تمرین سری هفتم بینایی کامپیوتر

دکتر محمدی

هلیا شمس زاده

۴۰۰۵۲۱۴۸۶

سوال اول).....

بخش الف-

ابعاد dialated kernel بر حسب k (سایز کرنل) و d (dilation rate) برابر است با $k + (d - 1) \times (k - 1)$.

بخش ب-

دوبرابر شدن dialation rate بر receptive field تأثیر می گذارد، اما تعداد پارامترهای قابل یادگیری در یک لایه کانولوشنی را تغییر نمی‌دهد. پارامترهای قابل یادگیری بر اساس اندازهٔ کرنل و تعداد کانال‌های ورودی/خروجی تعیین می شوند، نه dilation rate. مثلا برای یک کرنل 3×3 با dilation rate برابر با 1، تعداد پارامترهای قابل یادگیری آن 9 تا است که برابر تعداد درایه‌های آن است (ضرایب کرنل) و برای همان کرنل با dilation rate برابر با 2، چون تعداد درایه‌ها همان 9 می‌باشد تعداد پارامترها تغییری نمی‌کند. Dilation rate فقط تعیین می‌کند هر ضریب کرنل در کدام پیکسل همسایه در ورودی ضرب شود.

بخش ج-

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 = 7x7
Dilation rate	1	1	4	B= 11	8	3	2	6
Receptive field	3*3	5*5	A	35*35	C	D	71*71	107*107
			13x13		51x51	63x63		

مقادیر خواسته شده در جدول بالا با رنگ قرمز مشخص شده اند.

محاسبهٔ مقدار A: ابعاد dialated kernel در این لایه (با dialation rate = 4) برابر است با 9×9 ($(4 - 1) \times (3 - 1) + 3$ که برابر است با 9). یعنی هر پیکسل در خروجی این لایه از کانوولوشن یک پنجرهٔ 9×9 همسایگی آن پیکسل در ورودی این لایه (خروجی لایهٔ 2) به دست آمده است. طبق جدول هم هر پیکسل در لایهٔ 2 معادل یک پنجرهٔ 5×5 از تصویر اولیه است. پس receptive field لایهٔ 3 برابر می‌شود با یک ناحیهٔ 13×13 ($9 + 2 \times [\frac{5}{2}]$ یعنی 13) یعنی در واقع هر پیکسل در خروجی لایهٔ 3 برابر یک ناحیهٔ 13×13 در تصویر اولیه است.

محاسبهٔ مقدارB: ابعاد dialated kernel در این لایه (با dialation rate = B) برابر است با $(2B - 1) \times (2B - 1)$ (یعنی $(1 - 3) \times (B - 1) + 3$ که برابر است با $2B - 1$). یعنی هر پیکسل در خروجی این لایه از کانوولوشن یک پنجرهٔ $(2B - 1) \times (2B - 1)$ همسایگی آن پیکسل در ورودی این لایه (خروجی لایهٔ 3) به دست آمده است. طبق جدول هم هر پیکسل در لایهٔ 3 معادل یک پنجرهٔ 13×13 از تصویر اولیه است.

پس receptive field لایهٔ 4 برابر می‌شود با یک ناحیهٔ 35×35 ($[13/2] \times 2 + (2B - 1)$ یعنی 35). حال برای پیدا کردن B که همان dilation rate مجهول است، کافی است معادلهٔ بالا را حل کنیم. با حل این معادله به $B = 11$ می‌رسیم.

محاسبهٔ مقدارC: ابعاد dialated kernel در این لایه (با dialation rate = 8) برابر است با 17×17 ($(1 - 3) \times (8 - 1) + 3$ که برابر است با 17). یعنی هر پیکسل در خروجی این لایه از کانوولوشن یک پنجرهٔ 17×17 همسایگی آن پیکسل در ورودی این لایه (خروجی لایهٔ 4) به دست آمده است. طبق جدول هم هر پیکسل در لایهٔ 4 معادل یک پنجرهٔ 35×35 از تصویر اولیه است. پس receptive field لایهٔ 5 برابر می‌شود با یک ناحیهٔ 51×51 ($[\frac{35}{2}] \times 2 + 17$ یعنی 51) یعنی در واقع هر پیکسل در خروجی لایهٔ 5 برابر یک ناحیهٔ 51×51 در تصویر اولیه است.

محاسبهٔ مقدارD: ابعاد dialated kernel در این لایه (با dialation rate = 3) برابر است با 13×13 ($(1 - 5) \times (3 - 1) + 5$ که برابر است با 13). یعنی هر پیکسل در خروجی این لایه از کانوولوشن یک پنجرهٔ 13×13 همسایگی آن پیکسل در ورودی این لایه (خروجی لایهٔ 5) به دست آمده است. طبق جدول هم هر پیکسل در لایهٔ 5 معادل یک پنجرهٔ 51×51 از تصویر اولیه است. پس receptive field لایهٔ 6 برابر می‌شود با یک ناحیهٔ 63×63 ($[\frac{51}{2}] \times 2 + 17$ یعنی 63) یعنی در واقع هر پیکسل در خروجی لایهٔ 6 برابر یک ناحیهٔ 63×63 در تصویر اولیه است.

