ArTection: an Art Detection Tool to Locate and Recognize Paintings and People in Museums and Art Galleries

Roberto Amoroso
University of Modena and Reggio Emilia
219620@studenti.unimore.it

Abstract

This work proposes a method to locate and recognize paintings and people in a museum or art gallery. For this purpose, I created a Python program that can locate and recognize paintings and people in a video or image. For the part relating to the paintings, I used the OpenCV library, while to carry out the people detection task I used YOLOv3, a real-time object detection system.

1. Introduction

Detect a painting, computing the transformation to rectify the image, and then comparing the image obtained with those stored in a database, are all nontrivial tasks. Detecting and analyzing paintings is of course of great interest to art historians, and can help them to take full advantage of the massive databases that are built worldwide.

At the core of many recent computer vision works, the object detection task (classifying and localizing an object) has been less studied in the case of paintings.

In recent years, many works have been developed that investigate the problems of image detection [7, 10], recognition [19] and retrieval [26].

Many of these approaches use Deep Learning techniques and are based on the use of Convolutional Neural Networks (CNNs) to carry out operations such as Painting Detection and Identification [12]. This implies the need to have a large amount of annotated data, necessary to train and test the model.

The approach I propose avoids this problem, by submitting the input through a processing pipeline which, using the OpenCV [4] library, performs a series of operations and transformations which carry out the following tasks:

- **Painting Detection**: detects all paintings in the input.
- **Painting Segmentation**: creates a segmented version of the input, where the paintings, and also any statues, identified are white and the background is black.
- **Painting Rectification**: rectifies each painting detected, through an affine transformation.
- **Painting Retrieval**: matches each detected and rectified painting to a database containing a list of the paintings in the museum or gallery along with related information, such as the title of the painting, author, room in which the painting is located. I use ORB [25] as a feature detector, to find keypoints and execute matching between an input image and the various images in the database.
- **People Detection**: detects people in the input using YOLOv3 [23], a state-of-the-art real-time object detection system, for which pre-trained weights are available.
- **People and Painting Localization**: locates painting and people using the information discovered during the painting retrieval phase.

2. Related Work

2.1. Object Detection

One of the main problems in the field of computer vision is object detection. In the last few years, numerous works have been published to propose a possible solution capable of solving this task, leading to a continuous improvement in performance. A milestone, which led to great progress in this area, was the use of CNNs.

Pioneer in this sense was R-CNN (Regions with CNN features) [9], who had the idea of using a selective search [29] image segmentation algorithm to generate many candidate regions for potential object instances before CNN is used to perform classification to these regions individually. Subsequently, other works [8, 24] have unified the localization and classification phases to improve the speed of object detection.

In the wake of these pioneers, many other works have been conducted [6, 14, 16, 17, 22, 21, 28, 23] aimed at further improving the performance of the architecture.

Figure 1. Example of input (left) and final output (right) produced by the proposed pipeline.

I decided to use a CNN-based object detector only to perform the people detection task. In particular, I selected YOLOv3 as it is a balanced system in terms of speed and accuracy, it comes with a well-organized code and pre-trained weights.

2.2. Image local features

Paintings, statues, and all objects present in a museum or art gallery can be filmed and photographed from various viewpoints and with different lighting conditions. This implies the need for a technique capable of representing an image as invariant to rotation, affine transformation, and some noise.

Hand-crafted image local features [25, 1, 3, 13, 15, 18, 20] can be used to solve these problems. These are techniques used for object tracking, image stitching, image registration, etc., i.e. all those applications that are based on finding correspondences between two images.

In this work, I selected ORB. It is a good alternative to SIFT and SURF in computation cost, matching performance and mainly it's not patented (SIFT and SURF are patented and you are supposed to pay them for its use). ORB is a good choice for low-power devices.

Here we have a brief introduction to ORB, for more information and the details about how it works read the original paper [25].

ORB is a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. First, it uses FAST to find keypoints, then applies Harris corner measure to find top N points among them. It also uses pyramid to produce multiscale features. ORB's main contributions are:

- The addition of a fast and accurate orientation component to FAST.
- The efficient computation of oriented BRIEF features.
- Analysis of variance and correlation of oriented BRIEF features.
- A learning method for decorrelating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications.

3. Method

3.1. Material and Data Preparation

The data available and representing the inputs of the program are:

- A series of videos recorded inside the Estense Gallery in Modena, Italy.
- A database consisting of 95 images of as many paintings.
- A CSV file containing important information on each of the 95 paintings in the database, such as the title of the painting, author, room of the museum in which it is located, filename of the image.

The videos were recorded using different devices, angles, and orientations, and during normal museum activity, so they often show people.

