گزارش تمرین هفتم بینایی ماشین

نام تهیه‌کننده: ملیکا نوبختیان

شماره دانشجویی: ۹۷۵۲۲۰۹۴

نسخه: ۱

۱- سوال اول

در ابتدا تمام تصاویر را خوانده و در کنار هم نمایش می‌دهیم:

```
path = '/content/images'
image_list = glob.glob(path + '/*.png')
number_of_files = len(image_list)
fig=plt.figure(figsize=(10, 10))
rows, cols = 1, number_of_files
im_list = []
for j in range(cols * rows):
  img = cv2.imread(image_list[j])
  im_list.append(img)
  fig.add_subplot(rows, cols, j+1)
  plt.imshow(img)
  plt.axis('off')
plt.show()
```

سپس از لیست تصاویری که داریم استفاده می‌کنیم و sticher را روی آن اعمال می‌کنیم. status code ای که برگردانده می‌شود نشان می‌دهد که آیا عملیات متصل‌کردن تصاویر به یکدیگر موفق بوده‌است یا نه. اگر عددی که برگردانده می‌شود برابر صفر باشد، یعنی نتیجه عملیات موفقیت آمیز بوده‌است. در غیر این صورت به دلایل مختلفی مثل پیدانکردن تعداد کافی keypoint و یا کم بودن تصاویر بوده‌است.

```
# initialize OpenCV's image sticher object and then perform the image stit
ching on input images
stitcher = cv2.Stitcher_create()
(status, stitched) = stitcher.stitch(im_list)
print(status)
if status == 0:
  # display the output stitched image to our screen
  cv2_imshow(stitched)
```

در نهایت نتیجه نهایی به صورت زیر خواهد بود:

منبع:

https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/

۲- سوال دوم

۲۳ تیر پنج شنبه 14 Thu. Jul. 2022 ۱۴ ذی الحجه ۱۴۴۳

$$\theta = \tan^{-1}\left(\frac{x_1 y_2 - x_2 y_1}{x_1 x_2 + y_1 y_2}\right)$$

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

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

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

$$\frac{d\,Cost}{d\theta} = 2\sum \left(x_2 - x_1\cos\theta + y_1\sin\theta\right)\left(x_1\sin\theta + y_1\cos\theta\right) + \left(y_2 - x_1\sin\theta - y_1\cos\theta\right)\left(-x_1\cos\theta + y_1\sin\theta\right) = 0$$

$$x_1 x_2 \sin\theta + x_2 y_1 \cos\theta - x_1^2 \sin\theta\cos\theta - x_1 y_1 \cos^2\theta + x_1 y_1 \sin^2\theta + y_1^2 \sin\theta\cos\theta - x_1 y_2 \cos\theta + y_1 y_2 \sin\theta + x_1^2 \sin\theta\cos\theta - x_1 y_1 \sin^2\theta + x_1 y_1 \cos^2\theta - y_1^2 \sin\theta\cos\theta = 0$$

روز گفت و گو و تعامل سازنده با جهال

۲۴ تیر جمعه 15 Fri. Jul. 2022 ۱۵ ذی الحجه ۱۴۴۳

$$x_1 x_2 \sin\theta + x_2 y_1 \cos\theta - x_1 y_2 \cos\theta + y_1 y_2 \sin\theta = 0$$

$$\div \cos\theta \quad x_1 x_2 \tan\theta + x_2 y_1 - x_1 y_2 + y_1 y_2 \tan\theta = 0$$

$$\tan\theta\left(x_1 x_2 + y_1 y_2\right) = x_1 y_2 - x_2 y_1 \qquad \tan\theta = \frac{x_1 y_2 - x_2 y_1}{x_1 x_2 + y_1 y_2}$$

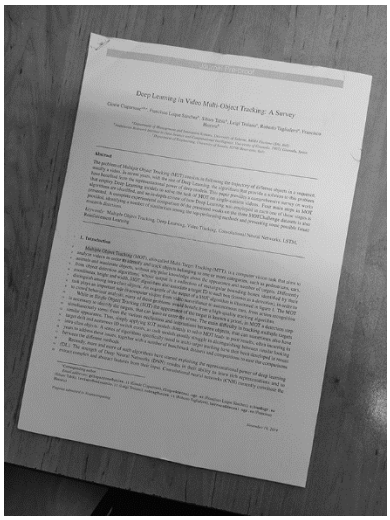
ولادت حضرت امام علی النقی الهادی علیه السلام(۲۱۲ هـ.ق)

