ARTISTO or ArtDet or ArtVisual or 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

Thsi work proposes a method to locate and recognize paintings and people in a museum or art gallery. For this purpose, we created a Python program that is able to locate and recognize paintings and people present in a video or single image. For the part relating to the paintings, we used the OpenCV library, while to carry out the people detection operation we used YOLO, 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. Detect and analysing 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 localising 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 we propose avoids this problem by submitting the input image through a processing pipeline which, using the OpenCV [4] library, performs a series of operations and transformations that produce the following results:

- Painting Detection: detects all paintingds in the image.
- 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 paintings DB, containing a list of the paintings in the museum or gallery with related information, such as title of the painting, author, room in which the painting is located. We used ORB [25] as feature detector, to find keypoints and execute matching between an input image and the various database images.
- People Detection: detect people in the input using YOLOv3 [23], a state-of-the-art real-time object detection system and pre-trained weights.
- People and Painting Localization: locates paitning and people using information, 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 [8, 24] works

have unified the localization and classification phases in order 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.

We decided to use a CNN-based object detector only to perform the people detection task. In particular, we have 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, among the various image local features proposed, we have 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 in low-power devices.

Here we have brief introductions 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 use FAST to find keypoints, then apply Harris corner measure to find top N points among them. It also use 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 our 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: 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, that is, when the input is a video, the solution we adopted is to consider each frame of the video as an image to submit to our processing pipeline.

The processing 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. In fact, it requires that for each painting present in an input image or video frames, 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 we faced concerns the different resolution of the videos, which are provided in the formats: HD (1280x720), full-HD (1920x1080) and 4K (3840 x 2160).

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

Our solution was to perform as a first operation of our processing pipeline a resize of all the inputs to the lower resolution, the HD one.

Subsequently, the resized images continue in the pipeline and are subjected to the various processing described below.

The information found (bounding boxes and information about paintings and room) is then subjected to

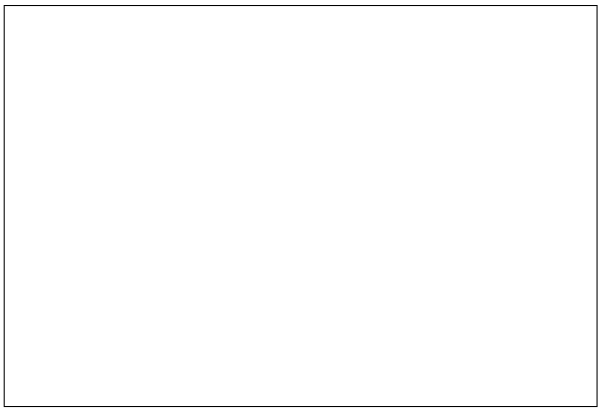

Figure 1. 'TODO', the painting we want to locate, rectify and recognize

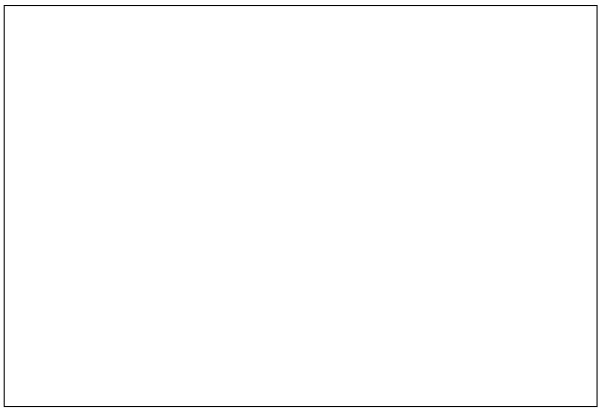

Figure 2. The input image of our processing pipeline.

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 us to maintain the original resolution and improve performance as the functions of the processing pipeline are performed on tensors of a smaller size (e.g. a ninth of the input size in the case of 4K images or videos).

Having obtained from our processing pipeline the various processed images, these 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

3.2.1 Obtain a Wall Mask

To aid the description of the processing of locating, rectifing and recognising a painting in an image, we suppose we want to recognize the painting 'TODO', represented in Figure 1, inside the image present in Figure 2.

All'interno di un museo o galleria d'arte i dipinti sono oggetti appesi alle pareti. Quest'ultime, sono assunte essere di un singolo colore uniforme, come di solito avviene nei musei e gallerie d'arte al fine di evitare di distrarre i visitatori dalle opere d'arte.

Sulla base di questa assunzione, la prima operazione da compiere per individuare e segmentare i dipinti è quella di distinguere quali porzioni dell'immagine non sono muro. Ciò avviene eseguendo l'operazione di Mean Shift Segmentation sull'immagine, la quale clusters nearby pixels with similar pixel values and sets them all to have the value of the local maximum of pixel value. Il risultato è quello di avere i pixel raggruppati per colore e location, come si evince dall'immagine in

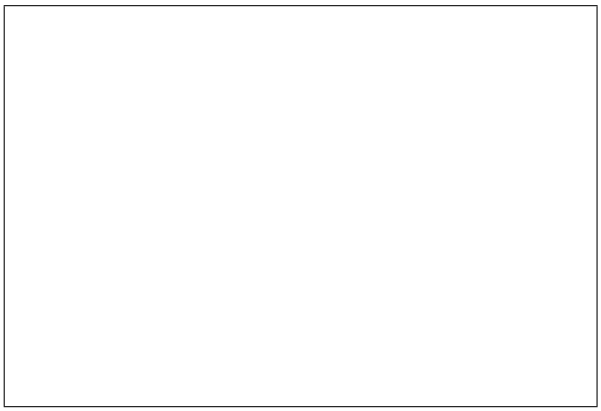

Figure 3. Result of the mean-shift-segmentation on the input image.