It was therefore necessary to create a program that was able to process not only individual images but also videos. In the latter case, the solution I adopted is to consider each frame of the video as an image to submit to the pipeline.

The pipeline, therefore, works with images as input and produces in output, for each of them, another image, on which the obtained information has been reported (painting and people bounding boxes, title of the paintings, number of the room where the paintings and people are located).

The Painting Rectification task represents a particular case. It requires that for each painting present in an input image or video frame, an image containing the rectified

version of the painting is saved as output. So in this case, against a single input, there are potentially multiple outputs.

One of the first problems I faced concerns the different resolution of the videos, which are provided in the formats: HD (1280x720), full-HD (1920x1080), and 4K (3840x2160).

Having inputs with different resolutions posed the problem of making the measures adopted in the various functions of the pipeline (often expressed in pixels) invariant with respect to the resolution.

The solution I adopted is to perform as a first operation of the pipeline a resize of all the inputs to the lower resolution, the HD (1280x720). Subsequently, the resized images continue in the pipeline and are subjected to the various tasks described below.

The information found (bounding boxes and information about paintings and room) is then subjected to an upscaling phase, i.e. the (x, y) coordinates of contours, corners, bounding boxes are multiplied by the scale factor for which the original image has been resized to obtain its HD version.

The output is generated by drawing this information on the original image. This allows me to maintain the original resolution in the output, and improve performance as the functions of the pipeline are performed on images of a smaller size (e.g. a ninth of the input size in the case of 4K images or videos).

The processed images, which constitute the output of the pipeline, are: directly stored in case the input is an image, treated as video frames, and stored as a single video in case the input is a video.

3.2. Painting Detection and Segmentation

To aid the description of the processing of locating, rectifying, and recognizing a painting in an image, let's suppose we want to recognize the painting '*Madonna col Bambino - Paolo di Bernardino di Antonio del Signoraccio detto Fra' Paolino da Pistoia (Pistoia 1488 - 1547)*', represented in Figure 2, inside the image present in Figure 3. The painting is located in room number 8 of the museum.

3.2.1 Obtain a Wall Mask

Inside a museum or art gallery, the paintings are objects hanging on the walls. These are assumed to be of single uniform color, as is usually the case in museums and art galleries to avoid distracting visitors from the artwork.

Based on this assumption, the first operation to be carried out to identify and segment the paintings is to distinguish which portions of the image are not walls.

To achieve this, a sequence of operations must be performed. First I apply the Mean Shift Segmentation operation on the image, which clusters nearby pixels with similar pixel values and sets them all to have the value of the local maximum of pixels value. The result is to have the pixels grouped by color and location, as can be seen from the image in Figure 4.

On the resulting image, I perform a second operation, called Flood Fill, which assigns the same color to connected components. The connectivity is determined by the color closeness of the neighbor pixels. That is, to be added to the connected component, the color of the pixel should be

Figure 2. '*Madonna col Bambino*', the painting we want to locate, rectify and recognize

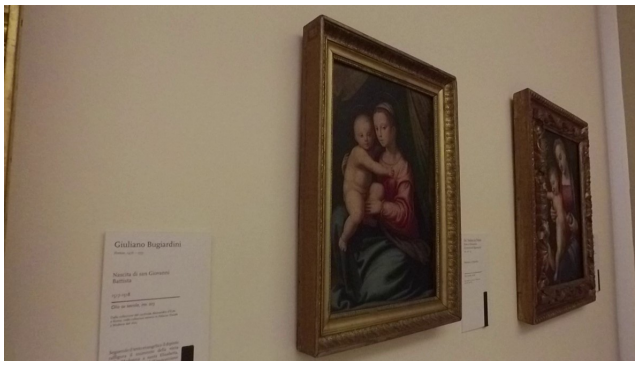

Figure 3. The input image of the pipeline.

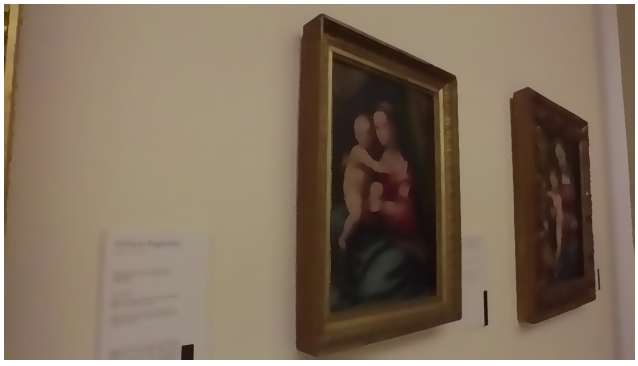

Figure 4. Result of the mean-shift-segmentation on the input image.

close enough to the color of one of its neighbors that already belong to the connected component.