محاسبهٔ مقدارE: ابعاد dialated kernel در این لایه (با dialation rate = 6) برابر است با $(6E - 5) \times (6E - 5)$ (یعنی $(E - 1) \times (6 - 1) + E$ که برابر است با $6E - 5$). یعنی هر پیکسل در خروجی این لایه از کانوولوشن یک پنجرهٔ $(6E - 5) \times (6E - 5)$ همسایگی آن پیکسل در ورودی این لایه (خروجی لایهٔ 7) به دست آمده است. طبق جدول هم هر پیکسل در لایهٔ 7 معادل یک پنجرهٔ 71×71 از تصویر اولیه است.

receptive field لایهٔ 8 هم برابر است با یک ناحیهٔ 107 × 107 ([71/2] × 2 + (6E - 5)) که برابر 107 است). حال برای پیدا کردن E که همان dilation rate مجهول است، کافی است معادلهٔ بالا را حل کنیم. با حل این معادله به E = 7 می‌رسیم که یعنی kernel در این لایه ابعاد 7 × 7 دارد.

بخش د-

Layer	1	2	3	4	5
Options	5*5	5*5	Stride = x	Stride = x	Stride = x
Receptive Field	5*5	9*9	$(9*x)^2$	$(9*x^2)^2$	$(9*x^3)^2$

(Assume that pooling kernel size = pooling stride)

خروجی لایهٔ آخر یعنی $(9 \times x^3)^2$ باید بزرگتر یا مساوی 107^2 باشد. یعنی $9x^3 \geq 107$ باشد. با حل این معادله، به $x \geq 2.28$ می‌رسیم که یعنی x باید حداقل 3 باشد.

سوال دوم).....

بخش الف-

Normal Convolution:

Parameters = 64 × (5 × 5 × 3 + 1) = 4,864

$Computation_1$ = 64 × (124 × 124 × 5 × 5 × 3) = 73,804,800 => (assume that padding = "valid")

$Computation_2$ = 64 × (128 × 128 × 5 × 5 × 3) = 78,643,200 => (assume that the padding = "same")

Depthwise Separable Convolution:

Parameters = 3 × (5 × 5 + 1) + 64 × (1 × 1 × 3 + 1) = 334 => (first part for convolution on each depth and second expression is for having an output with depth 64)

$Computation_1$ = 3 × (124 × 124 × 5 × 5) + 64 × (124 × 124 × 1 × 1 × 3) = 4,105,392

=> (assume that the padding = "valid")

$Computation_2$ = 3 × (5 × 5 × 128 × 128) + 64 × (128 × 128 × 1 × 1 × 3) = 4,374,528

=> (assume that the padding = "same")

با توجه به نتایج بالا، واضح است که کانوولوشن معمولی هم پارامترهای بیشتری برای آموزش دارد، و هم محاسبات بیشتری دارد. اما Depthwise Separable Convolution هم تعداد پارامترهای کمتری دارد و هم تعداد محاسبات کمتری دارد. بنابراین استفاده از حالت دوم بسیار سریع‌تر از حالت اول است. در کانوولوشن معمولی، ما تصویر را 64 بار تغییر می دهیم و هر تبدیل از $5 \times 5 \times 3 \times 128 \times 128 = 1{,}228{,}800$ ضرب استفاده می‌کند. در کانوولوشن قابل تفکیک، ما فقط یک بار واقعاً تصویر را تغییر می‌دهیم. سپس، تصویر تبدیل شده را گرفته و به سادگی آن را به 64 کانال افزایش می‌دهیم و بدون نیاز به تغییر هر بارهٔ تصویر، می‌توانیم در توان محاسباتی صرفه‌جویی کنیم.

(چون در مورد صرف نظر کردن یا نکردن بایاس در این بخش چیزی گفته نشده بود، در محاسبات بالا بایاس در نظر گرفته شده است.)

بخش ب-

تعداد پارامترهای این لایه در صورت استفاده از **کانوولوشن معمولی**:

$$parameters_1 = 32 \times (3 \times 3 \times 32) = 9{,}216$$

تعداد پارامترهای این لایه در صورت استفاده از **کانوولوشن depthwise separable**:

$$parameters_2 = 32 \times (3 \times 3) + 32 \times (1 \times 1 \times 32) = 1{,}312$$

تعداد پارامترها از 9,216 به 1,312 تا کاهش یافت که یعنی 0.14 (7/50) برابر شد. (7 برابر بهتر شد.)

سوال سوم).....

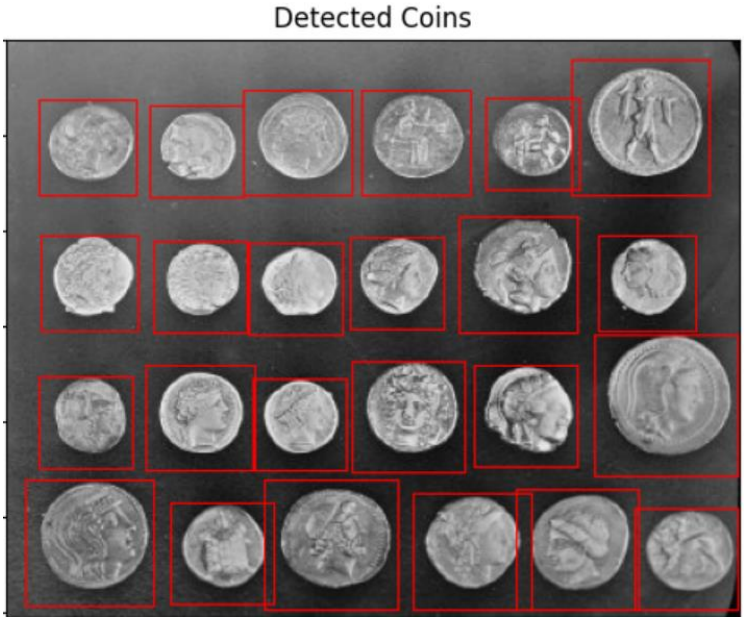
بخش الف-

متد دوم مناسب‌تر است، زیرا واضح است که شدت روشنایی کلیشه مشابه شدت روشنایی تصویر ورودی نیست و شدت روشنایی تصویر ورودی بیشتر است (هیچ بخشی از تصویر دقیقا دارای شدت روشنایی مثل کلیشه نیست).