Figure 3

Sull'immagine risultante si esegue un'operazione chiamata Flood Fill, la quale assegna lo stesso colore a componenti connessi.

The connectivity is determined by the color/brightness closeness of the neighbor pixels. That is, to be added to the connected component, a color/brightness of the pixel should be close enough to color/brightness of one of its neighbors that already belong to the connected component.

Assegnamo ai componenti connessi il valore 255 (bianco) e poniamo a 0 (nero) tutti i restanti pixel. In questo modo ad ogni perazione di Flood Fill si ottiene una maschera con due segmenti, uno bianco e uno nero. Si ripete questo procedimento sui pixel dell'immagine al fine di individuare la maschera in cui il segmento bianco è quello più esteso, assumimao che tale segmento sia il muro e che le componenti nere siano i dipinti.

Il risultato dell'operazioni di flooding è osservabile in Figura 4

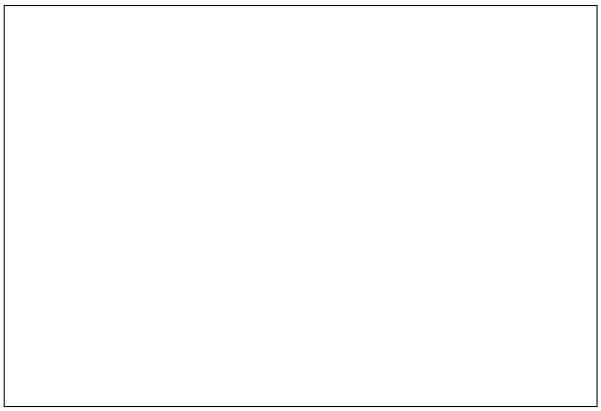

Figure 4. Mask showing the wall in white and the painting in black.

L'assunzione per cui si consiera essere muro il segmento più esteso, pur rivelandosi valida nella maggiorn parte dei casi, presenta dei problemi quando l'immagine ritrae un dipinto molto grande o un dipinto molto da vicino. In questo caso, l'assunzione non è più valida in quanto il segmento più esteso sarà quello associato al dipinto, che sarà bianco, mentre il muro verrà colorato di nero.

Ovviamente, un problema del genera all'inizio della pipeline causerebbe un catene di predizioni errate. Per questo motivo, abbiamo trovato una soluzione che seppur molto empirica si è rivelata essere valida. Prima di procedere con la sua descrizione è necessario aver sottoposto l'immagine ad una serie di altre operazioni.

Per rimuovere eventuale rumore presente all'interno della maschera, essa viene prima dilatata e poi erosa in egual misura. Da notare che We have not yet inverted the mask, therefore making dilation at this stage is equivalent to erosion and vice versa.

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 centre pixel is given a value of 1.

Erosion is the opposite of dilation. 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 is 1, otherwise it is eroded (made to zero).

A questo punto, si invertono i colori della maschera dilatata, assegnando 255 ai pixel aventi valore 0 e viceversa. Si ottiene così una maschera in cui il muro è nero e i dipinti bianchi, chiamata wall mask.

3.2.2 Connected Component Analysis

Come si può notare dall'immagine, non tutti i componenti neri sono dei dipinti, ad esempio anche le targe che contengono la descrizione del dipinto vengono considerate esse stesse dei dipinti. Anche evenuali porzioni di soffitto o pavimento presenti nell'immagine potrebbero essere considerati dei dipinti.

Per risolvere tali problemi abbiamo introdotto dei criteri che ciascun componente deve rispettare per poter essere considerato un dipinto quali:

- it should not span the entire image
- their bounding boxes devono avere un'area minima, che abbiamo fissato essere di 150x150 pixels. Questo valore ci ha permesso di eliminare oggetti piccoli come le targe descrittive o piccoli estintori.
- l'area del dipinto deve occupare almeno il 60% della sua bounding boxe. That 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 not be categorised as paintings. Questo potrebbe verificarsi ad esempio per very small paintings or pictures of paintings that have been taken very far away from the painting.

Per stabilire se un componente rispetta o meno tali criteri, si trovano i contorni di tutti gli oggetti bianci presenti nella maschera invertita precedentemente ricavata. Per ognuno di essi si ricava a rotated rectangle of the minimum area enclosing the contours 2D point set. Si considera come area del dipinto, l'area del relativo contorno, e si scartano tutte i contorni che non rispettano i precedenti criteri.