I assign the connected components the value 255 (white) and set all remaining pixels to 0 (black). In this way, for each Flood Fill operation performed, a mask is obtained in which we can find regions of 2 possible colors, white or black. This process is repeated on the pixels of the image to identify the mask in which the white region is the largest and I assume that this region is the wall and that the black regions are the paintings.

The assumption for which the largest region is considered to be the wall, although proving valid in most cases, presents problems when the image contains a very large or very close painting. In this case, the assumption is no longer valid as the largest region will be the one associated with the painting, which will be white, while the wall will be colored black.

Since this problem occurs right at the beginning of the pipeline, it would cause a chain of incorrect predictions. For this reason, I found a solution to solve the problem. Before proceeding with its description, it is necessary to have subjected the image to a series of other operations.

First, the colors of the mask are inverted, assigning 255 to the pixels having value 0 and vice versa. The result is a mask in which the wall is black and the paintings white, called the *wall mask*. To remove any noise present inside the mask, it is first eroded and then dilated equally.

Erosion involves moving a kernel over the pixels of a binary image. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel are 1, otherwise, it is eroded (made to zero).

Dilation is the opposite of erosion. Dilation involves moving a kernel (a matrix of a given size) over the pixels of a binary image. When the kernel is centered on a pixel with a value of 0 and some of its pixels are on pixels with a value of 1, the center pixel is given a value of 1.

The result of the flooding operations and the wall mask obtained after inversion, erosion, and dilatation can be seen in Figure 5.

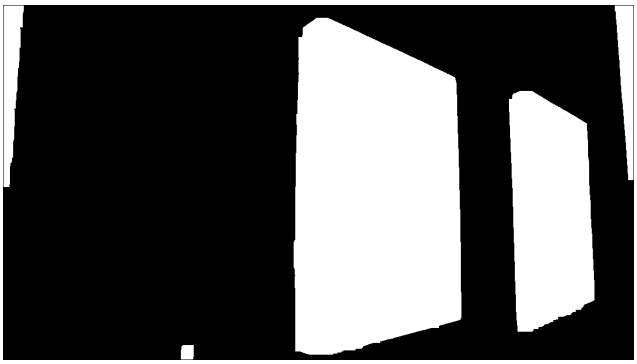

Figure 5. Mask showing the result of the flood fill operation and the wall mask obtained after inversion, erosion and dilatation.

3.2.2 Connected Component Analysis

Figure 5 shows that not all the white components are paintings, for example even the plaques containing the description of the painting are considered to be paintings themselves. Also, any portions of the ceiling or floor in the image could be considered paintings.

To solve these problems I introduced a selection phase, consisting of some criteria that each component must respect to be considered a painting:

- should not span the entire image.
- their bounding boxes should have a minimum area of 150x150 pixels. This value allowed me to eliminate small objects such as a descriptive plate or small fire extinguishers.
- the area of the painting must occupy at least 60% of its bounding box. This means that the paintings are also assumed to be somewhat rectangular. This assumption helps to eliminate parts of the wall and ceiling that may be in the input image.

If some paintings do not fall under these criteria, they are discarded. This could occur, for example, for very small paintings or pictures of paintings that have been taken very far away from the painting.

To determine whether or not a component meets these criteria, the contours of all the white objects in the wall mask are found. For each of them, I create a rotated rectangle of the minimum area enclosing the contours point set. The area of the contours is considered as the area of the paintings.

At this point, we have a list of contours, each associated with a painting.

3.2.3 Refine the Wall Mask

Here comes my solution to the problem that occurs when flood fill considers the wall as paintings and vice versa.

The solution I have adopted can be summarized as follows:

1. I consider the wall mask and apply a one-pixel white border along the outline of the mask.
2. Calculate again all contours of the white components.
3. I get two sets of contours, the first associated with the wall mask without a white border, the second associated with the wall mask having the white border.
4. I consider the set of contours with the largest number of elements. In case they have equal size, I choose the second set.

5. If the selected contour set is the first, then I perform the contour selection phase as described above. If the set of contours selected is the second, then I am in the situation where the flood fill has identified the paintings as a wall and vice versa. Then I invert the wall mask again, to bring it back to the correct situation in which the paintings are white and the wall black, and perform the contour selection phase, adding a new control at the end. It consists of eliminating all the contours enclosing other contours within them.

To understand why my solution works, let's consider the possible situations that can occur:

- The wall mask found with the flood fill is wrong and the addition of the white border determines the identification of several contours equal to the case of wall mask without a white border. We are in case the flood fill worked wrong because if the wall (background of the paintings) had been correctly black, we would certainly have found at least one more contour, the white one added to the wall mask. This situation also occurs when the incorrectly identified paintings (i.e. black regions in the wall mask) are in direct contact with the edges of the image. My solution is also able to manage this case because the addition of the white border separates the paintings from the edge and allows me to correctly identify their contours.

