Computer Vision
Dr. Mohammadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW7

۱. ابتدا تصاویر را در یک سطر نمایش می‌دهیم. بدین صورت که در یک حلقه‌ی for ابتدا هر تصویر را خوانده، سپس به figure خود اضافه کرده و در انتها تمام تصاویر را نمایش می‌دهیم.

```
# read input victoria images and show them in a row together
fig = plt.figure(figsize=(16, 16))
columns = 7
rows = 1
for i in range(1, 8):
    img = cv2.imread(f'images/victoria{i}.png')
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
    plt.axis(False)
    fig.subplots_adjust(wspace=0.05)
plt.show()
```

سپس به منظور جاگیری تصاویر کنار هم و سایز بزرگ تصاویر، سایز عکس‌ها را کوچک کرده و لیستی از تصاویر می‌سازیم. با استفاده از تابع cv2.stitcher.create() و تابع stitch() یک پانوراما از تصاویر تهیه می‌کنیم.

```
# initialize OpenCV's image sticher object and then perform the image stitching
on input images
# https://www.geeksforgeeks.org/opencv-panorama-stitching/
```

```
images = []
for i in range(1, 8):
    image = cv2.imread(f'images/victoria{i}.png')
    images.append(image)
    images[i - 1] = cv2.resize(images[i - 1], (0,0), fx=0.4, fy=0.4)

stitchy = cv2.Stitcher.create()

(status, stitched) = stitchy.stitch(images)
```

در انتها نیز عکس ترکیب‌شده را نمایش می‌دهیم. (موزائیک تصویر)

```
# show victoria panorama
imgplot = plt.imshow(stitched)
plt.axis(False)
plt.show()
cv2.imwrite("images/stitched.png", stitched)
```

(2)

$$x_2 = x_1 \cos\theta - y_1 \sin\theta$$

$$y_2 = x_1 \sin\theta + y_1 \cos\theta$$

$$\text{Cost} = \sum \left(x_2^n - x_1^n \cos\theta + y_1^n \sin\theta\right)^2 + \left(y_2^n - x_1^n \sin\theta - y_1^n \cos\theta\right)^2$$

$$\frac{d\,\text{Cost}}{d\theta} = \sum 2\left(x_2^n - x_1^n \cos\theta + y_1^n \sin\theta\right)\left(x_1^n \sin\theta + y_1^n \cos\theta\right)$$

$$+ 2\left(y_2^n - x_1^n \sin\theta - y_1^n \cos\theta\right)\left(y_1^n \sin\theta - x_1^n \cos\theta\right) = 0$$

$$\sum 2\Big[x_2^n x_1^n \sin\theta - (x_1^n)^2 \sin\theta\cos\theta + y_1^n x_1^n \sin^2\theta + y_1^n x_2^n \cos\theta +$$

$$- x_1^n y_1^n \cos^2\theta + (y_1^n)^2 \sin\theta\cos\theta + y_2^n y_1^n \sin\theta - y_1^n x_1^n \sin^2\theta$$

$$- (y_1^n)^2 \sin\theta\cos\theta - x_1^n y_2^n \cos\theta + (x_1^n)^2 \sin\theta\cos\theta + y_1^n x_1^n \cos^2\theta\Big] = 0$$

$$\Rightarrow \sum \left(x_2^n x_1^n \sin\theta + x_2^n y_1^n \cos\theta + y_1^n y_2^n \sin\theta - x_1^n y_2^n \cos\theta\right) = 0$$

$$\sum \sin\theta\left(x_2^n x_1^n + y_1^n y_2^n\right) + \cos\theta\left(x_2^n y_1^n - x_1^n y_2^n\right) = 0$$

$$\Rightarrow tg\theta = \frac{-\sum \left(x_2^n y_1^n - x_1^n y_2^n\right)}{\sum \left(x_2^n x_1^n + y_1^n y_2^n\right)}$$

$$\Rightarrow \theta = \text{Arc}\,tg\left(\frac{\sum \left(x_1^n y_2^n - x_2^n y_1^n\right)}{\sum \left(x_2^n x_1^n + y_1^n y_2^n\right)}\right)$$

۳. ابتدا با تابع زیر تصویر را سیاه سفید می‌کنیم.

