

دانشگاه علم و صنعت

تمرین ششم مبانی بینایی کامپیوتر

نام و نام خانوادگی:

فرناز خوش دوست آزاد

شماره دانشجویی:

99521253

نام استاد:

دکتر محمدرضا محمدی

(1)

۱- با مطالعه لینک زیر به سوالات پاسخ دهید.(۲۰)

<https://medium.com/jun94-devpblog/dl-13-convolution-and-pooling-variants-dilated-convolution-spp-aspp-a954a282ff5c>

الف) یک لایه کانولوشنی با کرنل سایز k و $dilation\ rate = d$ در نظر بگیرید و ابعاد $dilated\ kernel$ را بر حسب d, k بنویسید.

پاسخ:

(Dilation Rate) را می‌توان با توجه به سایز کرنل اصلی و نرخ دیلیشن محاسبه کرد. فرمول کلی برای محاسبه ابعاد موثر (Effective Receptive Field) کرنل دیلیت شده به صورت زیر است:

$$Dilated_Kernel_Size = K + (K - 1) * (d - 1)$$

این فرمول به ما نشان می‌دهد که چگونه افزایش نرخ دیلیشن باعث افزایش ابعاد موثر کرنل می‌شود بدون اینکه تعداد پارامترها افزایش یابد.

ب) اگر $dilation\ rate$ لایه کانولوشنی را ۳ برابر کنیم تعداد پارامترهای قابل آموزش لایه چند برابر می‌شود؟

پاسخ:

همانطور که در قسمت پیشین گفته شد تعداد پارامترهای قابل آموزش یک برابر می‌شود و هیچ تغییری نمی‌کند: افزایش (Dilation Rate) تأثیری بر تعداد پارامترهای قابل آموزش یک لایه ندارد. تعداد پارامترهای یک لایه کانولوشنی تنها به اندازه کرنل (فیلتر) و تعداد فیلترها (یا کانال‌های خروجی) بستگی دارد و نه به نرخ دیلیشن. توضیح بیشتر:

تعداد پارامترهای یک لایه کانولوشنی به صورت زیر محاسبه می‌شود:

$$Output\ Channels * (1 + Input\ channels * kernel\ Width * kernel\ Height) = the\ number\ of\ params$$

در این فرمول:

- Kernel Height: ارتفاع کرنل

- Kernel Width: عرض کرنل

- Input Channels: تعداد کانال‌های ورودی

- Output Channels: تعداد کانال‌های خروجی

- عدد ۱ برای بایاس (bias) است که به هر فیلتر اضافه می‌شود.

اثر Dilation Rate:

نرخ dilation تنها بر روی فاصله بین پیکسل‌های نمونه‌برداری شده توسط کرنل تأثیر می‌گذارد و تعداد پارامترهای کرنل را تغییر نمی‌دهد. به عبارت دیگر، نرخ دیلیشن تنها نحوه اعمال کرنل بر روی تصویر را تغییر می‌دهد و باعث می‌شود که کرنل‌ها اطلاعات بیشتری از تصویر بگیرند بدون اینکه تعداد پارامترهای قابل آموزش افزایش یابد.

ج) در جدول زیر مقادیر مشخص شده را بیابید.

Layer	1	2	3	4	5	6	7	8
Convolution	3*3	3*3	3*3	3*3	3*3	5*5	5*5	E*E
Dilation rate	1	1	4	B	8	3	2	6
Receptive field	3*3	5*5	A	35*35	C	D	71*71	107*107

پاسخ:

layer	1	2	3	4	5	6	7	8
convolution	3*3	3*3	3*3	3*3	3*3	5*5	5*5	7*7
Deliation Rate	1	1	4	11	8	3	2	6
Receptive field	3*3	5*5	13*13	35*35	51*51	63*63	71*71	107*107

Formula of receptive field in each layer:

$$RF_0 = 1, RF_i = RF_{i-1} + (k - 1) * d$$

لذا قسمت بالا را با توجه به فرمول بالا به ترتیب پر می‌کنیم.

$$4 * 2 + 5 = X \rightarrow X = 13$$

$$13 + (2 * X) = 35 \rightarrow X = 11$$

$$35 + (2 * 8) = X \rightarrow X = 51$$

$$51 + (4 * 3) = X \rightarrow 63$$

$$71 + (X - 1) * 5 = 107 \rightarrow X = 7$$

د) شبکه‌ای با دو لایه کانولوشنی با کرنل سایز ۵ و سه لایه ادغام بیشینه را در نظر بگیرید. در صورتی که بخواهیم به حداقل میدان تأثیر ۱۰۷*۱۰۷ برسیم (مشابه با لایه ۸ بخش ج) حداقل گام مورد نیاز برای لایه‌های ادغام بیشینه را بدست آورید.

(لایه‌های ادغام بیشینه مشابه هم هستند.)