A questo punto abbiamo una lista di contorni ognuno associato ad un dipinto.

3.2.3 Refine the Wall Mask

Qui entra in gioco la mia soluzione empirica al problema che si ha quando il flooding considera il muro come dipinti e viceversa.

La soluzione che ho adottato può essere così schematizzata:

1. Considero nuovamente la wall mask invertita.
2. Applico un bordo bianco di un pixel lungo il contorno dell'immagine.
3. Calcolo nuovamente tutti i contorni dei componenti bianchi

4. Ho due set di contorni, il primo associato alla wall mask invertita il secondo associato alla wall mask invertita con un contorno bianco.

5. Considero il set di contorni con il maggior numero di elementi, a parità scelgo il secondo.

6. Se il set di contorni selezionato è il primo, allora eseguo la fase di rifinitura dei contorni come descritto precedentemente. Se il set di contorni selezionato è il secondo, allora sono nell'ipotesi in cui il flood fill ha identificato come muro i dipinti e viceversa. Quindi inverto nuovamente la wall mask, in modo da riportarmi alla situazione corretta in cui i dipinti sono bianchi e il muero nero, ed eseguo la fase di rifinitura dei contorni aggiungendo però un nuovo controllo alla fine. Esso consiste nell'eliminare tutti i contorni che hanno al loro interno altri contorni.

Per capire perchè la mia soluzione funziona, consideriamo le possibili situazioni che si possono avere:

- la wall mask trovata con il flood fill è sbagliata e l'aggiunta del bordo bianco determina l'individuazione di un numero di contorni uguale al caso senza bordo. Sono necessariamente nel caso in cui la flood fill ha sbagliato, perchè se il muro (background dei dipinti) fosse stato nero avrei sicuramente trovato un contorno in più, quello da me aggiunto alla wall mask. Questa situazione si ha anche quando il dipinto individuato in maniera errata (ossia di colore nero nella wall mask) è a diretto contatto con i bordi dell'immagine. La mia soluzione è in grado di gestire anche questo caso, perchè l'aggiunta del contorno bianco separa il dipinto dal bordo e mi permette di individuare correttamente il suo contorno.

- la wall mask trovata con il flood fill è sbagliata e l'aggiunta del bordo bianco determina l'individuazione di un numero di contorni maggiore rispetto al caso senza bordo. Questa situazione si verifica quando la funzione di flood fill ha funzionato in modo errato e ho dei dipinti (porzioni dell'immagine di colore nero) che toccano il contorno dell'immagine. Esattamente come detto per il caso precedente la mia soluzione è in grado di individuare questo errore e correggerlo.

- la wall mask trovata con il flood fill è corretta e l'aggiunta del bordo bianco determina l'individuazione di un solo nuovo contorno attorno a tutta l'immagine. Il numero di contorni sarà di uno maggiore rispetto al caso di wall mask senza il bordo bianco. Ciò non rappresenta un problema perchè tale contorno verrà eliminato nella fase di refinizione dei contorni, in quanto grande quanto tutta l'immagine. Escluso il contorno di bordo, gli altri contorni individuati all'interno dell'immagine (quelli dei dipinti) resteranno invariati.

- la wall mask trovata con il flood fill è corretta e l'aggiunta del bordo bianco fa diminuire il numero di contorni trovati. Siamo nel caso in cui i dipinti (porzioni dell'immagine di colore bianco) si trovano solo in parte all'interno dell'immagine e/o toccano il bordo della stessa. In questo caso, si considerano i contorni trovati con la wall mask senza bordo bianco, perchè in numero superiore.

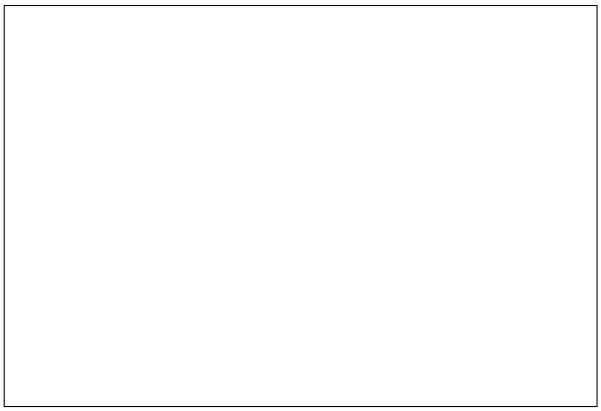

Figure 5. Wall mask errata (a sinistra) e wall corretta applicando la mia soluzione (a destra).

Un esempio di wall mark invertita e corretta grazie alla mia soluzione è osservabile in figura 5

Il controllo aggiuntivo, che elimina tutti i contorni che hanno altri contorni al loro interno, serve per evitare che l'aggiunta del bordo bianco unito a degli oggetti di disturbo presenti nell'immagine (e.g. una ringhiera messa a protezione di un dipinto) possano generare dei contorni indesiderati che contengano al loro interno uno o più dipinti. Ovviamente, sto supponendo che non sia possibile avere un dipinto all'interno di un altro dipinto, condizione che può considerarsi quasi sempre verificata e che in particolare lo è per quanto riguarda i video e le immagini che ci sono state fornite.