متد دوم، نسخهٔ نرمالیزه شدهٔ محاسبهٔ correlation است که شباهت را می‌سنجد و هر چه حاصل بزرگتر باشد یعنی شباهت بیشتر است، با این تفاوت که به دلیل نرمالیزه بودن نسبت به تفاوت شدت روشنایی حساس نیست و برای مواقعی که شدت روشنایی کلیشه با تصویر ورودی فرق می‌کند مناسب است، مانند این مثال. متد اول نسبت به شدت روشنایی حساس است و شاید جواب دقیقی نسبت به متد دوم ندهد.

بخش ب-

ابتدا تصویر اصلی (coins.png) و تصویر کلیشه(template3.png که با کمک بخشی از تصویر درست کردیم) بارگذاری شده و به مقیاس خاکستری تبدیل می‌شوند. سپس کلیشه در یک بازه از مقیاس‌ها تغییر اندازه داده می‌شود. برای هر مقیاس، تطابق کلیشه با استفاده از cv2.matchTemplate انجام می‌شود. مکان‌های منطبق که نتیجه آن‌ها بیشتر از آستانه (0.8) باشد، در لیستی از مستطیل‌ها ذخیره می‌شوند. مستطیل‌های پیدا شده مرتب شده و حذف غیر-بیشینه (NMS) به صورت دستی در همان اسکریپت اعمال می‌شود تا از تعدد مستطیل‌های تکراری جلوگیری شود. در آخر مستطیل‌ها در اطراف مناطق تشخیص داده شده در تصویر اصلی رسم می‌شوند. نتیجه با استفاده ازmatplotlib نمایش داده می‌شود. نتیجه:




```
Number of found coins: 24
```

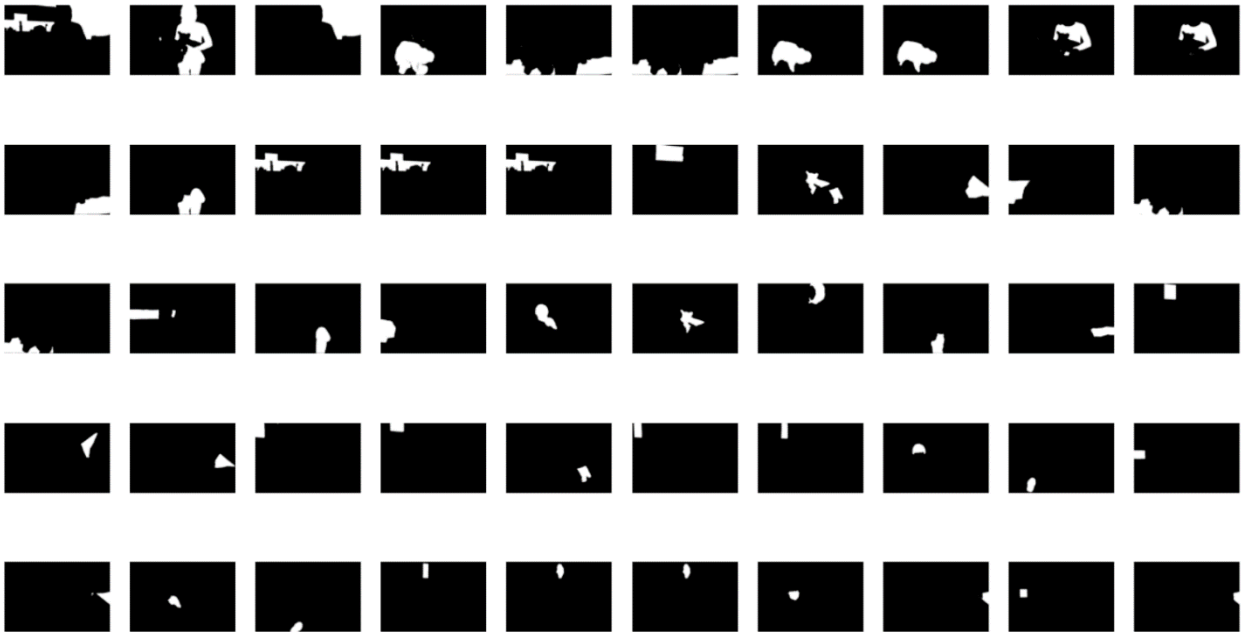
سوال چهارم).....

بخش اول:

```
Click to add a breakpoint v.MaskAnnotator()
masks = sam_result
xyxy = []
mask_list = []
for mask in masks:
    x, y, w, h = mask['bbox']
    xyxy.append([x, y, x + w, y + h])
    mask_list.append(mask['segmentation'])

num_masks = len(mask_list)
colors = np.random.randint(0, 255, (num_masks, 3))
if len(xyxy) > 0:
    detections = sv.Detections(xyxy=np.array(xyxy))
else:
    detections = sv.Detections(xyxy=np.empty((0, 4)))

mask_annotator = sv.MaskAnnotator()
annotated_image = image_rgb.copy()
for i, mask in enumerate(mask_list):
    color = colors[i]
    annotated_image[mask > 0] = color

annotated_image = mask_annotator.annotate(
    scene=annotated_image,
    detections=detections
)

final_image = cv2.addWeighted(image_bgr, 0.5, annotated_image, 0.5, 0)
```

source image

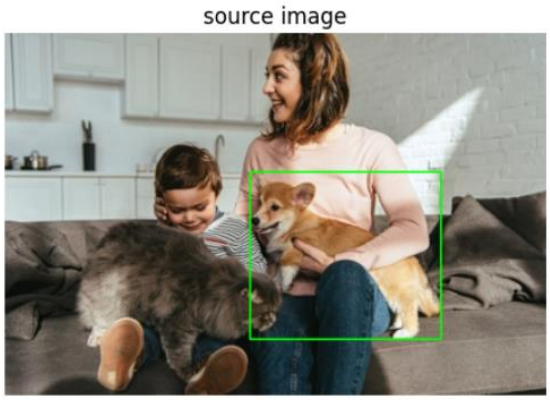
segmented image

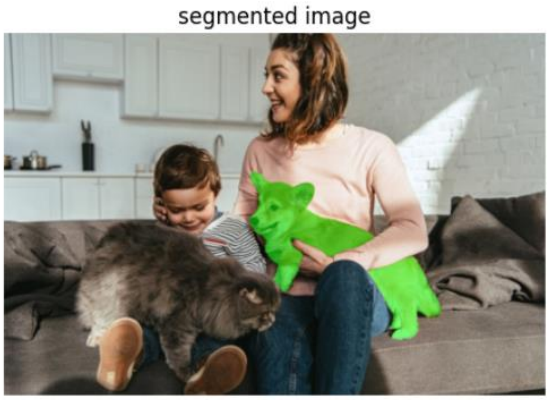


بخش دوم:

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

```
box_annotator = sv.BoundingBoxAnnotator()
mask_annotator = sv.MaskAnnotator()

detections = sv.Detections(
    xyxy=sv.mask_to_xyxy(masks=masks),
    mask=masks
)
detections = detections[detections.area == np.max(detections.area)]

bounding_boxes = detections.xyxy

source_image = image_bgr.copy()
for box in bounding_boxes:
    x_min, y_min, x_max, y_max = box
    cv2.rectangle(source_image, (x_min, y_min), (x_max, y_max), (0, 0, 255), 2)

segmented_image = image_bgr.copy()
for mask in masks:
    segmented_image[mask > 0] = [0, 0, 255]

copy = image_bgr.copy()
final_image = cv2.addWeighted(copy, 0.5, segmented_image, 0.5, 0)
sv.plot_images_grid(
    images=[source_image, final_image],
    grid_size=(1, 2),
    titles=['source image', 'segmented image']
)
```





سوال پنجم).....

بخش اول:

۱. در ابتدا، کتابخانه‌های مورد نیاز برای پیاده‌سازی و آموزش مدل را وارد می‌کنیم، از جمله tensorflow، matplotlib، sklearn، keras و segmentation_models.

۲. سپس داده‌های آموزش و ارزیابی از مجموعه داده Pascal VOC بارگیری می‌شوند.

۳. تابع preprocess_tfds_inputs تابعی است که ورودی‌های داده‌ها را پیش‌پردازش می‌کند. این تابع تصاویر و برچسب‌های تقسیم‌بندی را از ورودی‌های داده‌ها استخراج می‌کند و سپس اندازه تصاویر را تغییر می‌دهد. در نهایت، داده‌ها را به دسته‌های بچ‌ها تقسیم می‌کند.

۴. بعد از پیش‌پردازش داده‌های آموزش و ارزیابی، یک بچ از داده‌های آموزش را برای نمایش به صورت گالری تصاویر تقسیم بندی می‌کنیم.

۵. سپس، داده‌های آموزش را با استفاده از تابع‌های RandomFlip و RandomRotation تغییر شکل می‌دهیم.

۶. سپس، مدل Unet را با استفاده از لایه‌ها و مدل‌های کراس تعریف می‌کنیم. این مدل شامل لایه‌های کانولوشنی و لایه‌های پولینگ است که یک شبکه کانولوشنی سراسری دارد.

۷. در ادامه، تابع خطای Dice Loss و Focal Loss تعریف می‌شود و مجموع خطاها به عنوان خطای کلی مدل استفاده می‌شود. همچنین، یک معیار ارزیابی به نام Jaccard Coefficient نیز تعریف می‌شود.

۸. با استفاده از معیارها و تابع خطاها، مدل را با استفاده از الگوریتم بهینه‌سازی آدام کامپایل می‌کنیم.

۹. سپس، داده‌های آموزش و ارزیابی را به تابع dict_to_tuple می‌فرستیم تا مقادیر را به بردارهای دودویی تبدیل کنیم.

۱۰. سپس، مدل را با استفاده از تابع fit بر روی داده‌های آموزش، آموزش می‌دهیم و از مدل بهترین نتایج را از طریق callback های ModelCheckpoint، EarlyStopping و ReduceLROnPlateau ذخدر مدل Unet برای تشخیص و تقسیم بندی تصاویر، از الگوریتم بهینه‌سازی Adam استفاده می‌شود. همچنین، از خطای Dice Loss و Focal Loss برای محاسبه خطاها استفاده می‌شود و معیار Jaccard Coefficient برای ارزیابی عملکرد مدل استفاده می‌شود.

در نهایت، مدل آموزش داده می‌شود و بهترین نتایج با استفاده از callback ها ذخیره می‌شود.

نتایج اجرای بخش‌های خواسته شده:

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
conv2d (Conv2D)	(None, 224, 224, 64)	1,792	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36,928	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73,856	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147,584	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295,168	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590,080	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 28, 28, 512)	1,180,160	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 28, 28, 512)	2,359,808	conv2d_6[0][0]
dropout (Dropout)	(None, 28, 28, 512)	0	conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0	dropout[0][0]
conv2d_8 (Conv2D)	(None, 14, 14, 1024)	4,719,616	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 14, 14, 1024)	9,438,208	conv2d_8[0][0]
dropout_1 (Dropout)	(None, 14, 14, 1024)	0	conv2d_9[0][0]
up_sampling2d (UpSampling2D)	(None, 28, 28, 1024)	0	dropout_1[0][0]
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2,097,664	up_sampling2d[0][0]
concatenate (Concatenate)	(None, 28, 28, 1024)	0	dropout[0][0], conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 28, 28, 512)	4,719,104	concatenate[0][0]
conv2d_12 (Conv2D)	(None, 28, 28, 512)	2,359,808	conv2d_11[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 56, 56, 512)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 56, 56, 256)	524,544	up_sampling2d_1[0][0]
concatenate_1	(None, 56, 56, 512)	0	conv2d_5[0][0],

up_sampling2d_1 (UpSampling2D)	(None, 56, 56, 512)	0	conv2d_12[0][0]
conv2d_13 (Conv2D)	(None, 56, 56, 256)	524,544	up_sampling2d_1[0][0]
concatenate_1 (Concatenate)	(None, 56, 56, 512)	0	conv2d_5[0][0], conv2d_13[0][0]
conv2d_14 (Conv2D)	(None, 56, 56, 256)	1,179,904	concatenate_1[0][0]
conv2d_15 (Conv2D)	(None, 56, 56, 256)	590,080	conv2d_14[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 112, 112, 256)	0	conv2d_15[0][0]
conv2d_16 (Conv2D)	(None, 112, 112, 128)	131,200	up_sampling2d_2[0][0]
concatenate_2 (Concatenate)	(None, 112, 112, 256)	0	conv2d_3[0][0], conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 112, 112, 128)	295,040	concatenate_2[0][0]
conv2d_18 (Conv2D)	(None, 112, 112, 128)	147,584	conv2d_17[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 224, 224, 128)	0	conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 224, 224, 64)	32,832	up_sampling2d_3[0][0]
concatenate_3 (Concatenate)	(None, 224, 224, 128)	0	conv2d_1[0][0], conv2d_19[0][0]
conv2d_20 (Conv2D)	(None, 224, 224, 64)	73,792	concatenate_3[0][0]
conv2d_21 (Conv2D)	(None, 224, 224, 64)	36,928	conv2d_20[0][0]
conv2d_22 (Conv2D)	(None, 224, 224, 2)	1,154	conv2d_21[0][0]
conv2d_23 (Conv2D)	(None, 224, 224, 21)	63	conv2d_22[0][0]

Total params: 31,032,897 (118.38 MB)
Trainable params: 31,032,897 (118.38 MB)
Non-trainable params: 0 (0.00 B)

```
model.fit(train_ds, validation_data=eval_ds, epochs=10, callbacks=[checkpoint, early_stopping, reduce_lr])

Epoch 1/10
    265/Unknown 728s 3s/step - accuracy: 0.6780 - jaccard_coef: 0.5107 - loss: 1.0217/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure th
  self.gen.throw(typ, value, traceback)

Epoch 1: val_loss improved from inf to 1.01886, saving model to unet_model.weights.h5
265/265 ━━━━━━━━━━━━━━━━━━━━ 775s 3s/step - accuracy: 0.6780 - jaccard_coef: 0.5107 - loss: 1.0217 - val_accuracy: 0.6945 - val_jaccard_coef: 0.5337 - val_loss: 1.0189 - learning_rate: 0.0010
Epoch 2/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6849 - jaccard_coef: 0.5223 - loss: 1.0209
Epoch 2: val_loss improved from 1.01886 to 1.01883, saving model to unet_model.weights.h5
265/265 ━━━━━━━━━━━━━━━━━━━━ 793s 3s/step - accuracy: 0.6849 - jaccard_coef: 0.5223 - loss: 1.0209 - val_accuracy: 0.6946 - val_jaccard_coef: 0.5335 - val_loss: 1.0188 - learning_rate: 0.0010
Epoch 3/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6843 - jaccard_coef: 0.5216 - loss: 1.0210
Epoch 3: val_loss did not improve from 1.01883
265/265 ━━━━━━━━━━━━━━━━━━━━ 790s 3s/step - accuracy: 0.6843 - jaccard_coef: 0.5216 - loss: 1.0210 - val_accuracy: 0.6944 - val_jaccard_coef: 0.5333 - val_loss: 1.0189 - learning_rate: 0.0010
Epoch 4/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6891 - jaccard_coef: 0.5273 - loss: 1.0199
Epoch 4: val_loss did not improve from 1.01883

Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
265/265 ━━━━━━━━━━━━━━━━━━━━ 808s 3s/step - accuracy: 0.6891 - jaccard_coef: 0.5272 - loss: 1.0199 - val_accuracy: 0.6944 - val_jaccard_coef: 0.5338 - val_loss: 1.0189 - learning_rate: 0.0010
Epoch 5/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6841 - jaccard_coef: 0.5213 - loss: 1.0211
Epoch 5: val_loss improved from 1.01883 to 1.01880, saving model to unet_model.weights.h5
265/265 ━━━━━━━━━━━━━━━━━━━━ 815s 3s/step - accuracy: 0.6841 - jaccard_coef: 0.5213 - loss: 1.0211 - val_accuracy: 0.6946 - val_jaccard_coef: 0.5331 - val_loss: 1.0188 - learning_rate: 2.0000e-04
Epoch 6/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6866 - jaccard_coef: 0.5243 - loss: 1.0205
Epoch 6: val_loss did not improve from 1.01880
265/265 ━━━━━━━━━━━━━━━━━━━━ 810s 3s/step - accuracy: 0.6866 - jaccard_coef: 0.5243 - loss: 1.0205 - val_accuracy: 0.6944 - val_jaccard_coef: 0.5334 - val_loss: 1.0189 - learning_rate: 2.0000e-04
Epoch 7/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6859 - jaccard_coef: 0.5235 - loss: 1.0207
Epoch 7: val_loss did not improve from 1.01880

Epoch 7: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
265/265 ━━━━━━━━━━━━━━━━━━━━ 844s 3s/step - accuracy: 0.6859 - jaccard_coef: 0.5235 - loss: 1.0207 - val_accuracy: 0.6944 - val_jaccard_coef: 0.5335 - val_loss: 1.0189 - learning_rate: 2.0000e-04
Epoch 8/10
265/265 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.6871 - jaccard_coef: 0.5249 - loss: 1.0204
Epoch 8: val_loss did not improve from 1.01880
265/265 ━━━━━━━━━━━━━━━━━━━━ 755s 3s/step - accuracy: 0.6871 - jaccard_coef: 0.5249 - loss: 1.0204 - val_accuracy: 0.6944 - val_jaccard_coef: 0.5332 - val_loss: 1.0189 - learning_rate: 4.0000e-05
Epoch 8: early stopping
Restoring model weights from the end of the best epoch: 5.
<keras.src.callbacks.history.History at 0x7b62d042a9b0>
```

بخش دوم:

در این بخش ابتدا یک مدل Unet با وزن‌های اولیهٔ انکودر imagenet با اکیتویشن فانکشن softmax و ترکیب توابع ضرر DiceLoss و CategoricalFocalLoss تعریف می‌شود. سپس اتریبیوت trainable را برای لایه‌های مربوط به encoder برابر false و برای لایه‌های دیکودر برابر true قرار می‌دهیم تا انکودر فریز شده و فقط انکودر آموزش ببیند. سپس وزن‌های به دست آمده را نگه داشته و همهٔ لایه‌ها را دوباره قابل آموزش می‌کنیم تا از آخرین وزن به دست آمده شروع کند و تنظیم دقیق انجام دهد (tuning).

```
activation='softmax'

LR = 0.001
optim = keras.optimizers.Adam(LR)


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

```
BACKBONE1 = 'mobilenetv2'

n_classes=21
# define model
model1 = sm.Unet(BACKBONE1, encoder_weights='imagenet', classes=n_classes, activation=activation)
model1.compile(optim, total_loss, metrics=metrics)
print(model1.summary())
```

decoder_stage3b_conv (Conv2D)	(None, None, None, 32)	9,216	decoder_stage3a_relu[…
decoder_stage3b_bn (BatchNormalization)	(None, None, None, 32)	128	decoder_stage3b_conv[…
decoder_stage3b_relu (Activation)	(None, None, None, 32)	0	decoder_stage3b_bn[0]…
decoder_stage4_upsampling (UpSampling2D)	(None, None, None, 32)	0	decoder_stage3b_relu[…
decoder_stage4a_conv (Conv2D)	(None, None, None, 16)	4,608	decoder_stage4_upsamp…
decoder_stage4a_bn (BatchNormalization)	(None, None, None, 16)	64	decoder_stage4a_conv[…
decoder_stage4a_relu (Activation)	(None, None, None, 16)	0	decoder_stage4a_bn[0]…
decoder_stage4b_conv (Conv2D)	(None, None, None, 16)	2,304	decoder_stage4a_relu[…
decoder_stage4b_bn (BatchNormalization)	(None, None, None, 16)	64	decoder_stage4b_conv[…
decoder_stage4b_relu (Activation)	(None, None, None, 16)	0	decoder_stage4b_bn[0]…
final_conv (Conv2D)	(None, None, None, 21)	3,045	decoder_stage4b_relu[…
softmax (Activation)	(None, None, None, 21)	0	final_conv[0][0]

Total params: 8,050,341 (30.71 MB)
Trainable params: 8,014,245 (30.57 MB)
Non-trainable params: 36,096 (141.00 KB)

```
flag = False
for l in model1.layers:
    if l.name == 'decoder_stage0_upsampling':
        flag = True
    l.trainable = flag
```

```
model1.summary()
```

(Activation)			
decoder_stage3b_conv (Conv2D)	(None, None, None, 32)	9,216	decoder_stage3a_relu[…
decoder_stage3b_bn (BatchNormalization)	(None, None, None, 32)	128	decoder_stage3b_conv[…
decoder_stage3b_relu (Activation)	(None, None, None, 32)	0	decoder_stage3b_bn[0]…
decoder_stage4_upsampling (UpSampling2D)	(None, None, None, 32)	0	decoder_stage3b_relu[…
decoder_stage4a_conv (Conv2D)	(None, None, None, 16)	4,608	decoder_stage4_upsamp…
decoder_stage4a_bn (BatchNormalization)	(None, None, None, 16)	64	decoder_stage4a_conv[…
decoder_stage4a_relu (Activation)	(None, None, None, 16)	0	decoder_stage4a_bn[0]…
decoder_stage4b_conv (Conv2D)	(None, None, None, 16)	2,304	decoder_stage4a_relu[…
decoder_stage4b_bn (BatchNormalization)	(None, None, None, 16)	64	decoder_stage4b_conv[…
decoder_stage4b_relu (Activation)	(None, None, None, 16)	0	decoder_stage4b_bn[0]…
final_conv (Conv2D)	(None, None, None, 21)	3,045	decoder_stage4b_relu[…
softmax (Activation)	(None, None, None, 21)	0	final_conv[0][0]

Total params: 8,050,341 (30.71 MB)
Trainable params: 5,790,373 (22.09 MB)
Non-trainable params: 2,259,968 (8.62 MB)

```
#let all the layers be trained
for layer in model1.layers:
    layer.trainable = True
```

```
LR = 0.000005
optim = keras.optimizers.Adam(LR)
```

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

```
model1.summary()
```

decoder_stage3b_conv (Conv2D)	(None, None, None, 32)	9,216	decoder_stage3a_relu[…
decoder_stage3b_bn (BatchNormalization)	(None, None, None, 32)	128	decoder_stage3b_conv[…
decoder_stage3b_relu (Activation)	(None, None, None, 32)	0	decoder_stage3b_bn[0]…
decoder_stage4_upsampling (UpSampling2D)	(None, None, None, 32)	0	decoder_stage3b_relu[…
decoder_stage4a_conv (Conv2D)	(None, None, None, 16)	4,608	decoder_stage4_upsamp…
decoder_stage4a_bn (BatchNormalization)	(None, None, None, 16)	64	decoder_stage4a_conv[…
decoder_stage4a_relu (Activation)	(None, None, None, 16)	0	decoder_stage4a_bn[0]…
decoder_stage4b_conv (Conv2D)	(None, None, None, 16)	2,304	decoder_stage4a_relu[…
decoder_stage4b_bn (BatchNormalization)	(None, None, None, 16)	64	decoder_stage4b_conv[…
decoder_stage4b_relu (Activation)	(None, None, None, 16)	0	decoder_stage4b_bn[0]…
final_conv (Conv2D)	(None, None, None, 21)	3,045	decoder_stage4b_relu[…
softmax (Activation)	(None, None, None, 21)	0	final_conv[0][0]

Total params: 8,050,341 (30.71 MB)
Trainable params: 8,014,245 (30.57 MB)
Non-trainable params: 36,096 (141.00 KB)

```
model1.fit(train_ds, validation_data=eval_ds, epochs=5, batch_size=32 )
```

```
Epoch 1/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 586s 2s/step - accuracy: 0.6554 - jaccard_coef: 0.4740 - loss: 0.8499 - val_accuracy: 0.6462 - val_jaccard_coef: 0.4622 - val_loss: 0.8033
Epoch 2/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 526s 2s/step - accuracy: 0.7116 - jaccard_coef: 0.5359 - loss: 0.7602 - val_accuracy: 0.7067 - val_jaccard_coef: 0.5292 - val_loss: 0.7417
Epoch 3/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 568s 2s/step - accuracy: 0.7370 - jaccard_coef: 0.5660 - loss: 0.7174 - val_accuracy: 0.7415 - val_jaccard_coef: 0.5714 - val_loss: 0.6978
Epoch 4/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 524s 2s/step - accuracy: 0.7500 - jaccard_coef: 0.5822 - loss: 0.6989 - val_accuracy: 0.7631 - val_jaccard_coef: 0.5988 - val_loss: 0.6724
Epoch 5/5
265/265 ━━━━━━━━━━━━━━━━━━━━ 526s 2s/step - accuracy: 0.7579 - jaccard_coef: 0.5921 - loss: 0.6750 - val_accuracy: 0.7749 - val_jaccard_coef: 0.6145 - val_loss: 0.6475
<keras.src.callbacks.history.History at 0x7df45cc9be80>
```

سوال ششم).....

تابع preprocess دیتاست را به عنوان ورودی گرفته و تصاویر، bounding box ها و لیبل‌های آن را جدا می‌کند.

تابع download_dataset، دیتاست را از سرور دانلود کرده و دیرکتوری dataset/. ذخیره می‌کند.

```
def preprocess(dataset):
    images = []
    bboxes = []
    labels = []
    for img, target in dataset:
        images.append(img)
        annotations = target['annotation']['object']

        if not isinstance(annotations, list):
            annotations = [annotations]

        for annotation in annotations:
            bboxes.append(annotation['bndbox'])
            labels.append(annotation['name'])

    return images, bboxes, labels
```

```
def download_dataset():
    root_dir = './dataset'
    dataset = torchvision.datasets.VOCDetection(root_dir, year='2012', image_set='train', download=True)
    data_loader = data.DataLoader(dataset, batch_size=32, shuffle=True)
    images, bboxes, labels = preprocess(dataset)
    return images, bboxes, labels

images, bboxes, labels = download_dataset()
```

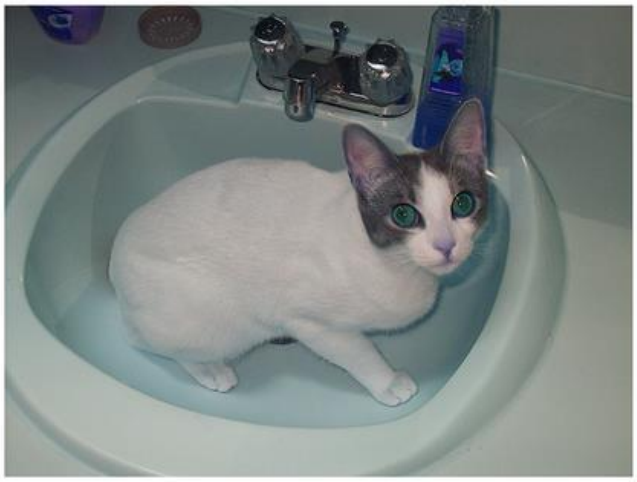
```
Using downloaded and verified file: ./dataset/VOCtrainval_11-May-2012.tar
Extracting ./dataset/VOCtrainval_11-May-2012.tar to ./dataset
```

```
images[25]
```

نمونهٔ تصویر:

```python
def visualize_image_with_bboxes(image, bboxes, labels=None):
    """
    نمایش یک تصویر با جعبه‌های مرزی.

    ورودی‌ها:
    - image (np.array or str or PIL.Image.Image): تصویر به عنوان آرایه numpy یا مسیر فایل تصویر یا تصویر PIL.
    - bboxes (list of list): لیست جعبه‌های مرزی، هرکدام به صورت [xmin, ymin, xmax, ymax].
    - labels (list of str, اختیاری): لیست برچسب‌ها که متناظر با جعبه‌های مرزی هستند.

    خروجی‌ها:
    - هیچکدام: تصویر با جعبه‌های مرزی نمایش داده می‌شود.
    """
    if isinstance(image, str):
        image = cv2.imread(image)
        if image is None:
            raise ValueError(f"Image at path {image} could not be loaded.")

    # convert PIL Image to numpy array if necessary
    if isinstance(image, Image.Image):
        image = np.array(image)

    # ensure image is a numpy array
    if not isinstance(image, np.ndarray):
        raise TypeError(f"Image must be a numpy array, PIL Image, or a valid path to an image. Got {type(image)} instead.")

    for i, bbox in enumerate(bboxes):
        if all(isinstance(coord, (int, float)) for coord in bbox):
            bbox = [int(coord) for coord in bbox]
            cv2.rectangle(image, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (0, 255, 0), 2)
            if labels:
                cv2.putText(image, labels[i], (bbox[0], bbox[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        else:
            print(f"Skipping bbox {bbox} due to non-numerical values.")

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

# نمایش نمونه‌ای از داده‌ها
visualize_image_with_bboxes(images[25], bboxes[25], labels[25])
```

```
Skipping bbox xmin due to non-numerical values.
Skipping bbox ymin due to non-numerical values.
Skipping bbox xmax due to non-numerical values.
Skipping bbox ymax due to non-numerical values.
```



استفاده از مدل ResNet برای این کار:

```
# مرحله 4: آماده‌سازی مدل
#  از مدل زو TensorFlow (Fast R-CNN شود استفاده مشابه روشی به (می‌تواند).

# TODO: مدل یک بارگذاری برای تابع نوشتن Faster R-CNN دیده پیش‌آموزش
def load_pretrained_model():
    """
    مدل یک بارگذاری Faster R-CNN زو مدل از دیده پیش‌آموزش TensorFlow.

    خروجی‌ها:
    - model: مدل Faster R-CNN دیده پیش‌آموزش.
    """
    # دانشجویان باید این قسمت را تکمیل کنند
    model = tf.keras.applications.ResNet50(include_top=False, input_shape=(None, None, 3))
    return model

model = load_pretrained_model()
```

تابع preprocess_image برای تبدیل عکس ورودی به input مناسب مدل:

```
def preprocess_image(image_path):
    """
    مدل برای تصویر یک پیش‌پردازش.

    ورودی‌ها:
    - image_path (str): تصویر فایل مسیر.

    خروجی‌ها:
    - input_image (آرایه numpy): مدل ورودی برای شده آماده پیش‌پردازش تصویر.
    - original_image (آرایه numpy): نمایش برای اصلی تصویر.
    """
    # Load the image
    original_image = cv2.imread(image_path)
    if original_image is None:
        raise ValueError(f"Image at path {image_path} could not be loaded.")

    input_image = cv2.resize(original_image, (224, 224))

    input_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB)

    input_image = input_image / 255.0

    input_image = np.expand_dims(input_image, axis=0)

    return input_image, original_image

# Example usage
image_path = './dataset/VOCdevkit/VOC2012/JPEGImages/2007_000063.jpg'
input_image, original_image = preprocess_image(image_path)
```

تابع detect_object (درست کار نمی‌کند☹):

```python
def detect_objects(model, image_path, threshold=0.5):
    input_image, original_image = preprocess_image(image_path)
    input_tensor = tf.convert_to_tensor(input_image, dtype=tf.float32)

    detections = model(input_tensor)
    print(detections)
    boxes = detections['detection_boxes'][0].numpy()
    scores = detections['detection_scores'][0].numpy()
    classes = detections['detection_classes'][0].numpy().astype(int)

    height, width, _ = original_image.shape
    valid_detections = scores >= threshold
    boxes = boxes[valid_detections]
    scores = scores[valid_detections]
    classes = classes[valid_detections]

    boxes = np.array([[int(box[0]*height), int(box[1]*width), int(box[2]*height), int(box[3]*width)] for box in boxes])

    for box, score, cls in zip(boxes, scores, classes):
        ymin, xmin, ymax, xmax = box
        cv2.rectangle(original_image, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)
        label_text = f'{cls}: {score:.2f}'
        cv2.putText(original_image, label_text, (xmin, ymin - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

# Load a pre-trained model (replace with actual Fast R-CNN model)
model = load_pretrained_model()

# Use a sample image path
image_path = './dataset/VOCdevkit/VOC2012/JPEGImages/2007_000063.jpg'
detect_objects(model, image_path)
```

منابع:

- **Link1**: Fast RCNN
- **Link2**: Fast RCNN