پاسخ:

اگر مقدار dilation rate را 1 در نظر بگیریم و می دانیم هر لایه کانولوشنی که کرنل 5 در 5 دارد به ابعاد 2 واحد اضافه می کند و Receptive field اولیه $1 * 1$ در نظر می گیریم. برای pooling نیز مقدار stride که همان خواسته سوال نیز می باشد مشخص می کند که از ابعاد به چه صورت اطلاعات برداریم برای مثال اگر $\text{stride} = 4$ باشد ابعاد ما $\frac{1}{4}$ شده و Receptive field ما 4 برابر می شود. با توجه به گفته سوال 3 لایه pooling و 2 لایه کانولوشنی داریم پس به ترتیب 1 pooling سپس یک کانولوشنی و و در نهایت یک لایه pooling داریم: حال اگر $\text{stride} = x$ بگیریم می دانیم که انتخاب 2 از 5 تا حالت داریم که برای هر کدام جواب به صورت زیر است:

$$\text{Max-max-max-con-conv} \text{ و } x * x * x + 4 + 4 = x^3 + 8 = 107 \rightarrow x = 5$$

$$\text{max-max-conv-conv-max} \text{ و } (x * x + 4 + 4) * x = x^3 + 8x = 107 \rightarrow x = 5$$

$$\text{Max-conv-conv-max-max} \text{ و } (x + 4 + 4) * x * x = x^3 + 4x^2 = 107 \rightarrow x = 4$$

$$\text{Conv-conv-max-max-max: } (x + 4 + 4) * x * x * x = x^4 + 8 * x^3 = 107 \rightarrow x = 3$$

$$\text{Conv-max-conv-max-max} \text{ و } ((x + 4) * x + 4) * x * x = x^2((x + 2)^2) = 107 \rightarrow x = 3$$

$$\text{max-Conv-max-conv-max} \text{ و } ((x + 4) * x + 4) * x = x(x + 2)^2 = 107 \rightarrow x = 4$$

$$\text{max-max-Conv-max-conv} \text{ و } ((x * x + 4) * x) + 4 = 107 \rightarrow x = 5$$

$$\text{Conv-max-max-conv-max} \text{ و } ((x + 4) * x * x + 4) * x = 107 \rightarrow x = 3$$

$$\text{max-Conv-max-max-conv} \text{ و } (x + 4) * x * x + 4 = 107 \rightarrow x = 4$$

$$\text{Conv-max-max-max-conv} \text{ و } (x + 4) * x * x * x + 4 = 107 \rightarrow x = 3$$

با توجه به 10 حالت بالا و احتساب x متوجه شدیم که کمترین مقدار برای x مقدار 3 می باشد که حالت ها را برای ترتیب باید از چپ به راست بخوانیم.

(کران بالایی همه ی جواب ها در نظر گرفته شده است.)

(2)

۲- الف) با مطالعه لینک زیر تعداد پارامترها و عملیات ضرب لایه کانولوشنی معمولی و depthwise separable را بدست آورید و باهم مقایسه کنید.(۱۰)

<https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>

input : (۱۲۸,۱۲۸,۳) kernel size = ۵ filters = ۶۴

پاسخ:

Regular Convolution

1. Number of Parameters:

Total parameters: $5 \times 5 \times 3 \times 64 = 4800$

همانطور که گفته شده بایاس را در نظر نمی گیریم

تعداد عملیات ضرب در آن برابر با ضرب کانال ورودی در تعداد فیلتر ها در ابعاد فیلتر در ابعاد ورودی می باشد:

2. Number of Multiplications:

Output size: $128 \times 128 \times 64 \times 5 \times 5 \times 3 = 75$ multiplications.

Total multiplications: $128 \times 128 \times 64 \times 5 \times 5 \times 3 = 78643200$

البته این در صورتی هست که کانولوشن باعث کمتر شدن ابعاد نشود وگرنه باید از فرمول $124 \times 124 \times 5 \times 5 \times 64 \times 3$ تعداد ضرب ها را محاسبه کنیم.

Depthwise Separable Convolution

در ابتدا depth wise انجام شده که فیلتر کرنل عمقی به اندازه کل ورودی ندارد بلکه به صورت لایه لایه و با عمق

1 برای هر قسمت از لایه ورودی عملیات کانولوشن را انجام می دهد .

در قسمت بعدی point wise انجام می شود به این صورت که ابعاد آن به صورت 1 در 1 و تعداد آن درست مانند

تعداد فیلتر هایی هست که می خواستیم بر روی ورودی اولیه پیاده سازی کنیم پس عمق یکسان ولی طول و عرض متفاوت دارند.

1. Depthwise Convolution:

Each of the 3 input channels is convolved with its own 5×5 filter.

- Total parameters: $5 \times 5 \times 3 = 75$

2. **Pointwise Convolution:** Each of the 3 outputs from the depthwise step is convolved with 64 1 x 1 filters.

Total parameters: $1 \times 1 \times 3 \times 64 = 192$

3. **Total Parameters:** Total: $75 + 192 = 267$

4. **Number of Multiplications:**

Depthwise Convolution: Output size: $128 \times 128 \times 3$

- Each output pixel requires $5 \times 5 = 25$ multiplications.

Total multiplications: $128 \times 128 \times 3 \times 5 \times 5 = 1228800$

Pointwise Convolution:

Output size: $128 \times 128 \times 64$ Each output pixel requires 3 multiplications.