3.2.4 Painting Segmentation

A questo punto, ho ottenuto una lista dei contorni dei dipinti, compresi delle relative cornici, che sono presenti all'interno dell'immagine.

Generare una versione segmentata dell'immagine originale è adesso molto semplice. Genero un'immagine

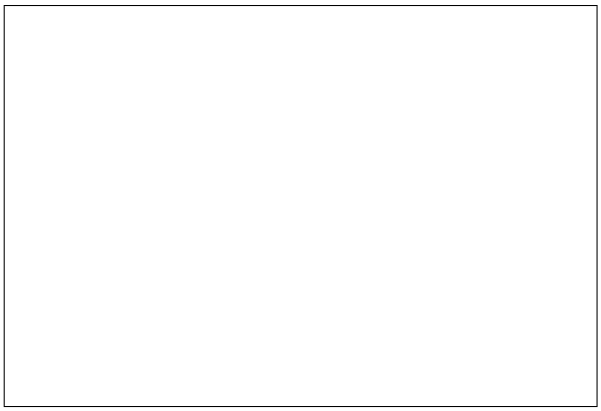

Figure 6. Segmented version of the input image.

Figure 7. Image showing the rectangular bounding box of the paintings.

completamente nera avente la stessa risoluzione dell'immagine originale e vi disegno sopra solo i contorni che hanno superato tutti i precedenti tests, riempendoli di bianco. I dipinti, incluse le relative cornici, saranno i soli componenti bianchi dell'immagine. I test effettuati hanno dimostrato che tale metodo è in grado anche di indivisuare e segmentare eventuali statue presenti nell'immagine, se sufficientemente grandi.

Il risultato della segmentazione è osservabile in Figura 6

Il task di Painting detection and Segmentation sembra quindi completato. In realtà il risultato così ottenuto si limita a disegnare delle bounding boxes rettangolari attorno al contorno dei dipinti, come mostrato in Figura 7.

Il risultato che invece vorrei ottenere è quello di avere una Region of Interest (ROI) che fitti al meglio possibile il contorno del dipinto, il quale a causa di distorsioni legate ad esempio alla prospettiva può non essere esattamente rettangolare, soprattutto quando posizionato non esattamente di fronte al dispositivo che

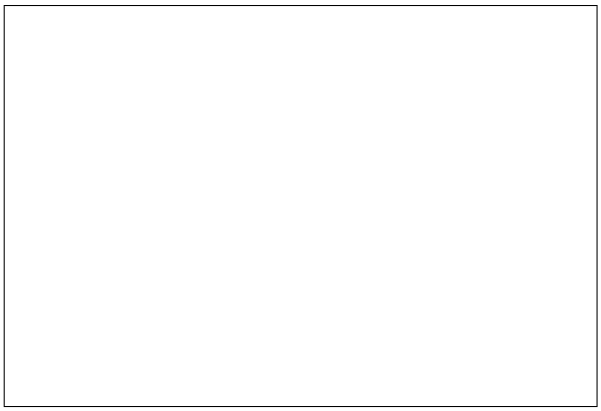

Figure 8. Sub-image (left) and sub-mask (right) of a painting component.

ha scattato la foto o registrato il video.

Per migliore il risultato della detection. Sono necessari una serie di passi e trasformazioni aggiuntive, che tra l'altro sono propedeutiche all'esecuzione del task successivo, quello della Painting Rectification.

3.3. Painting Rectification

3.3.1 Find Corners

La nostra base di partenza è l'immagine segmentata precedentemente costruita , che adesso sarà la nostra maschera, e una lista dei contorni dei dipinti individuati nell'immagine con relative bounding boxes.

In questa fase, i dipinti precedentemente individuati sono considerati comprendendo anche la cornice. Si renden dunque necessaria l'erosione della maschera, che ci consente di rimuovere buona parte della cornice e isolare solo il dipinto. Adesso le componenti bianche dell'immagine sono proprio i dipinti senza la cornice.

Da adesso in poi l'oggetto della processing pipeline sarà ognuna di queste componenti, ossia i vari dipinti individuati. Per isolare una componente basta considerare solo la porzione dell'immagine originale e della maschera che si trovano all'interno della bounding box del componente stesso, ottenendo così quelle che chiameremo sum-image e sub-mask rispettivamente. Un esempio, è presente in Figura 8

Descriviamo adesso la sequenza di operazioni necessarie per trovare i corners del dipinto e applicare la trasformazione affine che ci consente di rettificare il dipinto ed essere pronti alla fase successiva, quella del painting Retrieval.

