefore it is discarded.

We tested 3 different thresholds (retrieval score at least 2, 3, 4), in order to choose the best one, and with or without the rectification, in order to see how much the rectification is effective. This test is made with a test set of 40 frames.

\	threshold 2	threshold 3	threshold 4
with rectification	P=0.59 R=0.72 F1=0.65	**P=0.77 R=0.62 F1=0.69**	P=0.82 R=0.52 F1=0.64
without rectification	P=0.45 R=0.56 F1=0.50	P=0.63 R=0.46 F1=0.53	P=0.73 R=0.38 F1=0.50

P is the precision, R the recall and F1 is the F-score.

Looking at the results we can tell that rectification improves the performance and that the best threshold is 3 (science it is the best tradeoff between precision and recall).

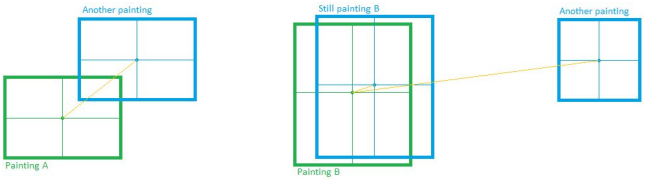
Precision, recall and F-score are computed with usual formulas:

$$\text{Precision} = \frac{TP}{TP+FP} \quad \text{Recall} = \frac{TP}{TP+FN} \quad F_1 = \frac{2}{\frac{1}{r}+\frac{1}{p}}$$

where TP, TN, FP, FN are interpreted in this way:
TP = painting is retrieved and is correct
TN = painting is not retrieved and is not present in the database
FP = painting is retrieved but is incorrect
FN = painting is not retrieved but is present in the database

There is another thing to consider. We observed that sometimes retrieval is not very precise and stable since generally in most of the frames the retrieved painting is correct but in some frames is wrong (on the same painting). Therefore, it looks like the retrieved painting changes frequently frame by frame. In order to make the retrieval more stable we added a memory system that remembers the best retrieved painting (till that moment) for each painting.
Each time a painting is detected the program tries the retrieval (on the current frame) and compares it with the best retrieval of that painting till now (the best on the past frames). If the current retrieval is better (higher score), then it becomes the best retrieval for that painting. In the end it returns just the best retrieval as the retrieved painting, discarding the current retrieval if worse (lower score).
To make it work, the system needs a way to track the paintings’ bounding box through the frames. It works like this: for each

bounding box of the current frame it looks at its nearest bounding box of the last frame (computing the distance of their centers), and if it is near enough, it is considered the same bounding box. In this way it is able to associate to each painting's bounding box their best retrieval.
Using this trick, the retrieval is more stable and more correct.

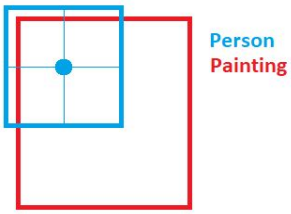


In the image above there are 2 consecutive frames overlapped. The blue rectangles are the paintings' bounding box of the current frame while the green ones are the paintings' bounding boxes of the last frame. The nearest bounding box of the blue one on the left is painting A, but it is too far, so it is considered a new painting. The nearest bounding box of the blue one in the center is painting B and it is quite near, therefore it is considered the same painting. The nearest boundix box of the blue one on the right is painting B but it is too distant so it is considered a new painting.

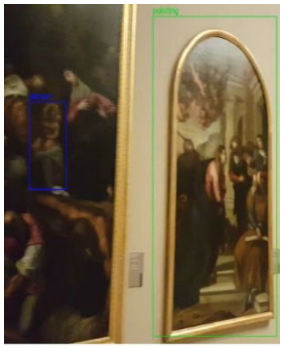
In the console, for each painting in that frame, it displays the best retrieval till now followed by the ranking of the current retrieval (in the current frame).
Example:
Best retrieval of the last frames: 040
Rank of current frame: [('067', 2), ('046', 1), ('048', 1), ('052', 1), ('062', 1), ('091', 1), ('000', 0), ('001', 0), ('002', 0), ('003', 0)]
This means that the current retrieval is 067 but the best one till now is 040, indeed 040 will be displayed and 067 will be discarded.

3.4. People Detection

For people detection, at the beginning, we decided to use YOLOv3 [1] but we have chosen even in this case Detecto (that uses Faster RCNN) because it is more efficient and precise.
For people detection we didn't use any custom training sets because the "person" label is very complex to train. So we used another model that is pre-trained on the COCO dataset, where database's labels are already included in PyTorch modules with the class FasterRCNNPredictor box predictor.
We noticed that there are a lot of false positives inside the paintings (a lot of characters in the paintings are detected as persons). To avoid this problem, we discard the people bounding boxes that are inside the paintings ones. To be more precise, if the center of the person's bounding box is inside a painting one, it is discarded.
To filter other false detections we have exploited the people's



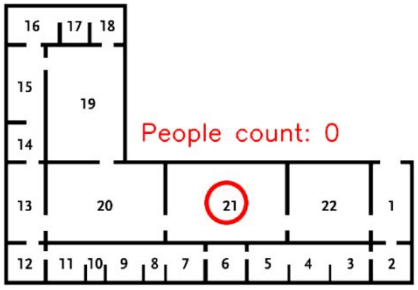
detection score given by the neural network and setting a threshold of 0,85 obtained making various tests.
Obviously, the problem of this approach is that if the painting is not detected, people's bounding boxes inside the painting cannot be deleted. We can see the problem in the image here, but this kind of error happens a few times.



3.5. People Localization

People localization is done with a simple approach: the system recognizes the room where the cameraman is recording and associates it to the people that appear in the video. The room where the video is taken is obtained by considering the most popular room among the paintings. To be more precise, firstly, the paintings in that frame are detected, retrieved and localized using the file containing the paintings information. Then, the paintings are counted by room and the one with the highest count is considered the room where the cameraman is recording. Finally, that room is treated as the one where the people in the video are standing.
The system memorizes the room's number. Even if there are no paintings recognised in that moment, the system displays anyway the last detected room.

People localization is displayed drawing a circle in the corresponding room on the museum map. In order to do that, we created a file containing the pixel coordinates of the rooms. Moreover, near the map there is also displayed the number of people detected at that moment.



4. Discussion

In making this project we learned several solutions about computer vision and its challenges.
We used neural networks more than machine learning algorithms because they are more performant and precise, but also more resource consuming.
For this project we have used a lot of false positives, like in Painting Retrieval to retrieve as much as possible the correct painting in the database or in Painting Rectification to find the four points to apply the homography transformation. We learned that to filter these false positives in the best possible way we have to use good thresholds to obtain good tradeoffs.
Either in order to understand the most common problems that characterize images and then choose the most suitable image pre-processing operations, or even trivially understand that for neural networks there is a need to have a large amount of data to predict certain imperfections

5. References

[1] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." *arXiv preprint arXiv:1804.02767* (2018).

[2] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems*. 2015.

[3] Rublee, Ethan, et al. "ORB: An efficient alternative to SIFT or SURF." *2011 International conference on computer vision*. Ieee, 2011.

[4] Padilla, Rafael, Sergio L. Netto, and Eduardo AB da Silva. "A Survey on Performance Metrics for Object-Detection Algorithms." *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 2020.