تیر: Wed 1, Thu 2, Fri 3, Sat 4, Sun 5, Mon 6, Tue 7, Wed 8, Thu 9, Fri 10, Sat 11, Sun 12, Mon 13, Tue 14, Wed 15, Thu 16, Fri 17, Sat 18, Sun 19, Mon 20, Tue 21, Wed 22, Thu 23, Fri 24, Sat 25, Sun 26, Mon 27, Tue 28, Wed 29, Thu 30, Fri 31

۳- سوال سوم

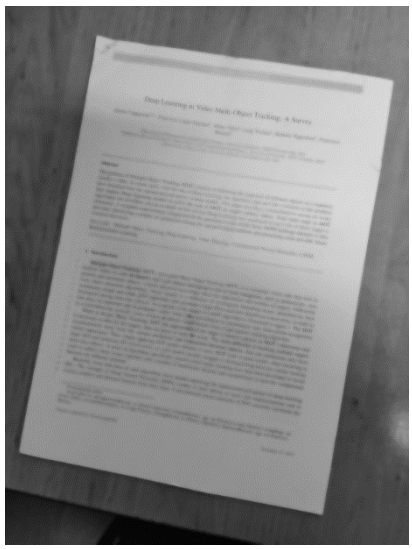
در ابتدا با استفاده از cvtColor تصویر را از حالت BGR به GRAY تغییر می‌دهیم:

```
def to_grayscale(im):
    # Your code goes here.
    return cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
```



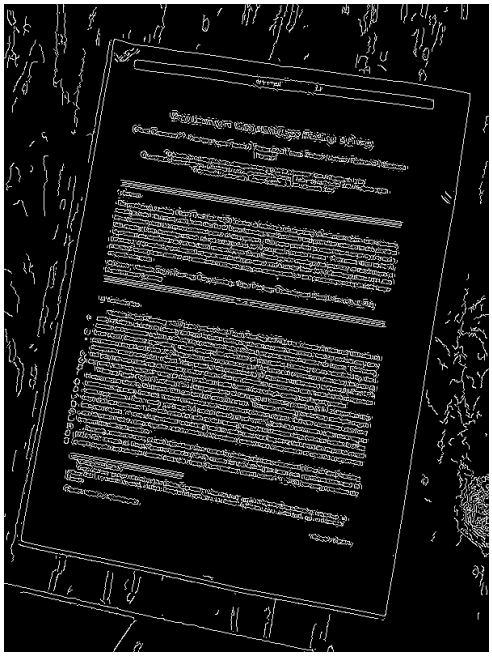
برای محوکردن تصویر از bilateral filter استفاده کردم. مزیت این فیلتر نسبت به فیلترهای دیگر این است که edge ها را بر خلاف فیلترهای میانگین و میانه حفظ می‌کند. پارامترهایی که این تابع می‌گیرد تصویر، d که شعاع همسایگی است، sigmaColor که فیلتر سیگما را در فضای رنگی مشخص می‌کند و هرچه بیشتر باشد رنگ‌های دورتر با هم ترکیب خواهند شد. پارامتر بعدی sigmaSpace است که فیلتر سیگما را در coordinate space نشان می‌دهد. هر چه مقدار آن بیشتر باشد، پیکسل‌های دورتر با هم ترکیب خواهند شد.

```
def blur(im):
    # Your code goes here.
    return cv2.bilateralFilter(im, 9, 75, 75)
```



برای لبه‌یابی هم از canny با threshold های ۱۰ و ۲۰ استفاده کردم:

```
def to_edges(im):
    # Your code goes here.
    return cv2.Canny(im,10,20)
```



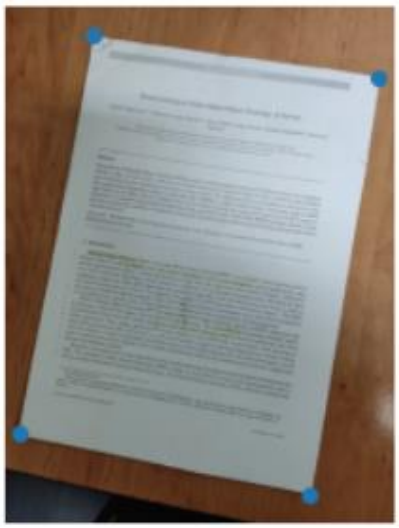
برای تشخیص رئوس در تصویر ابتدا باید contour detection انجام دهیم. ورودی‌های تابع findContours تصویر لبه‌یابی شده، روش contour retrival که ما در اینجا از RETR_TREE استفاده می‌کنیم و پارامتر آخر روش contour approximation است که در اینجا از مدل simple استفاده می‌کنیم.