Partiamo con l'individuare le linee che rappresentano i bordi del painting components. Per far ciò, si applica un Median Filter ai componenti al fine di smussarne gli outline. The median filter run through each pixel of an image and replace its value with the median

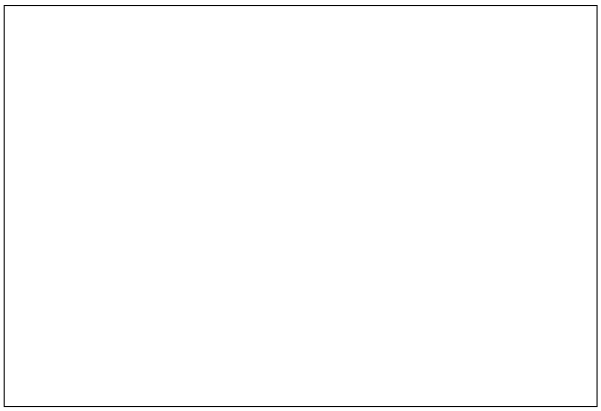

Figure 9. Result of component erosion e smoothing using a Median Filter.

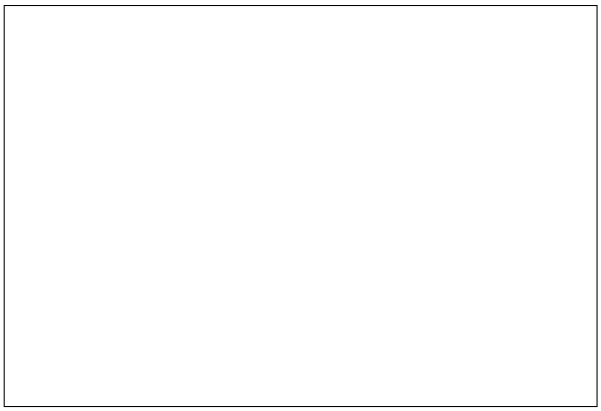

Figure 10. Result of Canny Edge detection (left) and the mask obtained from Hough Lines (right).

of its neighboring pixels (located in a square neighborhood around the evaluated pixel).

Il risultato delle operazioni di ersoione e smoothing è osservabile in Figura 9

Al fine di applicare la Hough Line Transform to detect straight lines che rappresentano i bordi del dipinto, abbiamo bisogno di costruire un'immagine in cui siano presenti solo i bordi.

Per fare ciò, si esegue il Canny Edge Detection [5] algorithm sul componente per ottenere un 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 orientations 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 local maxima

Sulla edge image si applica la Hough Tranform [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 are sought.

Dalle linee ottenute, si crea una maschera che isola il painting component racchiuso tra le linee. Tale maschera è ottenuta partendo da un'immagine nera avente la stessa dimensione della sub mask. Ciascuna linea trovata viene diegnata di colore bianco su questa maschera, con una lunghezza tale da attraversare tutta la maschera. Questa operazione porta con se un'altro vantaggio, che consiste nel rimuovere altri possibili frane pixels nel painting component e permette di accorpare molteplici linee che rappresentano lo stesso edge del dipinto.

Il risultato delle operazioni di edge detection e la maschera ricavata dalle Hough lines è osservabile in figura 10.

La maschera costruita partendo dalle linee di Hough ci consente di isolare il painting component con poche semplici operazioni. Per prima cosa di trovano tutti i contorni all'interno della maschera. Di questi si considera il contorno che racchiude l'area più ampia, la quale rappresenta il dipinto che stiamo cercando.

Disegno questo contorno su una maschera nera della stessa dimensione della sub-mask e lo riempio di bianco. Ho così isolato il mio painting component.

La maschera del painting component è usata per individuare i corners necessari a trasformare il detected painting prima di compararlo con le immagini presenti nel database.

Il programma utilizza lo Shi-Tomasi Corner Detector [27]. Esso shows better results compared to Harris Corner Detector [11] e consente di individuare gli N corners più forti presenti nell'immagine. Nel mio caso, ho posto $N = 4$, in quanto interessato a trovare i quattro angoli di un dipinto. Ovviamente, sto supponendo che i dipinti siano tutti in qualche modo rettangolari e che dunque sia sempre possibile ottenere quattro corner. Nella realtà, i dipinti possono presentare forme più disparate e avere un numero di corner maggiore o inferiore a 4.

Per risolvere questo problema eseguo una fase di check sui corner trovati dallo Shi-Tomasi Corner Detector. Infatti, questi vengono considerati validi solo

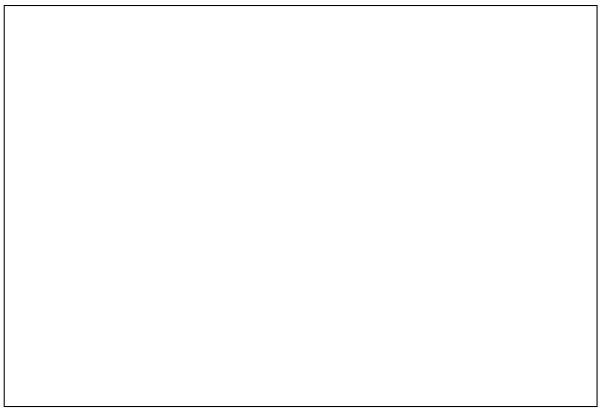

Figure 11. Corners found using the Shi-Tomasi Corner Detector.