- The wall mask found with the flood fill is wrong and the addition of the white border determines the identification of a greater number of contours than in the case of a wall mask without a white border. This situation occurs when the flood fill function worked incorrectly and we have paintings (in this case black regions of the mask) that touch the edges of the image. Exactly as said in the previous case, my solution can detect this error and correct it.

- The wall mask found with the flood fill is correct and the addition of the white border determines the identification of a single new contour around the whole image. The number of contours will be one greater than in the case of a wall mask without the white border. This is not a problem because this contour will be eliminated in the phase of contours selection, as large as the whole image. Excluding the border contour, the other contours identified within the image (those of the paintings) will remain unchanged.

- The wall mask found with the flood fill is correct and the addition of the white border decreases the number of contours found. We are in the case in which the paintings (in this case white regions of the mask) are only partially inside the image and/or touch the edge of the same. In this case, I correctly consider the contours found with the wall mask without a white border, because in higher numbers.

Figure 6. Original image generating flood fill error (top-left). Wrong result of the flood fill with wall and painting inverted (top-left). Wrong wall mask obtained from flood fill result (bottom-left). Correct wall mask obtained after applying my solution (bottom-right).

An example of an incorrect wall mask where the color of the wall and painting are inverted, and the correct version obtained applying my solution can be seen in Figure 6.

The additional control, which eliminates all the contours that enclose other contours within them, is necessary to prevent that the addition of the white border combined with disturbing objects in the image (e.g. a railing protecting a painting) can generate unwanted contours that contain one or more paintings inside. I am assuming that it is not possible to have a painting within another painting, a condition that can almost always be considered verified, and that in particular, it is with regard to the videos and images in the database.

3.2.4 Painting Segmentation

At this point, I obtained a list of the contours of the paintings that are inside the image, including their relative frames.

To segment the original image, I just need to generate a completely black image with the same resolution as the input image and draw on it only the contours that have passed all the previous tests, filling them in white. The paintings, including their frames, will be the only white components of the image.

The tests carried out have shown that this method is also capable of identifying and segmenting any statues present in the image, if sufficiently large.

The result of the segmentation can be observed in Figure 7.

The Painting Detection and Segmentation task, therefore, appears to have been completed. In reality, the result

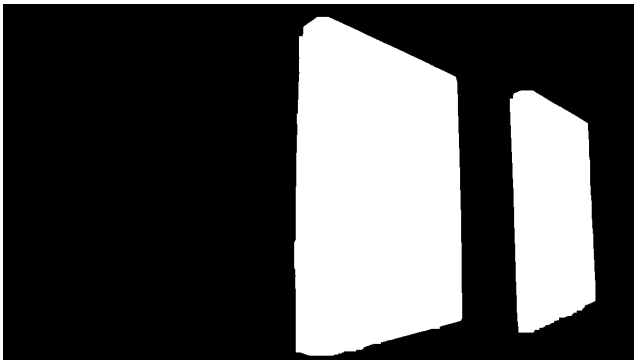

Figure 7. Segmented version of the input image.

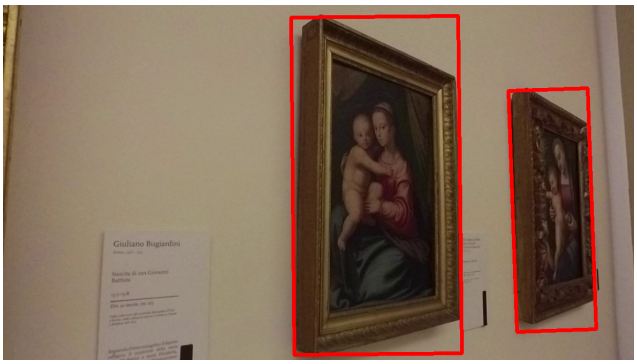

Figure 8. Image showing the rectangular bounding boxes of the paintings.

thus obtained is limited to drawing rectangular bounding boxes around the outline of the paintings, as shown in Figure 8.

The result that I would like to achieve is to build a Region of Interest (ROI) that fits the contour of the painting as best as possible, which due to distortions related for example to perspective, may not be exactly rectangular, especially when positioned not exactly in front of the device that took the photo or recorded the video.

To improve the detection result, a series of additional steps and transformations are necessary, which are also propaedeutics to the execution of the next task, the Painting Rectification.

3.3. Painting Rectification

3.3.1 Find Corners

The starting point is the previously constructed segmented image, which will now be our mask, and a list of the contours of the paintings detected in the image, along with their bounding boxes.

