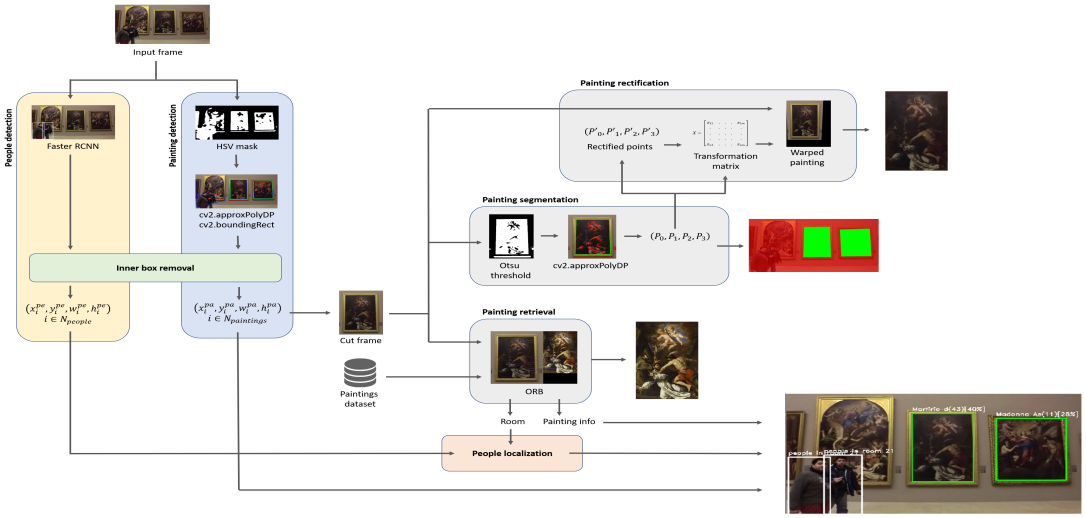
Vision and cognitive system final project

Bertellini Paolo 149491
Caputo Imbriaco Davide 142586
Doganieri Giuseppe 145099



Abstract

The purpose of the project is to create a visual system capable to detect and recognize the paintings exhibited in "Gallerie Estensi" museum of Modena and locating people that are visiting the museum understanding the room in which they are. The system in the future can be used by a robot for visiting the museum in an autonomous manner or can be installed on a mobile application to improve and guide the visitor's experience.

1. Introduction

The main aspects of the project's pipeline are presented in this section. The motivations behind the choices made and the details of each task are discussed in the following chapters.

The solution proposed takes as input a single frame from a video and passes it to the following pipeline:

1.1. Detections

People and paintings are detected through two different approaches.

- *People*: given the large amount of data available on the web people are detected with a deep learning approach. In particular a faster RCNN pretrained on COCO dataset is used. The model returns a list of detection with the corresponding labels and bounding boxes.

From the 80 COCO classes only the class "Person" is taken into account and all the others are discarded .

- *Paintings*: the necessary amount of data to train a classifier is not available, therefore an image processing approach is used. It is based on two main considerations:
  - The wall has a lighter color than the paintings;
  - Almost all the paintings are rectangular and the most of the circular ones have a rectangular frame.

  Starting from this two considerations the model creates a mask filtering the pixels of the wall that have hue between 80 and 255. The contours found in the mask are approximated to polygons and only the ones with more than four side are taken into account.

1.2. Inner boxes removal

The boxes containing the paintings and the people detected are compared together in order to discard boxes that are contained in other boxes. This control allows the model to:

- not consider people portrayed in the paintings;
- consider only the outermost border if more than one is found for a single painting.

1.3. Painting segmentation

Each painting detected and acknowledged by the inner box check is cropped from the frame with a padding and segmented using the Otsu threshold algorithm. The segmentation allows to detect the borders of the paintings with more precision and in particular to localize the four corner points that are crucial for the perspective rectification.

1.4. Painting rectification

The four edge points B found with segmentation are used to compute the height H and the width W of the rectified version of the painting. Using W and H the model computes the new rectified box B' with perpendicular corners. Then it warps the original frame using the transformation matrix obtained from the two set B and B' and it crops the warped image in order to select just the painting region.