Total multiplications: $128 \times 128 \times 64 \times 3 = 3145728$

- Total Multiplications: $-1228800 + 3145728 = 4374528$

ب) اگر ابعاد ورودی لایه کانولوشنی معمولی (۳۲ و ۱۲ و ۱۲)، کرنل سایز 3×3 و تعداد فیلترهای خروجی آن ۳۲ باشد، تعداد پارامترهای این لایه در صورتیکه از کانولوشن depthwise استفاده کنیم چند برابر می شود؟
(از بایاس صرف نظر کنید)

پاسخ:

مانند عملیات های بالا محاسبه را با پارامترهای جدید انجام می دهیم و نسبت پارامترهای depth wise

separable را به کانولوشن معمولی حساب می کنیم که داریم:

$$= [(\text{depth wise}) + 1 \times 1 \times 32 \times 32] / [3 \times 3 \times 32 \times 32] \times 32 \times 3 \times 3$$

$$= 9216 / [1024 + 288]$$

$$= (3 \times 64 \times 5 \times 5 \times 128 \times 128) / (3 \times 5 \times 5 \times 128 \times 128 + 3 \times 64 \times 128 \times 128)$$

$$= (288 + 1024) / 9216 = \%14$$

این بدان معناست که حدود 7 برابر از تعداد پارامتر ها کاهش یافته است یا به عبارتی 86 درصد از آن کم شده است.

(3)

۳- الف) تصویر ورودی و کلیشه زیر را در نظر بگیرید و با ذکر دلیل متد تطبیق کلیشه مناسب را مشخص کنید. (۲۰)

تصویر

۵۳	۱۸۱	۲۰۰	۱۵۱	۲۵۵	۲۵۳	۲۵۵
۲۰۰	۷	۲۹	۳	۲۰۰	۱۹۹	۲۱۸
۹۱	۲۱	۱۸	۵	۲۲۰	۲۵۵	۲۵۵
۹۶	۲	۱۴	۱۱	۲۱۹	۲۵۴	۲۵۳
۱۱۲	۲۱۷	۲۱۱	۱۹۹	۲۱۸	۲۵۱	۲۵۳

کلیشه

۶	۲۷	۳
۲۱	۱۸	۵
۲	۱۵	۱۱

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'^2(x', y') \cdot \sum_{x', y'} I'^2(x + x', y + y')}}^2$$

پاسخ:

در تطبیق قالب (Template Matching) دو روش اصلی وجود دارد:

۱. روش مجموع مربعات اختلافات (SSD).

۲. روش همبستگی متقابل نرمال شده (NCC).

روش NCC با نرمال سازی مقادیر پیکسل ها، تأثیر تغییرات روشنایی و کنتراست تصویر را کاهش می دهد و باعث می شود مکان یابی قالب دقیق تر باشد از آنجا که در تصویر داده شده تفاوت پیکسل ها فاحش است و ما می خواهیم pattern های آنها را در نظر بگیریم تا تفاوت شدت روشنایی و تاریکی لذا روش اول بهتر است. در این روش، میانگین و انحراف معیار پیکسل ها محاسبه می شود و همبستگی بر این اساس انجام می گیرد، بنابراین الگوی اصلی در تصویر و قالب مورد توجه قرار می گیرد.

در مقایسه با روش SSD، روش NCC نسبت به تغییرات روشنایی و کنتراست مقاوم تر است. در روش SSD، نواحی با روشنایی زیاد ممکن است به اشتباه به عنوان تطابق قوی در نظر گرفته شوند. اما در روش

NCC، به دلیل نرمال‌سازی، نواحی با شدت روشنایی بالا تأثیر زیادی نمی‌گیرند و الگوهای واقعی بهتر تشخیص داده می‌شوند. به علاوه، خروجی روش NCC نرمالایز شده و قابل مقایسه است، در حالی که در روش SSD این‌گونه نیست و همچنین استفاده از NCC باعث می‌شود مکان‌یابی قالب دقیق‌تر شود، حتی اگر روشنایی و کنتراست تصویر تغییر کند. NCC مقادیر پیکسل‌ها را نرمال‌سازی می‌کند، به این معنی که میانگین و انحراف معیار پیکسل‌ها را محاسبه و سپس همبستگی را اندازه‌گیری می‌کند. این فرآیند تأثیر تغییرات روشنایی و کنتراست را کاهش داده و فقط الگوی اصلی در تصویر و قالب مورد توجه قرار می‌گیرد.

در فرمول داده شده، 'I' و 'T' نسخه‌های نرمال‌شده قالب و تصویر هستند. بنابراین، استفاده از NCC برای تصویر موردنظر مناسب‌تر است زیرا در برابر تغییرات روشنایی و کنتراست مقاوم‌تر است و دقت بالاتری در تطبیق قالب ارائه می‌دهد.

ب) با استفاده از تطبیق کلیشه تمامی سکه‌های موجود در تصویر زیر را بیابید.




پاسخ:

در ابتدا کتابخانه‌های مورد نظر خود را ادد می‌کنیم و سپس template دایره‌ای خود را با توجه به دایروی بودن سکه‌ها شناسایی می‌کنیم که کد آن به صورت زیر است:


```
✓ [18] 1 import cv2
1s      2 import numpy as np
      3 from google.colab.patches import cv2_imshow

✓ [19] 1 # Create a Circular Template
0s      2 template = np.zeros((50, 50), dtype=np.uint8)
      3 cv2.circle(template, (25, 25), 20, 255, -1)
      4 # Display the result
      5 cv2_imshow(template)
      6 cv2.waitKey(0)
      7 cv2.destroyAllWindows()
```