At this point, the previously identified paintings are considered also including the frame. It is, therefore, necessary to erode the mask, to remove a large part of the frame, and isolate only the painting. After erosion, the white components of the image are the paintings, without the frame.

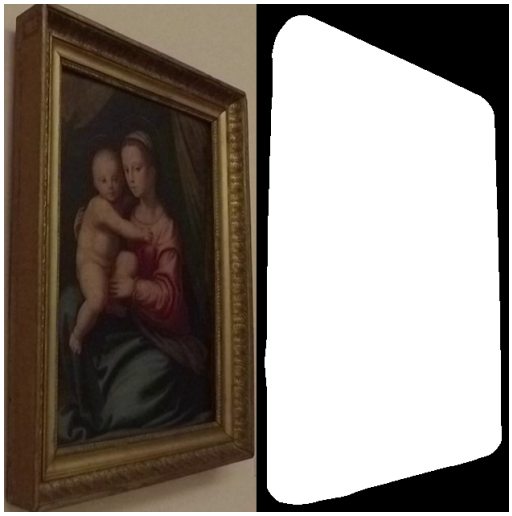

Figure 9. Sub-image (left) and sub-mask (right) of the painting component after erosion and smoothing.

Let's now consider each of these components (the detected paintings) individually. To isolate a component just consider only the portion of the original image and of the mask enclosed by the bounding box of the component itself, obtaining a sub-image and a sub-mask respectively.

I now describe the sequence of operations necessary to find the corners of the painting and apply the affine transformation that allows to rectify the painting and be ready for the next phase, the Painting Retrieval.

First, I identify the lines that represent the edges of the painting components. To do this, a Median Filter is applied to the mask in order to smooth the components' outline. The median filter runs through each pixel of an image and replaces its value with the median of its neighboring pixels (located in a square neighborhood around the evaluated pixel).

The result of the erosion and smoothing operations can be seen in Figure 9.

In order to apply the Hough Line Transform to detect straight lines representing the edges of the painting, we need to build an image in which only the edges of the painting are present. To do this, I run the Canny Edge Detection [5] algorithm on the smoothed sub-mask to obtain an edge image.

Canny edge detection examines the rate of change of brightness values in pixels in versions of the image to which a 5x5 Gaussian filter has been applied to remove the noise. Gaussian Filter involves setting the value of each pixel on a weighted average of the values of the neighboring pixel. In Canny Edge Detection, each pixel is checked to see how much the brightness changes from side to side for many ori-

entations and in many Gaussian filtered versions of the image. Since the edges can be represented as a gradual change in brightness over many pixels, Canny uses the local maximum.

On the edges of the image is applied the Hough Transform [2] to detect straight lines. Lines can be represented as an orthogonal distance to the origin and an angle of the line with respect to an axis. In Hough Lines, an "accumulator" of "cells" is created that represents different combinations of distance and angle. For each point on the edge of the image, all cells representing a line that crosses that point are incremented. Then the local maximum of the accumulator is sought.

From the lines obtained, a mask is created starting from a black image having the same size as the sub mask. Each line found is drawn in white and with a certain thickness on this mask, with a length sufficient to cross the whole mask. This operation brings with it another advantage. The thickness with which the lines are drawn allows me to remove other possible frame pixels from the painting component and to merge multiple lines that represent the same edge of the painting.

The result of the edge detection operations and the mask obtained from the Hough lines can be seen in Figure 10.

The mask built using the Hough lines allows us to isolate the painting component with a few simple operations. First of all, find all the contours inside the mask. Of these, I consider the contour that encloses the largest area, which represents the painting I am looking for. I draw this outline on a black mask of the same size as the sub-mask and fill it in white. So I isolated the painting component.

The painting component mask is used to identify the corners necessary to transform the detected painting before comparing it with the images in the database.

The program uses the Shi-Tomasi Corner Detector [27]. It shows better results compared to Harris Corner Detector [11] and allows me to identify the strongest N corners in the image. In my case, I placed $N = 4$, as interested in finding the four corners of a painting. I am assuming that the paintings are all somewhat rectangular and therefore it is always possible to obtain four corners. In reality, paintings can have more disparate shapes and have several corners greater than or less than 4.

To solve this problem I perform a check phase on the corners found by the Shi-Tomasi Corner Detector. These are considered valid only if they meet the following conditions:

- the corners detected must be 4
- the area of the polygon having as vertices the identified corners must be greater than or equal to a certain threshold obtained as a percentage of the area of the contour of the painting component.