در گام بعد لازم است contour ای که بزرگ‌ترین مساحت را دارد پیدا کنیم. با استفاده از contourArea مساحت همه contour های به‌دست‌آمده را حساب می‌کنیم و بزرگ‌ترین آن‌ها را پیدا می‌کنیم.

در گام بعد سعی می‌کنیم با approxPolyDP کانتور به دست آمده را تقریب بزنیم و به شکل موردنظر برسیم. یکی از پارامترهای موردنیاز این تابع epsilon است. برای این کار نیاز به arclength کانتور خود داریم که از تابع خود آن استفاده می‌کنیم. پارامتر دوم این تابع را True می‌گذاریم چون مشخص می‌کند که کانتور ما باز یا بسته است. از یک دهم این طول برای epsilon استفاده می‌کنیم. هم چنین در اینجا نیز چون کانتور ما بسته است پارامتر سوم True خواهد بود. در نهایت لیست نقاط چهار راس را بازمی‌گردانیم و نتیجه آن روی تصویر به صورت زیر است:

```
def find_vertices(im):
    # Your code goes here.
    countors = cv2.findContours(im, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE
)[0]
    max_area = 0
    max_countor = None

    for cnt in countors:
        area = cv2.contourArea(cnt)
        if area > max_area:
```

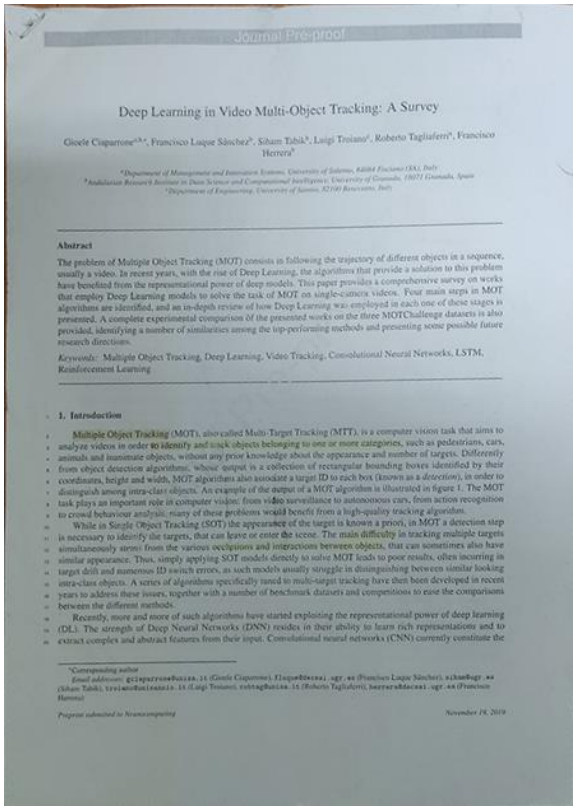
```
            max_area = area
            max_countor = cnt

    epsilon = 0.1 * cv2.arcLength(max_countor,True)
    approx = cv2.approxPolyDP(max_countor, epsilon, True)
    return [[point[0][0], point[0][1]] for point in approx]
```



در مرحله بعد به سراغ نگاشت دورنما می‌رویم. برای اینکه بتوانیم این نگاشت را به دست آوریم نیاز به نقاط راس در تصویر اصلی و نقاط نهایی در مقصد در صورتی که تنها آن شکل را داشته باشیم داریم. در ابتدا باید ترتیب نقاط رئوس را تصحیح کنیم تا به ترتیب چپ بالا، راست بالا، چپ پایین و راست پایین باشد. برای اینکه بتوانیم نقاط نهایی را به دست آوریم نیاز به عرض و طول صفحه داریم. پس با استفاده از نقاط رئوس این مقادیر را به دست آورده و مقادیر مقصد را هم می‌سازیم. حالا می‌توانیم تبدیل را به دست آوریم.

پس از به دست آوردن تبدیل آن را با warpPerspective به تصویر اعمال می‌کنیم. تصویر crop شده به شکل زیر خواهد بود:

```
def crop_out(im, vertices):
    # Your code goes here.
    vertices = [vertices[0], vertices[3], vertices[1], vertices[2]]
    vertices = np.array(vertices, dtype=np.float32)
    width = math.ceil(np.sqrt(np.sum(np.power(vertices[0] - vertices[1], 2
))))
    height = math.ceil(np.sqrt(np.sum(np.power(vertices[0] - vertices[2],
2))))
    target = np.float32([[0, 0], [width, 0],
                         [0, height], [width, height]])
    transform = cv2.getPerspectiveTransform(vertices, target)  # get the t
op or bird eye view effect
    return cv2.warpPerspective(im, transform, (width, height))
```