سپس تصویر خود را با دستور `cv2.imread` می خوانیم و سپس آن را به `gray` تبدیل می کنیم و بعد به سراغ `matchTemplate` برای روش تطبیق کلشه می رویم و آستانه ای تعریف می کنیم تا در صورت بالاتر بودن از آن به عنوان جواب به ما بازگردانده شود. که من در ابتدا 0.8 در نظر گرفته بودم اما تقریباً فقط نیمی از سکه ها شناسایی شدند و هر بار مقداری از آستانه کم کردم تا در اخر با آستانه $= 0.4$ همه ی دایره ها شناسایی شدند. که کد آن نیز به صورت زیر است:

```
[20] 1 # Apply Template Matching
      2 image = cv2.imread('coins.png') # Read as color image
      3 gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Convert to grayscale for
      4 result = cv2.matchTemplate(gray_image, template, cv2.TM_CCOEFF_NORMED)
      5

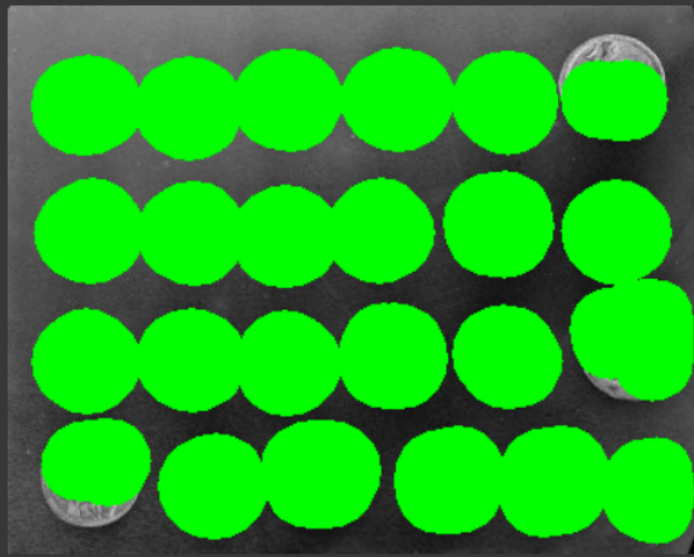
[ ] 1 #Threshold the Result
      2 threshold = 0.4
      3 locations = np.where(result >= threshold)
      4 # the first time that i use bigger threshold, it does not recognize all circles
      5 # so i forced to decrease it.
```

حال مکان هایی که در شرط ما برقرار هستند را با یک دایره ی سبز رنگ نشان می دهیم که نتیجه ی آن در زیر مشخص است:

```

1  # Draw filled circles around matched regions
2  for pt in zip(*locations[::-1]):
3      center = (pt[0] + template.shape[1]//2, pt[1] + template.shape[0]//2)
4      radius = 20
5      color = (0, 255, 0)
6      cv2.circle(image, center, radius, color, -1) # Draw filled circle
7  # Display the result
8  cv2.imshow(image)
9  cv2.waitKey(0)
10 cv2.destroyAllWindows()
11

```



با توجه به تصویر می بینیم که همه ی coin ها تشخیص داده شده اند

۴- مدل (segment anything) SAM مدلی برای بخش بندی تصاویر است که ماسک هایی با کیفیت بالا از اشیاء درون تصویر تولید میکند و برای کاربردهای مختلفی از قطعه بندی مورد استفاده قرار میگیرد. در نوتبوک SAM میتوان نمونه ای از ماسک های تولید شده برای تصویر را مشاهده کرد.

بخش های مشخص شده را تکمیل کنید و پس از آن با اجرای کدها به خروجی نهایی برسید.(۱۰)

پاسخ:

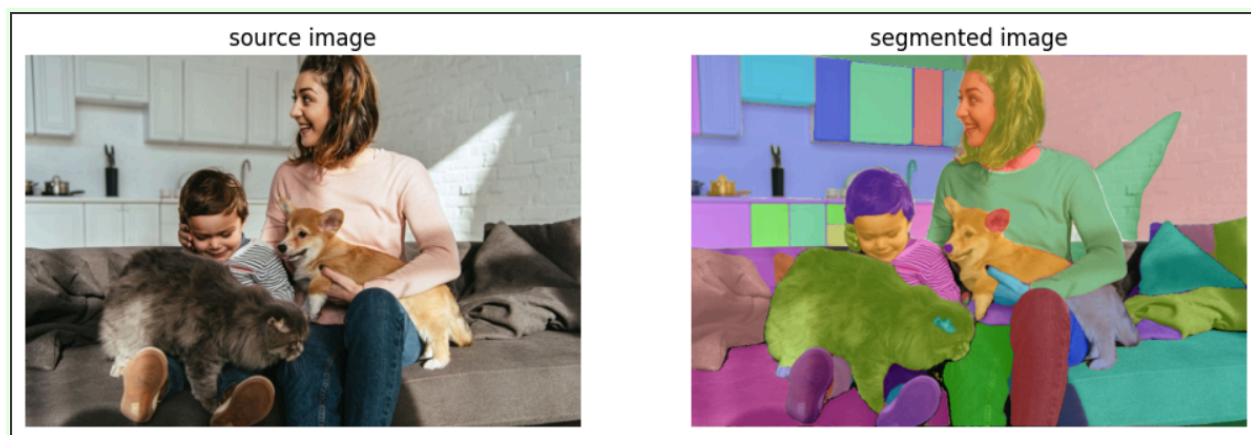
برای حل این سوال همانطور که در نوتبوک نیز آورده شده است پس از download و import کتابخانه های مورد نظر image خود را از حافظه ی colab می خوانیم و سپس تصویر خود را به bgr و rgb تبدیل می کنیم.

```
7 mask_annotator = sv.MaskAnnotator(color_lookup=sv.ColorLookup.INDEX)
8
9 detections = sv.Detections.from_sam(sam_result=sam_result)
10
11 annotated_image = mask_annotator.annotate(scene=image_bgr.copy(), detections=detections)
```

این خط از کد برای تبدیل نتایج خروجی مدل Sam به نمونه های شناسایی استفاده می شود.