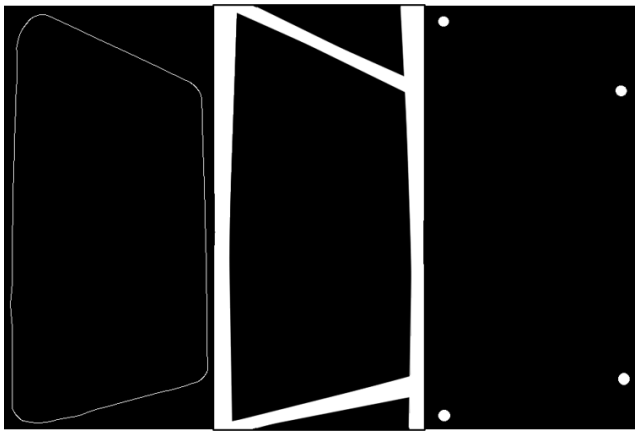

Figure 10. Result of Canny Edge detection (left), the mask obtained from Hough Lines (middle) and the corners found using the Shi-Tomasi Corner Detector (right).

If the above conditions are not respected, the four vertices (top-left, top-right, bottom-right, bottom-left) of the sub-image are considered as corners of the painting component.

This allows me to manage without errors even non-rectangular images. The drawback of this solution is that, if the previous conditions are not respected, after setting the vertices of the sub-image as corners, the next phase of rectification will leave the painting components unchanged.

Figure 10 shows a mask in which the correctly identified corners have been drawn.

3.3.2 Rectify Painting

We have all the necessary ingredients to be able to rectify the identified painting.

To compare the features of detected painting with those of the paintings in the database, it is advisable to carry out an Affine Transformation that uses the previously found corners.

This Affine Transformation is done by translating every pixel in an image to a new location. The transformation is defined by a matrix multiplication, that can be found when the translation of some points is known. So to obtain this matrix, we need to know a set of source points and a set of destination points.

The source points are the previously found corners. As for the destination points, these are determined in the following way:

1. The corners are sorted according to the top-left, top-right, bottom-right, and bottom-left order.
2. The width w of the new image is calculated as the maximum distance between bottom-right and bottom-

Figure 11. Original painting sub-image (left), its rectified version (middle) and the rectified image with brightness and contrast auto-adjusted (right).

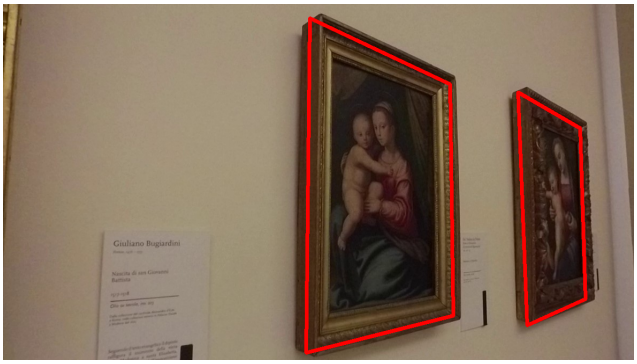

Figure 12. ROI of the detected paintings obtained using the paintings corners.

left x-coordinates or the top-right and top-left x-coordinates.

3. The height h of the new image is calculated as the maximum distance between the top-right and bottom-right y-coordinates or the top-left and bottom-left y-coordinates.

4. Now that dimensions of the new image are known, the program constructs the set of destination points (dst_points) to obtain a top-down view of the image, again specifying points in the top-left, top-right, bottom-right, and bottom-left order:

$$dst\_points = [[0, 0], [w, 0], [w, h], [0, h]] \quad (1)$$

Figure 11 shows a comparison between the original image of the painting and its rectified version.

The program accepts as a target for the affine transformation, not only a set of points but also an entire image. In this case, the program translates the corners of the painting component identified to the four vertices of the destination image, always following the top-left, top-right, bottom-right, and bottom-left order. In this way, source and destination images have the same dimensions.

This additional functionality is useful for performing the next task, the Painting Retrieval.

3.3.3 Improve Paintings ROI

Before proceeding to the next task, let's briefly consider the Painting Detection task again. As previously said, the bounding box that is associated with each detected painting is represented so far by a rectangle. Now, however, I have the ingredient needed to build ROIs that best fit the outline of the painting, respecting the perspective. This ingredient is precisely the corners. By drawing a straight line that connects the various corners, the program builds a polygon that constitutes the ROI I was looking for. In Figure 12 you can see the result obtained, and in particular, you can appreciate how it is better than the previous result based on rectangular bounding boxes and was shown in Figure 8.

3.4. Painting Retieval

At this stage, we have a rectified version of a painting detected within the original image. All that remains is to compare it with all the paintings in the database to find the one that generates the best match.

Lighting problems, light reflection, image exposure, and image saturation due to the exposure of paintings in exhibitions, however, could negatively influence the chances of correctly identifying a painting among those present in the database.

For these reasons, before using ORB to match the paintings, it is advisable to try to improve the brightness and contrast of the image.