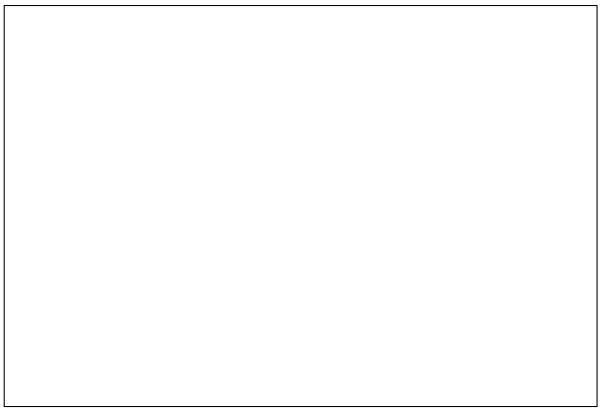

Figure 12. Original painting image (left) and its reftified version (right).

se rispettano le seguenti condizioni:

- sono esattamente 4
- l'area del poligono avente come vertici i corner individuati deve essere maggiore o uguale ad una certa soglia ottenuta come percentuale dell'area del contorno del painting component presente nella sub-image.

Se non vengono rispettare le seguenti condizioni, si considerano come corners i quantro vertici (top-left, top-right, bottom-right, bottom-left) della sub-image.

Questo mi permette di gestire senza errori anche immagini non squarish. Il drawback di questa soluzione è che, nal caso in cui non vengano rispettate le precedenti condizioni, impostando come corner i vertici dell'immagine, la fase successiva di rectification, lascerà invariati i painting components.

La Figura 11 mostra una maschera in cui sono strati disegnati i corners correttamente individuati.

3.3.2 Rectify Painting

Abbiamo tutti gli ingredienti necessari per poter effettuare la rectification del dipinto individuato.

Infatti, per poter comparare le features del detected painting con quelle dei dipinti che costituiscono il db, è opportuno effettuare una trasformazione affine che utilizza i corners precedentemente trovati.

An 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 are known. Quindi, per ricavare questa matrice, abbiamo bisogno di conoscere un set di punti sorgente e un set di punti destinazione.

I punti sorgente sono sempre i corners precedentemente trovati. Per quanto riguarda i punti destinazione, questi vengono determinati nel seguente modo:

1. Si ordinano i corner secondo the top-left, top-right, bottom-right, and bottom-left order.
2. Si calcola the width w of the new image, which will be the maximum distance between bottom-right and bottom-left x-coordiates or the top-right and top-left x-coordinates.
3. Si calcola the height h of the new image, which will be the maximum distance between the top-right and bottom-right y-coordinates or the top-left and bottom-left y-coordinates.
4. Now that we have the dimensions of the new image, construct 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:

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

La Figura 12 mostra un cinfronto tra l'immagine originale e la sua vrsione rettificata.

Il programma consente di accettare come destinazione della trasformazione affine, non solo un set di punti ma anche un'intera immagine. In questo caso il programma sa che i corner del painting component individuato devono essere traslati nei quattro vertici dell'immagine destinazione, sempre seguendo the top-left, top-right, bottom-right, and bottom-left order.

Questa ulteriore funzionalità è utile per eseguire il task successivo quello di Painting Retireval.

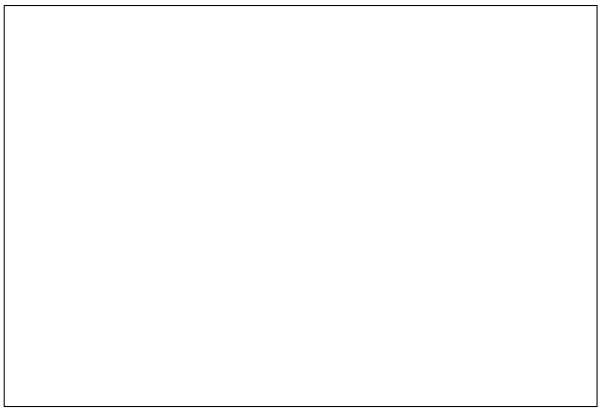

Figure 13. ROI of the detected points obtained using paintings corners.

3.3.3 Improve Paintings ROI

Prima di procedere con task successivo, riprendiamo per un attimo in considerazione il task di painting detection. Come precedentemente detto, la bounding boxe che è associata ad ogni dipinto detected è rappresentata fino a questo momento da un rettangolo. Adesso però abbiamo l'ingrediente che ci serve per costruire delle ROI che fittino al meglio possibile il contorno del dipinto e la sua prospettiva. Tale ingrediente sono appunto i corner. Tracciando una retta che congiunge i vari corner, siamo infatti in grado di costruire un poligono che costituisce proprio la ROI che stavamo cercando. In Figura 13 si può ammirare il risultato ottenuto, e in particolare si può apprezzare come esso sia migliore del precedente risultato basato su bounding box rettangolari.

3.4. Painting Retieval

Arrivati a questa fase, siamo in possesso di una versione reffiticata di un dipinto detected all'interno dell'immagine originale. Non rimane che confrontarlo con tutti i dipinti presenti nel database al fine di trovare quello che genera il match migliore.