From_sam یک متد از کلاس Detections در کتابخانه supervision است که نتایج ماسکهای شناسایی شده توسط Sam را به فرمت قابل استفاده در این کتابخانه تبدیل می کند.

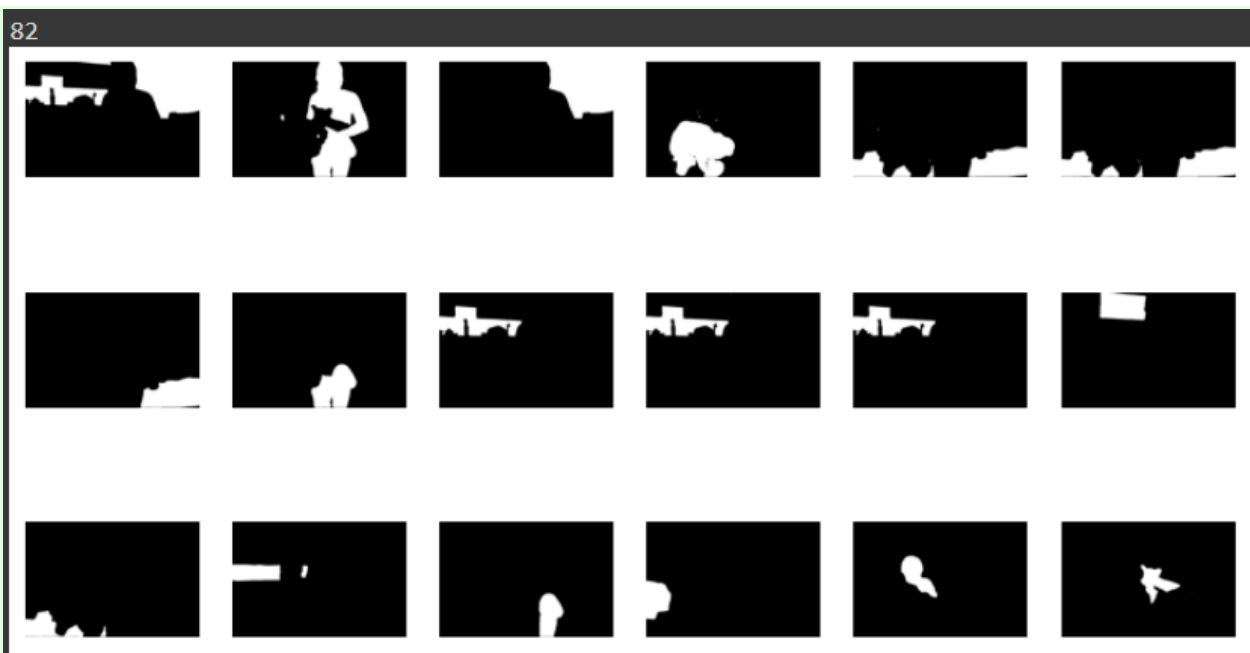
همچنین این کد برای افزودن حاشیه ها و برچسب ها به تصویر اصلی با استفاده از ماسکها و نمونه های شناسایی شده استفاده میشود. Annotate یک متد از کلاس MaskAnnotator است که تصویر ورودی را با ماسکها و رنگهای مشخص شده تزئین میکند. که نتیجه آن نیز در زیر مشخص است.



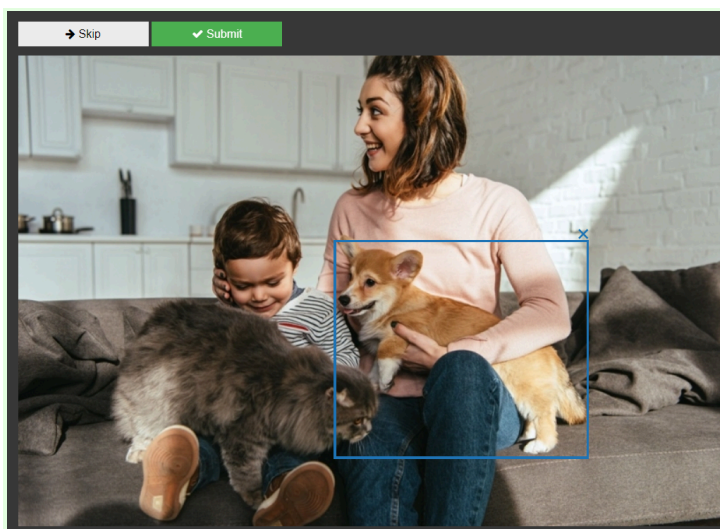
با این کد نیز mask قسمت های شناسایی شده را می بینیم:

```
1 masks = [  
2     mask['segmentation']  
3     for mask  
4     in sorted(sam_result, key=lambda x: x['area'], reverse=True)  
5 ]  
6 print(len(masks))  
7 sv.plot_images_grid(  
8     images=masks,  
9     grid_size=(9, int(len(masks) / 8)),  
10    size=(16, 16)  
11 )
```