To do this, the program is able to automatically adjust the brightness and construction of the image. Brightness and contrast are linear operator with parameter α and β:

$$g(x, y) = \alpha * f(x, y) + \beta \quad (2)$$

The question is: How to automatically calculate the value of α and β?

To do this, we can look at the histogram of the image. Automatic brightness and contrast optimization calculates α and β so that the output range is [0...255].

We calculate the grayscale histogram of the image and use it to calculate a cumulative distribution, necessary to determine where the color frequency is less than some threshold value (typically 1%) and cut the right and left sides of the histogram. This gives us our minimum (min_gray) and maximum (max_gray) grayscale ranges.

To calculate α, we take the minimum and maximum grayscale range after clipping and divide it from our desired

output range of 255:

$$\alpha = 255/(max\_gray - min\_gray) \tag{3}$$

To calculate β, we plug it into the formula 2 where $g(x, y) = 0$ and $f(x, y) = min\_gray$. After solving we obtain:

$$\beta = -min\_gray * \alpha \tag{4}$$

The result of the brightness and contrast auto-adjusting process can be seen in Figure 11.

Once the auto-adjusted version of the rectified image has been obtained, the matching with the database is carried out. The program uses ORB to locate the keypoints in the paintings and calculates how many matches the auto-adjusted painting has produced with each painting in the database.

The key points are calculated for each pair of images being compared and the program checks which keypoints are shared between the two images.

To reduce the number of false positives, i.e. matches with incorrect paintings, the program does not just select the painting from the database that produced the highest number of matches, but determines the best match in the following way:

1. For each painting in the database, calculate the matches with the painting component that is being analyzed. The matches obtained are sorted in ascending order based on the distance (the first are the best matches). Of these, it considers the first N and calculates the average value, i.e. the average distance. If no match has been recorded, a value is returned which acts as a flag and which allows us to understand that there was no match (i.e. a very high value).

2. Create a list with the average distance recorded for each painting in the database and order it in ascending order.

3. If the first value of the list, the lower one, is the flag value, then it means that there was no match with the database.

4. If the first value of the list is different from the flag value then this match is considered valid only if the ratio between the first value of the list and the second value of the list is lower than a certain threshold. That is, I want the first value, the one that determines which painting of the database is assigned to the painting component, is sufficiently smaller than the second value in the list. If this is verified then I can consider the first match of the list sufficiently robust and reliable, otherwise, I drop it.

If ORB has not produced any match with any of the paintings in the database for a given detected painting, the program also offers the possibility to perform a histogram comparison for each painting saved in the database, to identify which painting has the histogram more similar to that of the detected painting. The painting with the best matching histogram is chosen as a painting to classify the detected painting

Histograms are made for the Hue and Saturation of each image. This involves making 'bins' for each possible Hue or Saturation value in the image, counting up the number of pixels in the image that have that Hue or Saturation value. The images' histograms are then compared to see which of the saved painting images has the most similar histogram to the one of the located painting.

To compare two histograms (H_1 and H_2), I use the histogram Intersection as a metric ($d(H_1, H_2)$) to express how well both histograms match:

$$d(H_1, H_2) = \sum_I min(H_1(I), H_2(I)) \tag{5}$$

After histogram matching, the match that produced the highest intersection value is considered the best match.

The result of the Painting Retrieval task can be seen in Figure 14.

Suppose we were able to find a match with the database. We can now access all the information associated with the painting such as title, author, filename, and room in which it is located.

3.5. People Detection

This task is performed independently from the previous ones also because it uses a different approach. It uses the real-time object detection YOLOv3 to identify any people present in the image.

YOLOv3 outputs a list of bounding boxes each of which is associated with a person identified in the image. However, we are inside a museum or art gallery, many paintings portray people, so it often happens that YOLO identifies the subjects of the paintings as people. This, although semantically correct, is not the result I want to achieve.

To solve this problem, I consider it unacceptable and consequently rejected all bounding boxes which overlap more than a fixed threshold with any one of the bounding boxes of the detected paintings.

Figure 13 shows an example of the result of the People Detection operation performed on an image. It should be noted that, although the person covers a portion of the painting with his body, both the painting and the person are correctly identified.

3.6. Painting and People Localization

The last step of the pipeline is the Painting and People Localization task. It aims to determine which room in the

Figure 13. Example of output of the People Detection task.

Figure 14. The output of the pipeline where the names of the paintings found in the Painting Retrieval phase are printed above the paintings' ROI. In the lower-left corner of the image is the number of the room where the paintings are located, found with the Painting and People Localization task.

museum the paintings and people in the input image are located in.

By taking advantage of the information we got to reach this point, this task can be solved with a trivial but effective solution.

The basic idea is: people and paintings in the image are in the same room. So, if I can determine in which room the paintings are located, I have consequently also identified where the people are.

This assumption may not be verified if the image contains portions of different rooms, because perhaps taken by including a door or access to an adjacent room. However, these are particular cases, which do not occur too frequently within the available database.

For each of the paintings recognized in the image, thanks to the information obtained from the database during the Painting Retrieval phase, the program can determine in which room of the museum it is located.

The program creates a list containing the rooms associated with the various paintings identified. Due to incorrect identification of the paintings, the list may contain different values.

The program counts how many times each room number is present in the list and considers the one that has the most occurrences. It is a majority decision. If most of the paintings detected are in a specific room, then it is very likely that it is the correct room.

Figure 14 shows the result of the Painting and People Localization process, which consists of superimposing on the original image the information of the room where paintings and people are located.

4. Results

The tests carried out have shown that the program performs well for most of the data in the database. An example of complete output showing the final result produced by the pipeline can be seen in Figure 1.

The results obtained are great, although the system has some difficulties in dealing with particularly tough situations. This is due to the fact that common image processing techniques suffer from some problems (lighting, scale changes, distortion, etc.) which become even more relevant in the case of moving cameras, which in turn introduce further undesirable effects, e.g. blur, noise, motion.

In addition, the exhibition of paintings in museums and art galleries, adds problems such as light reflection, image exposure, and image saturation, and involves the presence of people who overlap, to a greater or lesser extent, with the paintings.

Although the problem of brightness and contrast has been solved using auto-adjusting techniques, some problems are instead related to the assumptions made during the implementation of the various tasks.

For example, the program may work not correctly when the frames of the paintings overlap, the paintings are too small or taken from afar, the frame is particularly wide. The latter case could adversely affect the result as the erosion phase could not be able to remove enough frame, which could compromise the features matching or histogram comparison phase, making the result less accurate.

However, the program can correctly manage situations in which the painting is very close or is very large and occupies the largest portion of the image (i.e. the flood fill operation mistake the wall and paintings in the Painting Detection phase) and the situation in which YOLO identifies as persons the subjects portrayed within the paintings.

An important factor to highlight is that the various functions of the OpenCV library that have been used to implement the various tasks, often require a large number of parameters as input. This overall determines a very large number of possible combinations of these parameters. Each of these combinations determines a different set-up of the program and therefore different final results. Again, numerous

tests were carried out to find the right trade-off, that is the combination that produced the best results in the various presentable situations.

This, at the same time, opens the way to many other tests that can still be done to improve the results obtained with this work.

5. Conclusions

This work shows how it is possible to complete the Painting Detection, Retrieval, and Localization tasks without the use of deep learning techniques, but through operations and transformations of image processing and analysis.

I was able to face many of the challenges presented to me, such as the problem of incorrect flooding that recognized the wall and paintings in an inverted way, the problem of people detection that identified the subjects within the paintings, or the contrast and lighting problems of the images.

This work presents great results but also opens the way to a series of new possibilities, alternative solutions and different combinations of the operations carried out in the proposed pipeline, which bode well in the possibility that future works will be able to obtain an improvement in results and performance.

References

[1] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 510–517. Ieee, 2012.

[2] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. In *Readings in computer vision*, pages 714–725. Elsevier, 1987.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[4] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[6] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[7] Mahmood Fathy and Mohammed Yakoob Siyal. An image detection technique based on morphological edge detection and background differencing for real-time traffic analysis. *Pattern Recognition Letters*, 16(12):1321–1330, 1995.

[8] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[10] NC Hambly, MJ Irwin, and HT MacGillivray. The supercosmos sky survey—ii. image detection, parametrization, classification and photometry. *Monthly Notices of the Royal Astronomical Society*, 326(4):1295–1314, 2001.

[11] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[12] Yiyu Hong and Jongweon Kim. Art painting detection and identification based on deep learning and image local features. *Multimedia Tools and Applications*, 78(6):6513–6528, 2019.

[13] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[14] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016.

[15] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. Ieee, 2011.

[16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[18] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[19] Niki Martinel, Christian Micheloni, and Gian Luca Foresti. Robust painting recognition and registration for mobile augmented reality. *IEEE Signal Processing Letters*, 20(11):1022–1025, 2013.

[20] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM journal on imaging sciences*, 2(2):438–469, 2009.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[22] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[23] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region

proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[25] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[26] Yong Rui, Thomas S Huang, and Shih-Fu Chang. Image retrieval: Current techniques, promising directions, and open issues. *Journal of visual communication and image representation*, 10(1):39–62, 1999.

[27] Jianbo Shi et al. Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pages 593–600. IEEE, 1994.

[28] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.

[29] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.