1.5. Painting retrieval

For retrieval the information of the detected painting are computed the ORB descriptors on the crop portion of the frame and they are compared with the ones of all the paintings stored in the database. The function returns a sorted list of the number of strong matches found for each painting. The retrieval is considered reliable if there are almost 20 strong matches.

1.6. People localization

The *peopleLocalization* combines the results of *peopleDetection* and *paintingRetrieval*. Whenever a person is detected the model is able to localize her retrieving the room from the database information of a painting recognized in the same frame of the person.

2. Painting detection

The function *paintingDetection* takes as input a single frame of a video and passes it through a 25x25 median blur filter to reduce the Gaussian noise due to camera quality.

The frame is then converted to HSV colour space in order to filter the pixels of the wall and create a mask that using the *cv2.inRange* function excludes them. Observing the videos it is possible to see that in most cases the hue value of the wall's pixels is between 80 and 255. This values are obtained from many tests done on the videos changing the mask's filtering range in real time with trackbars.

This values work well for the great majority of the cases but in condition of too high or too low brightness, low difference between the wall and the painting colour and in case of reflexes on the painting the mask is not particularly efficient. In development phase changing the filtering range with the trackbars is possible to obtain a better mask. For

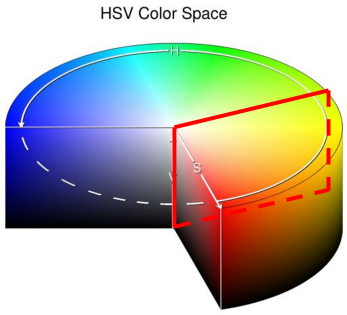


Figure 1: From the image it is possible to see that the mask filters all level of saturation (S) and value (V) from the pixels between red and yellow hue. This allows to extract the paintings and discard the walls.

example changing the saturation value it is possible to remove the reflexes. A possible future improvement of the solution can be to adapt the filtering range to the image characteristics.

Figure 2: The figure shows the differences between a good mask (the two on the top) and a mask that needs some corrections (the two on the bottom).

The negative version of the obtained mask that binaries the image putting to white the painting pixels and to black the others is then passed to the *cv2.findContours* function in order to detect the contours.

For each contour detected a rectangle approximation through the *cv2.approxPolyDP* is computed and only the rectangles that have an area greater than 15000 pixels and a number of sides greater or equal than 4 are taken into account. This area is empirically obtained with some tests in order to select only the paintings and to discard smaller rectangle shapes. The function at the end returns a list with the top-left corner coordinates, the width and the height of the detected paintings.

3. Painting segmentation

The function *paintingSegmentation* takes as input the crop portion of the frame containing the detected painting with a padding of 60 pixels. The detected painting is passed with a padding to include possible missed parts from the detection and to have also some pixels of the wall to better understand the shape and the limits of the painting.

The crop portion is then converted in gray scale and it is denoised using the *cv2.medianBlur* function with a 5x5 kernel.

On the blurred image is then applied the Otsu thresholding with *cv2.threshold* function. The mask obtained, as in the *paintingDetection*, is inverted in order to have white pixels for the painting and black pixels for all the others elements. The contours of the painting are computed passing the inverted mask to *cv2.findContours* fucntion. All the contours found are approximated to polygons using *cv2.approxPolyDP* function.

The polygons found are filtered in order to select just the ones that have four sides and an area at least greater 1/3 of the total area of the image. This controls are done in order to consider only quadrilateral shapes that are big enough to be a painting. If there are more then one polygon with the researched characteristics the function returns just the points of polygon with the smaller area because the other ones are probably generated by the frame of the painting.

Before to return the points of the best polygon found they are ordered with the *orderPoints* function. That function, contained in the *utils* file, sorts

(a) Segmented corner points (b) Rectified box (c) Warp perspective (d) Cropped painting

Figure 3: Example of how rectification works.

the points from the top left to the bottom left clockwise in order to have always the same order.