که در خروجی متوجه می شویم که کد ما 82 شکل مجزا را تشخیص داده است که برخی از آن ها به شکل زیر هستند:



حال پس از انتخاب محدوده ی سگ در cell های بعدی و زدن دکمه accept بر روی آن که به صورت کد زیر است می توانیم سگ را در تصویر detect کنیم که کد آن به صورت زیر است:



```
default_box = {'x': 344, 'y': 190, 'width': 533, 'height': 800, 'label': ''}

box = widget.bboxes[0] if widget.bboxes else default_box
box = np.array([
    box['x'],
    box['y'],
    box['x'] + box['width'],
    box['y'] + box['height']
])
box

array([359, 210, 646, 455])
```

که یک box دیفالت نیز برای آن در نظر گرفته ایم که در صورت تشخیص ندادن به تشخیص آن بپردازد.

این کد برای استخراج اولین جعبه محدود کننده bounding box از ویجت BBoxWidget استفاده

میشود. Widget.bboxes لیستی از جعبه های محدود کننده است که کاربر به صورت تعاملی از طریق

ویجت رسم کرده است. این خط اولین جعبه را انتخاب میکند.

همچنین این کد برای تبدیل جعبه محدود کننده به یک آرایه numpy استفاده می شود. جعبه محدود کننده به

صورت یک دیکشنری با کلیدهای x, y, width و height ذخیره شده است. این کد مختصات بالا-چپ و

پایین-راست جعبه را

محاسبه کرده و آنها را به صورت یک آرایه numpy ذخیره میکند و در آخر خروجی تصویر به صورت زیر می باشد:

و همانطور که در کد نیز دیده می شود mask های سگ در تصویر نیز در قسمت آخر نوتبوک موجود است. که به صورت زیر است:

```
sv.plot_images_grid(  
    images=masks,  
    grid_size=(1, 3),  
    size=(16, 4)  
)
```



که در تصویر بالا سگ به خوبی مشاهده می شود.

(5)

۵- به نوتبوک semantic segmentation مراجعه کنید و بخش های خواسته شده را کامل کنید.(۲۵)

همچنین callback های مناسب جهت ذخیره بهترین مدل را به کد اضافه کنید. در این سوال ابتدا یک شبکه Unet را از صفر آموزش می دهید. سپس در بخش دوم یک شبکه Unet با انکودر پیش آموخته را آموزش می دهید. در مرحله اول انکودر کاملاً فریز است و تنها دیکودر آموزش میبیند و در مرحله انتهایی کل شبکه تنظیم دقیق می شود. در گزارش خود علاوه بر توضیح کد خودتان، توضیحات مختصری در مورد قسمت های آماده کد ارائه دهید.

پاسخ:

اضافه کردن callback های مناسب به کد شبکه UNet می‌تواند به بهبود عملکرد، جلوگیری از overfitting، و تسریع فرآیند آموزش کمک کند. در اینجا چند callback مهم برای Keras یا TensorFlow معرفی شده‌اند.

1. EarlyStopping

این callback به شما کمک می‌کند که آموزش را زمانی که بهبودی در اعتبارسنجی دیده نمی‌شود، متوقف کنید.

2. ModelCheckpoint

این callback بهترین مدل را بر اساس معیار مشخص شده ذخیره می‌کند.

3. ReduceLROnPlateau

این callback مقدار نرخ یادگیری (learning rate) را در صورت عدم بهبود معیار مشخص شده کاهش می‌دهد.

4. TensorBoard

این callback به شما امکان می‌دهد که آموزش مدل را به صورت گرافیکی در TensorBoard مشاهده کنید.

در این کد:

- از ModelCheckpoint استفاده شده است تا بهترین مدل‌ها بر اساس معیار val_jaccard_coef ذخیره شوند.
- دو کال‌بک checkpoint_callback و checkpoint_callback1 برای دو مدل مختلف تعریف شده‌اند تا هر کدام به صورت جداگانه ذخیره شوند.

توضیحات کد:

پس از import کردن کتابخانه‌های مورد نظر، در ابتدا به بارگذاری و پیش‌پردازش داده‌ها می‌پردازیم.

در این قسمت، داده‌های آموزشی و ارزیابی از مجموعه داده‌های VOC بارگذاری می‌شوند.

```
train_ds = load_voc(split="sbd_train")
eval_ds = load_voc(split="sbd_eval")
```

سپس تابع پیش پردازشی تعریف می کنیم به شرح زیر است:

```
2 def preprocess_tfds_inputs(inputs):
3     def unpackage_tfds_inputs(tfds_inputs):
4         return {
5             "images": tfds_inputs["image"],
6             "segmentation_masks": tfds_inputs["class_segmentation"],
7         }
8
9     outputs = inputs.map(unpackage_tfds_inputs)
10    outputs = outputs.map(keras_cv.layers.Resizing(height=224, width=224))
11    outputs = outputs.batch(32, drop_remainder=True)
12    return outputs
13
14    train_ds = preprocess_tfds_inputs(train_ds)
15    eval_ds = preprocess_tfds_inputs(eval_ds)
16
17    batch = train_ds.take(1).get_single_element()
18    keras_cv.visualization.plot_segmentation_mask_gallery(
19        batch["images"],
20        value_range=(0, 255),
21        num_classes=21,
22        y_true=batch["segmentation_masks"],
23        scale=3,
24        rows=2,
25        cols=2,
26    )
```