Problemi di illuminazione,riflessione delle luci, image exposure and image saturation dovuti all'esposizone di dipinti in esibizioni e mostre, potrebbero tuttavia influenzare negativamente le possibilità di individuare correttamente un dipinto tra quelli presenti nel database.

Per queste ragioni, prima di procedere all'utilizzo di ORB per effettuare il matching dei dipinti, è opportuno cercare di migliorare la luminosità e il contrasto dell'immagine.

Per fare ciò, il programma aggiusta automaticamente la luminosità e il constrato dell'immagine.

Brightness and contrast is linear operator with parameter α and β:

$$g(x, y) = \alpha * f(x, y) + \beta \tag{2}$$

The question is: How to automatically calculate 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 color frequency is less than some threshold value (tipically 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 obatin:

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

Ottenuta la versione auto-adjusted dell'immagine rettificata si procede alla fase di matching con il database. Il programma utilizza ORB per individuare i keypoints nei dipinti, e calcola quanti matches il auto-adjusted painting ha con ciascun dipinto del database.

I key point vengono calcolati per ogni coppia di immagini che si sta confrontando e il programma verifica quali keypoints sono condivisi tra le due immagini.

Al fine di ridurre il numero di falsi positivi, ossia di match con dipinti errati, il programma non si limita a selezionare il dipinto del database che ha prodotto il numero più alto di match, ma determina il match migliore nel modo seguente:

- per ogni dipinto del databse, calcola i match con il painting component che si sta anlizzando. I matches ottenuti vengono ordinati in ordine crescente sulla base della distanza (i primi sono i match migliori). Di questi considera i primi N (parametro settabile) e ne calcola il valore medio. Se non si è registrato nessun match si ritorna un valore che funge da flag e che permette di capire che non c'è stato nessun match (i.e. un valore molto elevato).

- Si crea una lista con la distanza media registrata per ogni dipinto del database e la si ordina in modo crescente.

- Se il primo valore della lista, quello inferiore, è il valore flag, allora significa che non c'è stato alcun match con il database.
- Se il primo valore della lista è diverso dal valore flag allora tale match si considera valido solo se il rapporto tra il il primo valore della lista e il secondo valore della lista è inferiore ad una certa soglia. Ossia, voglio che il primo valore, quello che mi determina quale dipinto del db assegnare, sia sufficientemente più piccolo del secondo valore della lista. Se ciò è verificato allora posso considerare il primo match della lista sufficientemente robusto e affidabile, altrimenti vuorrà dirre che il match è debole e poco affidabile e non lo considero.

Nel caso in cui ORB non abbia prodotto alcun match con nessuno dei dipinti presenti nel databse per un determinato detected painting, il programma offre anche la possibilità effettuare un histogram comparison per ogni dipinto salvato nel database, in modo da individuare quale dipinto ha l'istogramma più simile a quello del detected painting. Il dipinto con il migliore 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 amount 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 the most similar histogram to the histogram of the located painting.

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

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

In quest'ultimo caso si considera come match migliore quello che ha dati il valore di intersezione più elevato.

Supponiamo di essere stati in grado di aver individuato un match col database. Adessiamo possiamo accedere accedere a tutte le informazioni asoziate al dipinto quali titolo, autore, filename e stanza in cui si trova.

Tutte informazioni molto utili che ci serviranno per eseguire il task di Painting and People Detection.

3.5. People Detection

Questo task viene eseguito in modo del tutto dipendente dai precedenti anche perchè utilizza un approccio profondamente diverso. Esso infatti sfrutta il real-time object detection YOLOv3 per individuare le eventuali persone presenti all'interno dell'immagine.

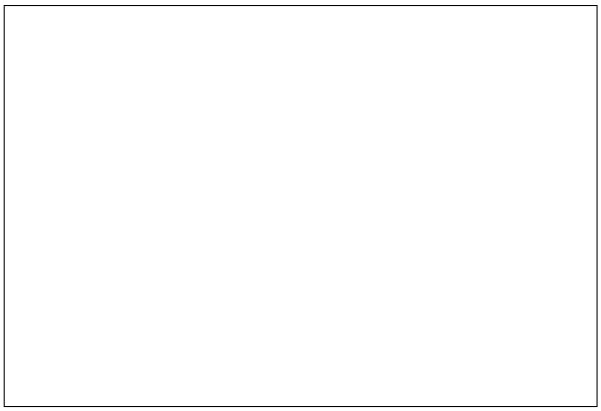

Figure 14. Example of output of the people detection task.

YOLOv3 produce in output una lista di bounding boxes ognuna delle quali associata ad una persona individuata nell'immagine.

Siamo però all'interno di un museo o galleria d'arte, molti dipinti ritraggono delle persone, quindi accade spesso che YOLO individui delle persone che in realta sono i soggetti dei dipinti.

Per risolvere questo problema, considero come non accetabili, e di conseguenza scarto, tutte le baounding boxes which overlap more than a fixed threshold with any one of bounding boxes of the detected paintings.

La Figure 14 mostra un esempio di risultato dell'operazione di People detection eseguita su un'immagine. Da notare come nonostante copra con il proprio copro una porzione del dipinto, sia il dipinto che la persona vengono individuate correttamente.

3.6. Painting and People Localization

L'ultimo tassello della processing pipeline, il task di Painting and People Localization, si pone come obiettivo quello di determinare in quale stanza del museo si trovano i dipinti e le persone presenti nell'immagine di input.

Sfruttando le informazioni che abbiamo ricavato per giungere a questo punto, tale task può essere risolto con una soluzione trivial ma efficace.

L'idea di base è, le persone e i dipinti presenti nell'immagine si trovano nella stessa camera. Quindi, se sono in grado di determinare in quale camera si trovano i dipinti presenti nell'immagine allora ho di conseguenza individuato anche dove si trovano le persone.

Questa assunzione, può non essere verificata nel caso in cui l'immagine ritragga porzioni di camere differenti, perchè magari scattata includendo una porta o un ac-

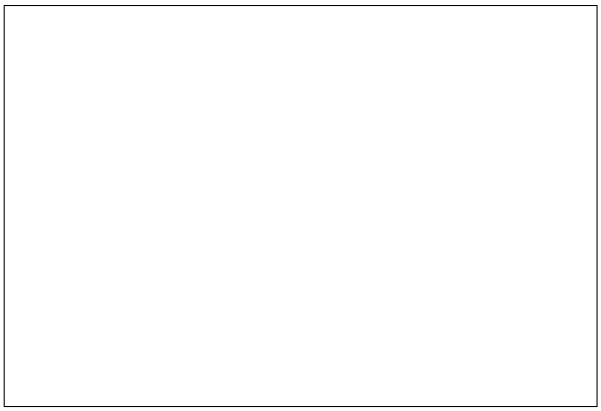

Figure 15. Example of output of the Painting and People Localization task.

cesso ad una camera adiacente. Si tratta però di casi particolari, che non si verificano troppo di frequente all'interno del dataset a disposizone.

Per ognuno dei dipinti riconosciuti nell'immagine, grazie alle informazioni ricavate dal db nella fase di painting retireval, il programma è in grado di determinare in quel stanza del museo si trova.

Il programma crea una lista delle camere di tutti i dipinti individuati all'interno dell'immagine. A causa di errori di identificazione dei dipinti, può accadere che la lista contenga valori differenti.

Si conta quante volte ogni numero di camera eè presente nella lista e si considera quello che è èresenta il maggior numero di occorrenze. Si tratta di una decisione presa a maggioranza. Se la maggior parte dei dipinti individuati si trovano in una determinata camera, allora è molto probabile che sia quella la camera corretta.

La Figura 15 mostra il risultato del processo di Painting and People Localization, che consiste nello stampare in sovraimpressione l'informazione della camera dove dipinti e persone si trovano.

4. Results

Il programma proposto in questo lavoro constente di elaborare video e foto al fine di produrre

Mostra i risultati complessivi e dove l'approccio fallisce quando non vengono rispettate le premesse e i presupposti fatti precedentemente (vedi ad esempio il flooding della parete). Inserire tempistiche e che precisa che YOLO gira su GPU.

Un fattore molto importante da evidenziare riguarda la suscettibilità del risultato ottenuto al variare dei parametri con cui sono state invocate le funzioni della libreria OpenCV. In alcuni casi, modificare anche leggermente un paramtro, porta ad una netta variazione dell'output. Ciò ha richiesto molti test al fine di trovare la combinazione di valori dei parametri che mi permettesse di ottenre i risultati migliori. Questo però apre la strada a molti altri test che possono ancora essere fatti al fine di migliorare i risultati ottenuti con questo lavoro.

5. Conclusions

Questo lavoro ha mostrato come sia possibile portare a termine i task di painting Detection, Retrieval and Localization senza l'utilizzo di tecniche di deep learning, ma attraverso delle operazioni e trasformazioni di image processing and analysis. I risultati ottenuti sono degni di nota seppur il sistema mostri il fianco in situazioni particolarmente ostiche.

Ciò è legato al fatto che le tecniche comuni di image processing soffrono di alcuni problemi (illuminazione, scale changes, distorsione, etc.) che diventano ancora più rilevanti nel caso di moving cameras, le quali a loro volta introducono ulteriori effetti indesiderati, e.g. blur, noise, motion. In aggiunta a ciò, l'esposione dei dipinti in esibizioni e mostre aggiunge problemi di riflessione delle luci, image exposure and image saturation, nonchè la presenza di persone che si sovrappongono in misura più o meno rilevante ai dipinti esposti.

Sono riuscito ad affrontare alcuni di questi problemi, come ad esempio il problema del flooding errato che riconosceva in modo invertito il muro e i dipinti, oppure i problemi di contrasto e illuminazione delle immagine, risolti attraverso tecniche di auto-adjusting che mi hanno inoltre permesso di migliorare i risultati di match corretti con il DB.

Questo lavoro presenta buoni risultati ma apre anche la strada a una serie di nuove strade e soluzioni alternative e differenti combinazioni delle operazioni svolte che fanno sperare nella possibilità di poter ottenere un miglioramento dei risultati e delle 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.