Otsu turned out to be the best solution for the thresholding segmentation of the paintings. It works by minimising the intra-class variance between the two classes and the decision to use it only on a crop portion of the frame and not since the beginning on the whole frame comes from some observations:

- in most cases, in a frame there is more than one painting;
- paintings on the same wall can have different appearance;
- the brightness is not homogeneous on all the wall and all over the paintings.

Using Otsu just on the painting ROI allows to segment with more precision the painting maximising the homogeneity of the two classes.

4. Painting rectification

The *painting rectification* function takes as parameters the crop frame with the detected painting and the four segmented edge points of the painting $B = [P_0, P_1, P_2, P_3]$.

The function computes the dimensions for the new rectified image using the euclidean distance of the top right edge point with respect to the top left and the bottom right ones like is possible to see in Figure 3a.

$$H = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$W = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$$

Using P_0 as reference is then computed the rectified box B' with height H and width W showed in Figure 3b.

$$P'_0 = (x'_0, y'_0) = (x_0, y_0)$$
$$P'_1 = (x'_1, y'_1) = (x_0 + W, y_0)$$
$$P'_2 = (x'_2, y'_2) = (x_0 + W, y_0 + H)$$
$$P'_3 = (x'_3, y'_3) = (x_0, y_0 + H)$$

With the 2 set of points B (the original ones) and B' (the rectified ones) it is possible to compute the transformation matrix using *cv2.getPerspectiveTransform* function.

The matrix obtained is used for warping the crop frame on a new black frame with the *cv2.warpPerspective* function obtaining the persperctive correction as showed in Figure 3c.

Using the B' points is then possible to select and to cut only the region with the painting in the warped image as it is possible to see in Figure 3d.

5. Painting retrieval

The *painting retrieval* function creates a ranked list of all the images in the painting DB sorted by descending similarity with the detected painting.

The painting database is composed by a list of dictionaries containing:

- **Title**: the title of the painting
- **Author**: the author of the painting
- **Room**: the room containing the painting

- **Image**: the *png* file name of the high resolution image of the painting
- **Descriptors**: the painting descriptors created by ORB
- **Painting**: the high resolution image of the painting

The database is created when the program starts. Files, Authors, Rooms and Images data are read from a *csv* file, while Paintings are loaded from the *png* files and Descriptors are computed in real time.

The descriptors of the paintings are computed with ORB [4]. It is a good alternative to *SIFT* and *SURF* in terms of computation cost and matching performance and it is included in new version of opencv. A ORB object is instantiated and it is used its *detecteAndCompute* method that given an image returns the key-points and their descriptors.

So the *paintingRetrieval* function receives as input the database and the frame portion with the detected painting. It receives the original cropped frame and not its rectified version because ORB is invariant to image transformations and not always it is possible to have the segmentation and rectification of a painting. Then, the function computes the descriptors of the detected painting and compares them with the ones of all the paintings in the database using the *Brute-Force Matcher*.

The *Brute-Force Matcher* is able to compute the distances between the descriptors of two images using *cv2.NORM_HAMMING* that is considered the most indicate distance measure for ORB descriptors. Once the *BFMatcher* object is created the *BFMatcher.knnMatch()* returns the k=2 nearest DMatch for each query keypoint.

The two DMatch objects have the following attributes:

- **DMatch.distance**: Distance between descriptors. (The lower it is, the better);
- **DMatch.trainIdx**: Index of the descriptor in train descriptors;
- **DMatch.queryIdx**: Index of the descriptor in query descriptors;
- **DMatch.imgIdx**: Index of the train image;

From all the DMatches found it is necessary to consider only the strongest ones. In order to do that it is necessary to control the difference between the first best match and the second best one.

The best match, the one with the smallest distance, is considered the ”good” match, while the other, with the greater distance is considered random noise.

If the distance of the ”good” match is less then 0.75 times the distance of the other it means that the two descriptors are very similar and so they don't bring any interesting information-wise because they can't be distinguished from noise.

The function creates a list of dictionaries that contains the index of the paintings and the number of the strong matches found and sorts it by descending number of good matches.