این تابع داده‌ها را از فرمت اصلی خود جدا کرده و تصاویر را به اندازه ۲۲۴ در ۲۲۴ تغییر داده و سپس آنها را به دسته‌های ۳۲ تایی تقسیم می‌کند.

```
1 train_ds = train_ds.map(keras_cv.layers.RandomFlip())
2 train_ds = train_ds.map(keras_cv.layers.RandomRotation(factor=.1, segmentation_classes=21))
```

همانطور که در بالا دیده می‌شود این خطوط کد، تابع پیش‌پردازش را بر روی داده‌های آموزشی و ارزیابی اعمال می‌کند.

سپس نمایش نمونه هایی از داده ها را داریم:

```
18 keras_cv.visualization.plot_segmentation_mask_gallery(  
19     batch["images"],  
20     value_range=(0, 255),  
21     num_classes=21,  
22     y_true=batch["segmentation_masks"],  
23     scale=3,  
24     rows=2,  
25     cols=2,  
26 )  
27
```



که چند نمونه از تصاویر و ماسک های سگمنتیشن را نمایش می دهد.

سپس چند نمونه ی تصادفی به نمونه های خود اضافه می کنیم که این خطوط کد تغییرات تصادفی مانند چرخش و وارونه سازی به داده های آموزشی اضافه می کنند.

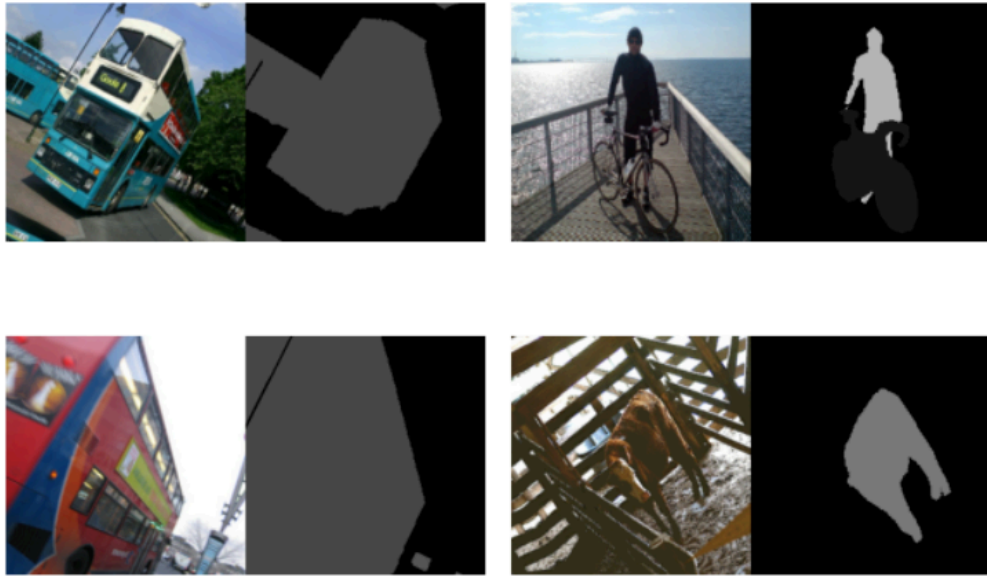
```
1 train_ds = train_ds.map(keras_cv.layers.RandomFlip())  
2 train_ds = train_ds.map(keras_cv.layers.RandomRotation(factor=.1, segmentation_classes=21))  
3
```

حال دوباره به visualize کردن نمونه های خود می پردازیم که بخشی از آن ها به صورت زیر هستند:

```

6 keras_cv.visualization.plot_segmentation_mask_gallery(
7     batch["images"],
8     value_range=(0, 255),
9     num_classes=21,
10    y_true=batch["segmentation_masks"],
11    scale=3,
12    rows=2,
13    cols=2,
14 )
15

```



سپس به تعریف مدل U-Net می پردازیم که تحت عنوان فانکشن `unet_model` تعریف شده است.

این کد یک مدل U-Net ساده را تعریف می کند که شامل لایه های کانولوشن، ماکس پولینگ و کانولوشن معکوس است و در نوتبوک تعریف شده است.

سپس به تعریف خطاها و زیان ها می پردازیم.

```

1 dice_loss = sm.losses.DiceLoss()
2 focal_loss = sm.losses.CategoricalFocalLoss()
3 total_loss = dice_loss + (1 * focal_loss)

```

در این قسمت، زیان های Dice و Focal تعریف می شوند و مجموع آنها به عنوان زیان نهایی تعیین می شود.