Journal Pre-proof

Deep Learning in Video Multi-Object Tracking: A Survey

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

[a]Department of Management and Information Systems, University of Salerno, 84084 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 (Giocle Ciaparrone), fluque@decsai.ugr.es (Francisco Luque Sánchez), stabik@ugr.es (Siham Tabik), ltroiano@unisannio.it (Luigi Troiano), robtag@unisa.it (Roberto Tagliaferri), herrera@decsai.ugr.es (Francisco Herrera)

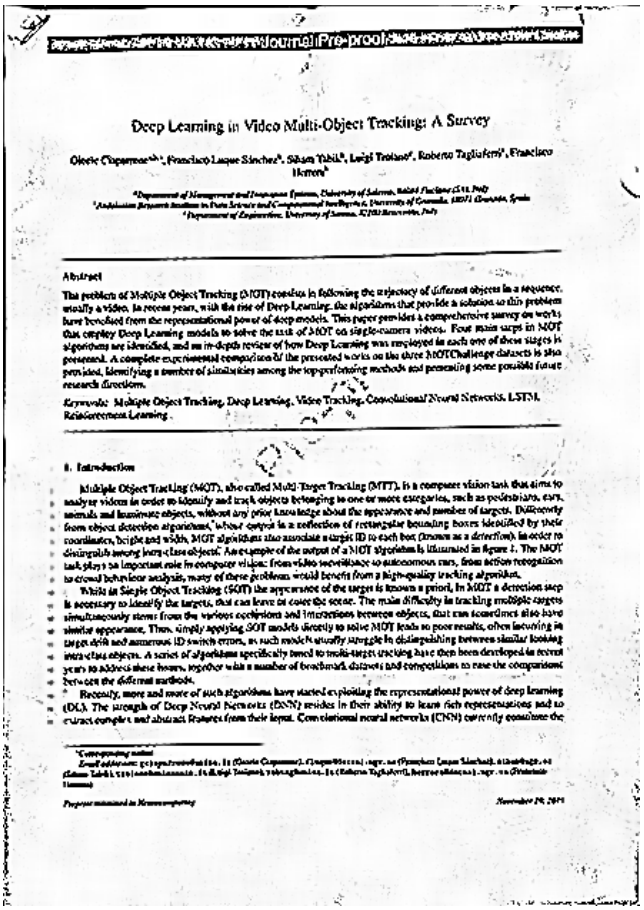
Preprint submitted to Neurocomputing November 19, 2019

برای بهبود کیفیت تصویر و برای اینکه بتوانیم تصویری مانند CamScanner ارائه دهیم می‌توانیم روش‌های مختلفی را استفاده کنیم. یکی از روش‌ها که می‌توانیم استفاده کنیم thresholding یا binarization است. با این روش می‌توانیم به شکلی متن موجود در تصویر را بیشتر از کل زمینه متمایز کنیم. در اینجا ما از adaptive Thresholding استفاده می‌کنیم. در این روش تنها از یک مقدار آستانه کلی استفاده نمی‌کنیم زیرا این روش برای تصاویری که شرایط روشنایی متفاوتی دارند مناسب‌تر است. در این الگوریتم مقدار آستانه برای هر پیکسل بر اساس مقدار کوچک همسایگی کنار آن تعیین می‌شود. روش‌های مختلفی برای محاسبه این آستانه وجود دارد که در اینجا از روش گاوسی استفاده می‌کنیم.

در بخش بعد هم از روش fastNlMeanDenoising برای denoise کردن تصویر استفاده می‌کنیم. این روش برای denoise کردن تصاویر سیاه و سفید بهتر است. این روش Non-local است و صرفا به همسایگی اکتفا نمی‌کند. پارامترهای این تابع قدرت فیلتر، templateWindowSize و searchWindowSize هستند:

```
def enhance(im):
    # Your code goes here.
    im = to_grayscale(im)
    adapt_thresh = cv2.adaptiveThreshold(im, 255, cv2.ADAPTIVE_THRESH_GAUS
SIAN_C, cv2.THRESH_BINARY, 11, 2)

    # denoishing
    denoised = cv2.fastNlMeansDenoising(adapt_thresh, 11, 45, 9)
```

```
    return denoised
```



Deep Learning in Video Multi-Object Tracking: A Survey

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 best-performing methods and proposing some possible future research directions.

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

1. Introduction

منابع:

https://www.projectpro.io/recipes/remove-noise-from-images-opencv
https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html
https://www.tutorialspoint.com/how-to-perform-bilateral-filter-operation-on-an-image-in-opencv-using-python
https://stackoverflow.com/questions/32913157/how-to-get-magic-color-effect-like-cam-scanner-using-opencv
https://github.com/satvik007/Scanner_OP
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html#gsc.tab=0
https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html