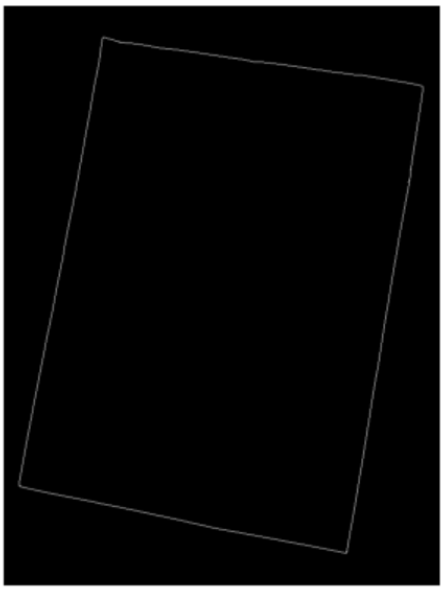
```
def to_grayscale(im):
    # Your code goes here.
    im_gray = cv2.cvtColor(im, cv2.COLOR_RGB2GRAY)
    return im_gray
```

سپس با تابع زیر تصویر را blur می‌کنیم تا فرکانس‌های بالا که موردنیاز نیستند، حذف شوند. بدین منظور از فیلتر گوسی استفاده کردیم.

```
def blur(im):
    # Your code goes here.
    im_blur = cv2.GaussianBlur(im, (5,5),0)
    return im_blur
```

با استفاده از لبه‌یاب canny و انتخاب آستانه‌ی بالا و پایین مناسب، لبه‌های کاغذ در تصویر را پیدا می‌کنیم. بدلیل اینکه در این مرحله می‌خواهیم فقط رئوس کاغذ را پیدا کنیم، بهتر است مقدار آستانه‌ی بالا و پایین را جوری انتخاب کنیم که فقط لبه‌های کاغذ در خروجی باشند.

```
def to_edges(im):
    # Your code goes here.
    im_edges = cv2.Canny(im, 50, 300)
    return im_edges
```



با استفاده از تابع زیر ابتدا threshold مناسب را پیدا کرده، سپس نقاط دور صفحه را استخراج می‌کنیم. به کمک تابع cv.approxPolyDP() شکل کاغذ و نقاط رأس‌های بزرگ‌ترین چهارضلعی موجود را تخمین زده و به لیست رئوسی که داریم اضافه می‌کنیم. در انتها لیست مختصات رئوس کاغذ را به عنوان ریترن می‌کنیم. پس از چاپ کردن این لیست متوجه می‌شویم که بیش از چهار نقطه را به عنوان رأس استخراج کرده است، لذا چون بقیه‌ی نقاط با دقت خوبی به چهار نقطه‌ی اول نزدیک اند، ما هم فقط چهار نقطه‌ی اول این خروجی را به عنوان رئوس کاغذ در نظر می‌گیریم.

```
# https://www.geeksforgeeks.org/find-co-ordinates-of-contours-using-opencv-
python/
def find_vertices(im):
    # Your code goes here.
    vertices = []
    _, threshold = cv2.threshold(im, 110, 255, cv2.THRESH_BINARY)
    contours, _= cv2.findContours(threshold, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for cnt in contours :
        approx = cv2.approxPolyDP(cnt, 0.009 * cv2.arcLength(cnt, True), True)
        n = approx.ravel()
        i = 0

        for j in n :
            if(i % 2 == 0):
                x = n[i]
                y = n[i + 1]
                vertices.append((x, y))
            i = i + 1
    return vertices
```

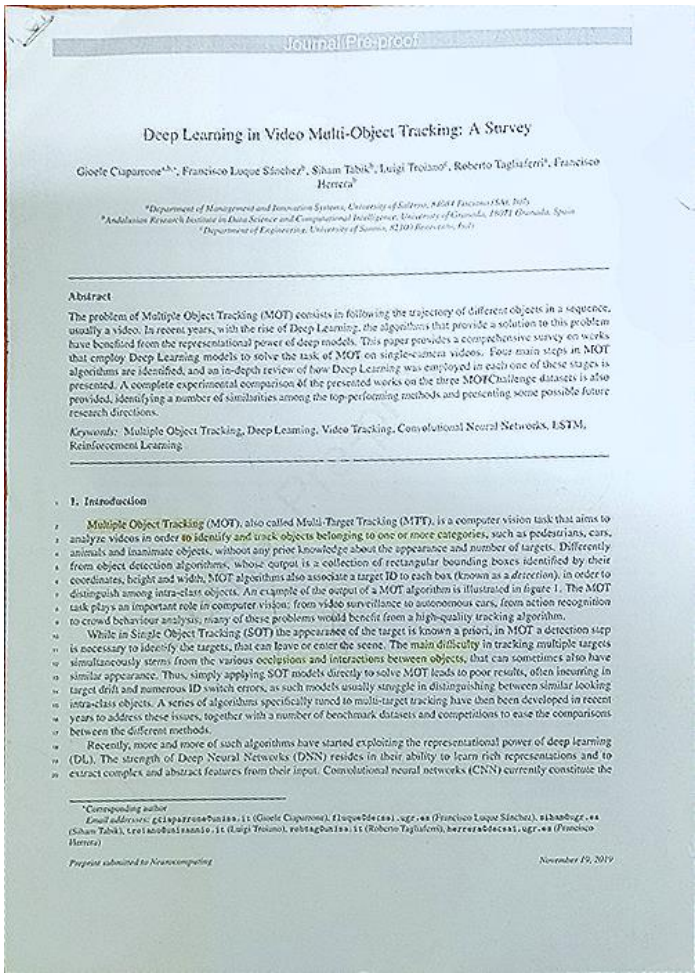
با استفاده از تابع زیر، ابتدا رئوس را با تابع reorder() که از پیش تعریف شده بود، مرتب می‌کنیم. سپس نقاط رئوس جدید را تعریف کرده و مطابق آرایه‌ی قبلی مرتب می‌کنیم. این دو آرایه را به عنوان ورودی به تابع cv2.getPerspectiveTransform() می‌دهیم تا تبدیل مناسب را پیدا کند. دقت کنید که ورودی این تابع باید دو آرایه‌ی np و همچنین به صورت float باشند. سپس تصویر تبدیل یافته را به کمک تابع cv2.wrapPerspective() ریترن می‌کنیم. بدین صورت نقاط موجود در تصویر اولیه که ممکن بود متوازی‌الاضلاع هم باشند، به نقاط یک مستطیل مپ می‌شوند.

```
# https://www.geeksforgeeks.org/perspective-transformation-python-opencv/
def crop_out(im, vertices):
    # Your code goes here.
    vertices = reorder(np.array(vertices[0:4], np.float32))
    target = reorder(np.array([[0, 0], [0, 700], [500, 700], [500, 0]],
np.float32))
    transform = cv2.getPerspectiveTransform(vertices, target)
    return cv2.warpPerspective(im, transform, (500, 700))
```

در مرحله‌ی آخر برای بهبود تصویر می‌توانیم از کرنل sharpening استفاده کرده و لبه‌ها را تیزتر کنیم. سپس تصویر را به فضای HSV برده و پارامتر V را به اندازه‌ای افزایش دهیم. راه حل دیگر برای آن که بیشتر به رنگ اسکنرها نزدیک شویم این است که تصویر را با استفاده از پیدا کردن adaptiveTreshold() باینری کنیم. یعنی جاهای روشن‌تر سفید و جاهای تیره کاملا مشکی شوند. سپس نویز موجود در تصویر را حذف کنیم. با استفاده از clahe هم می‌توان کیفیت عکس را افزایش داد تا خروجی بهتری داشته باشیم.

```
# https://stackoverflow.com/questions/32913157/how-to-get-magic-color-effect-
like-cam-scanner-using-opencv
# https://github.com/harshilp24/document_scanner_python/blob/main/scan
def enhance(im):
    # Your code goes here.
    kernel_sharpening = np.array([[0, -1, 0],
                                  [-1, 5, -1],
                                  [0, -1, 0]])
    im = cv2.filter2D(im, -1, kernel_sharpening)
    value = 50
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value
    final_hsv = cv2.merge((h, s, v))
    im = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    # im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    # thresh = cv2.adaptiveThreshold(im, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 2)
    # denoised = cv2.fastNlMeansDenoising(thresh, 11, 31, 9)
    return im
```

خروجی:



Journal Pre-proof

Deep Learning in Video Multi-Object Tracking: A Survey

Gioele Ciaparrone[a,b,*], Francisco Luque Sánchez[b], Siham Tabik[b], Luigi Troiano[c], Roberto Tagliaferri[a], Francisco Herrera[b]

[a]Department of Management and Innovation Systems, University of Salerno, SBAI Fisciano (SA), Italy
[b]Andalusian Research Institute in Data Science and Computational Intelligence, University of Granada, 18071 Granada, Spain
[c]Department of Engineering, University of Sannio, 82100 Benevento, Italy

Abstract

The problem of Multiple Object Tracking (MOT) consists in following the trajectory of different objects in a sequence, usually a video. In recent years, with the rise of Deep Learning, the algorithms that provide a solution to this problem have benefited from the representational power of deep models. This paper provides a comprehensive survey on works that employ Deep Learning models to solve the task of MOT on single-camera videos. Four main steps in MOT algorithms are identified, and an in-depth review of how Deep Learning was employed in each one of these stages is presented. A complete experimental comparison of the presented works on the three MOTChallenge datasets is also provided, identifying a number of similarities among the top-performing methods and presenting some possible future research directions.

Keywords: Multiple Object Tracking, Deep Learning, Video Tracking, Convolutional Neural Networks, LSTM, Reinforcement Learning

1. Introduction

Multiple Object Tracking (MOT), also called Multi-Target Tracking (MTT), is a computer vision task that aims to analyze videos in order to identify and track objects belonging to one or more categories, such as pedestrians, cars, animals and inanimate objects, without any prior knowledge about the appearance and number of targets. Differently from object detection algorithms, whose output is a collection of rectangular bounding boxes identified by their coordinates, height and width, MOT algorithms also associate a target ID to each box (known as a *detection*), in order to distinguish among intra-class objects. An example of the output of a MOT algorithm is illustrated in figure 1. The MOT task plays an important role in computer vision: from video surveillance to autonomous cars, from action recognition to crowd behaviour analysis, many of these problems would benefit from a high-quality tracking algorithm.

While in Single Object Tracking (SOT) the appearance of the target is known a priori, in MOT a detection step is necessary to identify the targets, that can leave or enter the scene. The main difficulty in tracking multiple targets simultaneously stems from the various occlusions and interactions between objects, that can sometimes also have similar appearance. Thus, simply applying SOT models directly to solve MOT leads to poor results, often incurring in target drift and numerous ID switch errors, as such models usually struggle in distinguishing between similar looking intra-class objects. A series of algorithms specifically tuned to multi-target tracking have then been developed in recent years to address these issues, together with a number of benchmark datasets and competitions to ease the comparisons between the different methods.

Recently, more and more of such algorithms have started exploiting the representational power of deep learning (DL). The strength of Deep Neural Networks (DNN) resides in their ability to learn rich representations and to extract complex and abstract features from their input. Convolutional neural networks (CNN) currently constitute the

*Corresponding author
Email addresses: gciaparrone@unisa.it (Gioele Ciaparrone), fluque@decsai.ugr.es (Francisco Luque Sánchez), stabik@ugr.es (Siham Tabik), troiano@unisannio.it (Luigi Troiano), robtag@unisa.it (Roberto Tagliaferri), herrera@decsai.ugr.es (Francisco Herrera)

Preprint submitted to Neurocomputing *November 19, 2019*

در اینجا کیفیت کم تصویر اصلی نیز در خروجی موثر بوده است.