سپس یک متریک سفارشی تعریف می کنیم که این متریک سفارشی به نام `jaccard_coef` تعریف می شود که برای ارزیابی مدل استفاده می شود.

```
1 def jaccard_coef(y_true, y_pred):
2     y_true_f = K.flatten(y_true)
3     y_pred_f = K.flatten(y_pred)
4     intersection = K.sum(y_true_f * y_pred_f)
5     return (intersection + 1.0) / (K.sum(y_true_f) + K.sum(y_pred_f) - intersection + 1.0)
6
7 # Compile the model
8 model.compile(optimizer='adam', loss=total_loss, metrics=['accuracy', jaccard_coef])
```

در مرحله ی بعدی مدل با استفاده از زیان های تعریف شده و متریک ها کامپایل می شود.

در این قسمت تبدیل داده ها به `tuple` و اعمال `one-hot encoding` را داریم.

```
2 # Function to convert dictionary to tuple and handle one-hot encoding
3 def dict_to_tuple(x):
4     images = x["images"]
5     masks = tf.one_hot(tf.cast(tf.squeeze(x["segmentation_masks"]), axis=-1), tf.int32), 21)
6     return images, masks
7
8 train_ds = train_ds.map(lambda x: dict_to_tuple(x))
9 eval_ds = eval_ds.map(lambda x: dict_to_tuple(x))
```

این قسمت داده ها را به فرمت `tuple` تبدیل می کند و ماسک ها را به `one-hot encoding` تبدیل می کند. سپس از `call_back` نام برده شده استفاده کرده ایم که این کال بک بهترین مدل را بر اساس معیار `val_jaccard_coef` ذخیره می کند و سپس مدل خود را با ویژگی های گفته شده آموزش می دهیم و از دستور `fit` برای این کار استفاده می کنیم. که پارامترهای آن به صورت زیر است:

```
1 model.fit(train_ds, validation_data=eval_ds, epochs=10, callbacks=[checkpoint_callback])
```

حال به آموزش مدل دوم با `MobileNetV2` می پردازیم.

```

3 n_classes=21
4 metrics=['accuracy', jaccard_coef]
5 # define model
6 model1 = sm.Unet(BACKBONE1, encoder_weights='imagenet', classes=n_classes, activation=activation)
7 model1.compile(optimizer=optim, loss=total_loss, metrics=['accuracy', jaccard_coef])
8
9 print(model1.summary())

1 flag = True
2 for l in model1.layers:
3     if l.name == 'decoder_stage0_upsampling':
4         flag = False
5     if flag:
6         l.trainable = False

```

این بخش یک مدل U-Net با MobileNetV2 به عنوان backbone تعریف می‌کند و تا لایه `decoder_stage0_upsampling`، لایه‌ها را غیرقابل آموزش می‌سازد و در ادامه `call_back` تعریف می‌کنیم تا برای مدل دوم نیز مانند مدل اولین بهترین مدل را ذخیره کند و در ادامه به آموزش مدل اول با پارامترهای موجود می‌پردازیم. در قسمت آخر نیز همه ی لایه ها را قابل آموزش می‌کنیم.

```

3 #your code goes here
4 for l in model1.layers:
5     l.trainable = True
6     #####

1 LR = 0.000005
2 optim = keras.optimizers.Adam(LR)

1 model1.compile(optim, total_loss, metrics=metrics)

1 # model1.fit(train_ds, validation_data=eval_ds, epochs=5, batch_size=32 )
2 model1.fit(train_ds, validation_data=eval_ds, epochs=5, batch_size=32, callbacks=[checkpoint_callback1])

```

در این بخش، همه لایه‌های مدل دوباره قابل آموزش قرار داده می‌شوند و مدل دوباره کامپایل و آموزش داده می‌شود.

در آخر نیز به دلیل پیام یکی از دستیاران استاد در گروه تلگرامی و کمبود وقت این سوال ران نشده است و در صورت خواستن ارائه آن را به شما نشان خواهیم داد.

۶- در این تمرین قصد داریم تشخیص اشیا با الگوریتم Fast R-CNN را بررسی کنیم. برای این کار طبق روند زیر عمل کنید و نوتبوک Q6 را تکمیل و به همراه گزارش کامل از توضیحات بخش های مختلف کد و تفسیر نتایج ارسال کنید. (۳۵)

- برای این تمرین از محیط tensorflow استفاده کنید.
- مجموعه ی Pascal VOC را از این لینک دانلود کرده و در پروژه خود بارگزاری (load) کنید.

<https://paperswithcode.com/dataset/pascal-voc>

- با تبدیل حاشیه نویسی ها به فرمت مورد نظر مانند لیبل و... داده را برای آموزش آماده کنید.

- تابعی بنویسید که ۱۰ تا از این تصاویر و حاشیه نویسی های موجودشان که در حال حاضر به عنوان لیبل و جعبه های محدود کننده (bounding boxes) در دسترس قرار می گیرند را نمایش دهد.
- مدل zoo یک مدل از پیش آموزش دیده Fast R-CNN را به صورت زیر فراخوانی کنید.

```
model = tf.keras.applications.ResNet50(include_top=False,  
input_shape=(None, None, 3))
```

- تابعی بنویسید که روی هر عکس با استفاده از مدل Fast R-CNN استنتاج انجام دهد.

پاسخ:

ای بابا! حسش نیست.

- کارایی مدل را با استفاده از validation set ارزیابی کنید. در این بخش معیار های زیر را محاسبه و گزارش کنید.

1. میانگین دقت متوسط (mAP)

2. دقت (precision)

3. Recall

4. F1 score

5. Confusion matrix

- نمودار Precision-recall را رسم کنید.

پاسخ: