VCS 2020: Locate and Recognize Paintings and People

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 [5, 8], recognition [16] and retrieval [23].

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[9]. 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 [3] 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 [22] 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 [20], 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) [7], who had the idea of using a selective search [25] 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 [6, 21] 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 [4, 11, 13, 14, 19, 18, 24, 20] 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 [22, 1, 2, 10, 12, 15, 17] 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 [22].

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. Methods

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

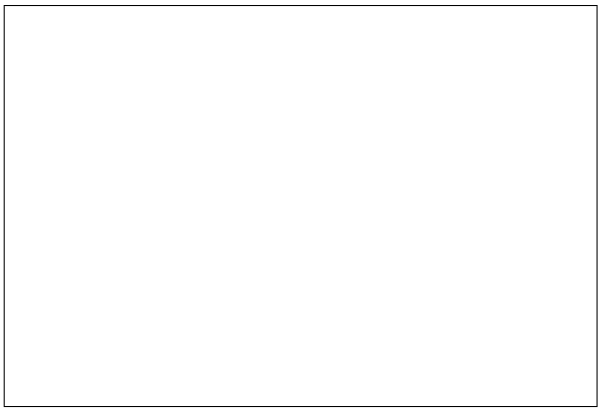

Figure 1. 'TODO', the painting we want to locate, rectify and recognize

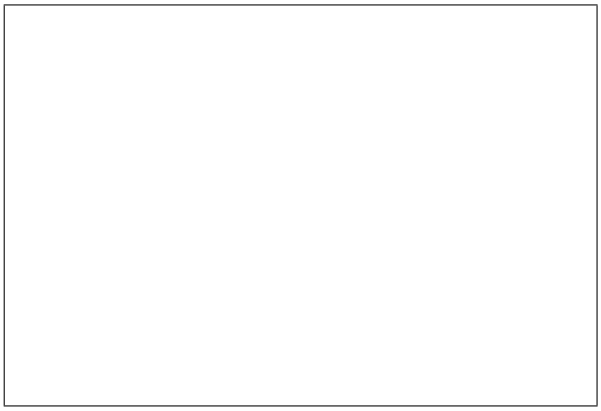

Figure 2. The input image of our processing pipeline.

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

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.

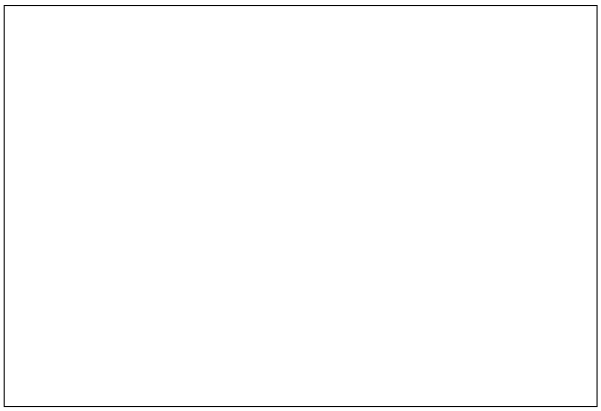

Figure 3. Result of the mean-shift-segmentation on the input image.

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

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

L'assunzione per cui si consiera essere muro il seg-

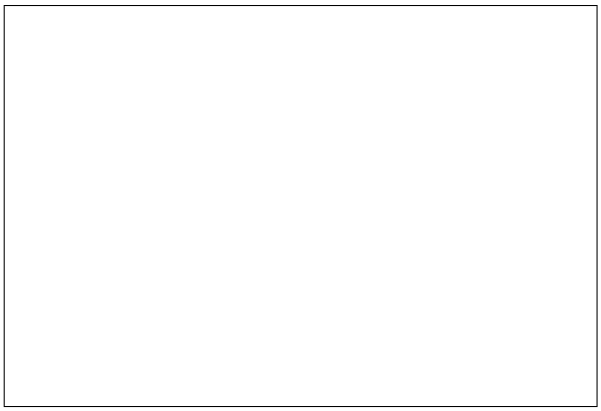

Figure 4. Mask showing the wall in white and the painting in black.

mento 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.

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.

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 cosniderate 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 il rettangolo ad area minima che lo contiene, ossia una bounding box ruotata e con base non parallela alla base dell'immagine. Si scartano tutte i contorni che non rispettano i precedenti criteri.

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

Qui entra in gioco la 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

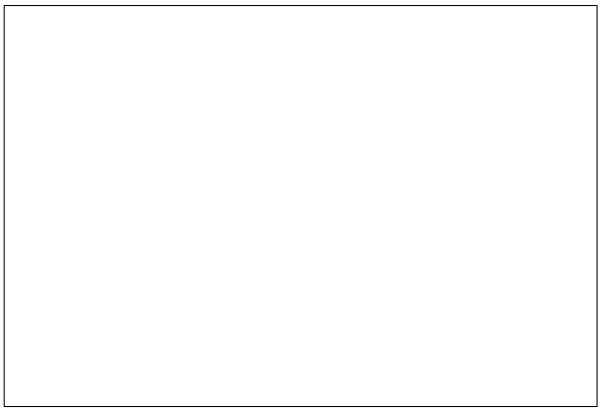

Figure 5. Segmented version of the input image.

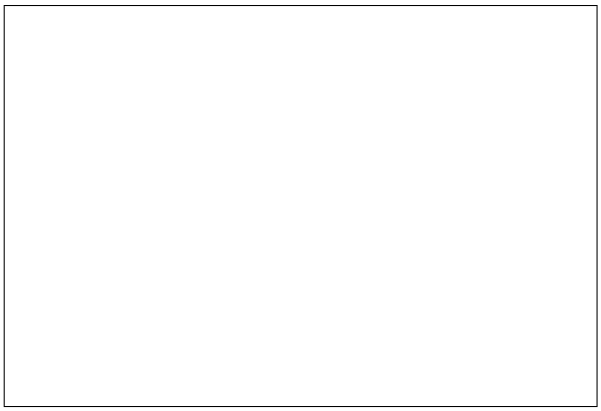

Figure 6. Image showing the rectangular bounding box of the paintings.

cui il flood fill ha identificato come muro i dipinti e viceversa. Quindi inverto nuovamente la wall mask 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 basti considerare che nel caso in cui la wall mask è già corretta (muro bianco e dipinti neri) aggiungere un bordo dello stesso colore del muro non farà altro che lasciare il numero di contours trovati invariato, oppure lo farà diminuire nel caso di dipinti che si trovano solo in parte all'interno dell'immagine e/o toccano il bordo della stessa. Ciò mi permette di capire se flood fill ha funzionato correttamente o nel verso opposto. 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.

A questo punto, ho ottenuto una lista dei contorni dei dipinti all'interno dell'immagine.

Generare una versione segmentata dell'immagine originale è adesso molto semplice. Genero un'immagine 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. Il risultato della segmentazione è osservabile in Figura 5

Il task di Painting detection and Segmentation sembra quindi completato. In realtà il risultato così ottenuto si limita a disegnare dei rettangoli attorno al contorno dei dipinti, come mostrato in Figura 6.

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

– TODO –

3.4. Painting Retieval

– TODO –

3.5. People Detection

– TODO –

3.6. Painting and People Localization

– TODO –

4. Results

– TODO – 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).

5. Conclusions

– TODO –

PROBLEMI: 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.

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

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

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.