The list is so ordered by descending similarity and its first element represents the most similar painting in the database.

Since there are some missing paintings in the database, the function controls the similarity score and visually represents the confidence about the retrieval changing the color of the bounding box showed in the detection:

- if the score is grater than 20 it considers reliable the result of the retrieval, sets the color for the bounding boxes to green and shows the high definition painting;
- if the score is between 10 and 20 the result is not completely reliable so it sets the color to yellow;
- for all the smaller values of the score the color is set to red because the model is not able to find any similar painting.

At the end, the painting retrieval function returns the ranked list of all similarity scores of all the pictures in the database with the detected painting, the color of the bounding box, the information of the most similar painting and the image of the painting itself if the scores is greater than 20.

6. People detection

People are detected trough a deep learning approach passing the input frame and NN model to the *peopleDetection* function.

The model used is a ResNet50 faster RCNN [3] with pretrained weights. Depending on the different situations in which people are, it is possible to use different weights:

- the ones pre trained on *COCO Dataset* with people in general contests
- the ones finetuned on *Penn-Fudan Database* specific for pedestrian detection [1]

COCO is a large-scale object detection, segmentation, and captioning dataset. It has 80 classes and for the project purpose it is just the 'Person' class to be taken into account . The weights for the classification of this 80 classes with the selected model are taken from pytorch *torchvision.models.detection. fasterrcnn_resnet50_fpn(pretrained=True)*.

The other possibility has been made modifying the last layer of the model in order to predict just the 2 classes 'Person' and 'Not person' and finetuning the COCO weights with a pedestrian dataset. This choice was made in order to avoid the detection of sculptures and of people portrayed in the painting given the fact that in many cases visitors of the museum are in movement.

The new model was retrained on google colab GPUs `https://bit.ly/2UwXBQH` on the new dataset for some epochs saving the weights found in a .pt file. [2]

The finetuned weights are more precise on walking people but they doesn't recognize sitting or simply standing people. Comparing the two possibilities on the available videos the first model turns out to have better performance since in too many cases visitors are standing in front of artwork.

7. People localization

The *peopleLocalization* is based on the results of *peopleDetection* and *paintingRetrieval* functions. Whenever a person is detected the model is able to localize her only if a painting is recognized in the same frame. Once the model recognizes also a painting looking at his information in the database it is possible to retrieve its room and assign it also to the person. The result is visually represented by a green square that empathizes the retrieved room on the museum map.

In some videos there could be frames with more than one room. In this situation this approach could fail if the recognized painting and the people are in the same frame but not in the same room.

In this situation though the results are not correct, they are still a good approximation of the people's position since the fact that if the person is filmed in the same frame of the painting, means that they are close to each other.

8. Conclusions and considerations

The solutions proposed for each task have been tested on all the videos multiple times using an interface able to show a continuous sequence of videos randomly picked from the dataset.

Having a not annotated dataset is not possible to do quantitative considerations or compute metrics but is although possible to do some qualitative considerations:

- the *detection* in standard conditions is able to find the majority of the paintings. In some cases there are some false positives like metal railing, corner of the ceiling or doors due to the fact that all the squared shapes are detected;
- the *segmentation* struggles a little bit more than the detection because it requires more precision to detect the corners so it needs less approximation of the straight contours. Relaxing the sensibility is possible to segment more paintings but with less precision that instead is fundamental for the rectification task;
- *remove inner box* fails on boxes near to the frame borders and in boxes only partially overlapped;
- the *painting retrieval* has a good precision. The only cases in which it fails are due to the poor painting database with just only 95 painting or to low quality video;

- the *people detection* is able to detect people with a good recall but with some false positive due to persons portrayed in undetected paintings and statues.
- the *people localization* uses a very simple idea that is exposed to errors or mismatches when are detected in the same frame a painting and a person that are in different rooms.

References

[1] Penn-fudan database for pedestrian detection and segmentation `https://www.cis.upenn.edu/~jshi/ped_html/#pub1`.

[2] Torchvision object detection finetuning tutorial `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.ht`.

[3] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.