به نام خالق رنگین کمان

ستاره باباجانی – گزارش تمرین سری هفتم

سوال 1: الف)

Dilated Kernel Size = k + (k - 1) * (d - 1)

ب) تعداد پارامترهای قابل آموزش در یک لایه کانولوشن توسط اندازه کرنل و تعداد کانال های ورودی و خروجی تعیین می شود، نه توسط dilation rate. سه برابر شدن این نرخ بر receptive field تأثیر می گذارد اما تعداد پارامترهای قابل آموزش را تغییر نمی دهد.

ج) همانطور که میدانیم فرمول receptive field برای هر لایه به شرح زیر است:

RF0 = 1, RFi = RFi-1 + (k - 1) * d

پس جدول به شرح زیر میشود:

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
Dilation 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

د) اگر فرض کنیم pool size و stride یکی باشند، در لایه کانولوشنی خواهیم داشت:

Receptive_out = Receptive_in + (kernal_size - 1) * Dilation_rate

و در لایه max pooling خواهیم داشت:

Receptive_out = Receptive_in + (stride - 1) * Stride_Of_Previous_Layer (call J)

پس برای لایه‌ها به ترتیب خواهیم داشت:

1. First conv: = 1 + (5 - 1) * 1 = 5
2. Second conv: = 5 + (5 - 1) * 1 = 9 => so the receptive field is 9*9
3. First max-pooling: = 9 + (s - 1) * 1 = 8 + s
4. Second max-pooling: J = 1 * s, so: 8 + s + (s - 1) * s = 8 + s^2
5. Third max-pooling: J = s * s, so: 8 + s^2 + (s - 1) * (s^2) = 8 + s^3
   So => 8 + s^3 >= 107, s = 5

سوال 2: الف) در کانولوشن معمولی داریم:

- Number of parameters = kernel_size * kernel_ size * input_channels * output_channels

  So: 5 * 5 * 3 * 64 = 4800

- Operations per filter = input_width * input_height * kernel_size * input_channels
  So: 3 * 64 * 5 * 5 * 128 * 128 = 78,643,200

در Depthwise Separable Convolution داریم:

- Number of parameters (depthwise) = kernel_size * kernel_size * input_channels
  So: 5 * 5 * 3 = 75
  Number of parameters (pointwise) = 1 * 1 * input_channels * output channels
  So: 1 * 1 * 3 * 64 = 192
  Total: 75 + 192 = 267
- Depthwise Convolution Operations = output_size * operations_of_each_output_pixel
  So: 128 * 128 * 75 = 1,228,800
  Pointwise Convolution Operations = output_size * operations_of_each_output_pixel
  So: 128 * 128 * 3 * 64 = 3,145,728
  Total: 1,228,800 + 3,145,728 = 4,374,528

که همانطور که مشاهده میشود Depthwise separable convolution بطور قابل توجهی تعداد پارامترها و تعداد عملیات ضرب را کاهش می دهد.

ب) در این بخش تعداد پارامتر هرکدام را طبق فرمول‌های بالا محاسبه کرده و سپس نسبت میگیریم:

- Regular convolution: 3 * 3 * 32 * 32 = 9216
- Depthwise Separable Convolution = 3 * 3 * 32 + 1 * 1 * 32 * 32 = 1312

که نسبت آنها برابر با 0.142 میشود. (1312 / 9216)

سوال 3: الف) جواب متد دوم که NCC است، است. همانطور که مشاهده میشود در متد اول مقادیر نرمالیزه نمیشوند اما در متد دوم، پیکسل‌های تصویر و کلیشه نرمالیزه میشوند که این به میزان صحت و robustness مدل می‌افزاید. همچنین طبق مقادیر پیکسل‌های تصویر، بازه گسترده‌ای از مقادیر داریم پس استفاده از این متد، بهتر است.

ب) مراحل کد زده شده به شرح زیر است:

1. نصب کتابخانه مورد استفاده: همانطور که مشاهده میشود از کتابخانه mtm استفاده شده است.

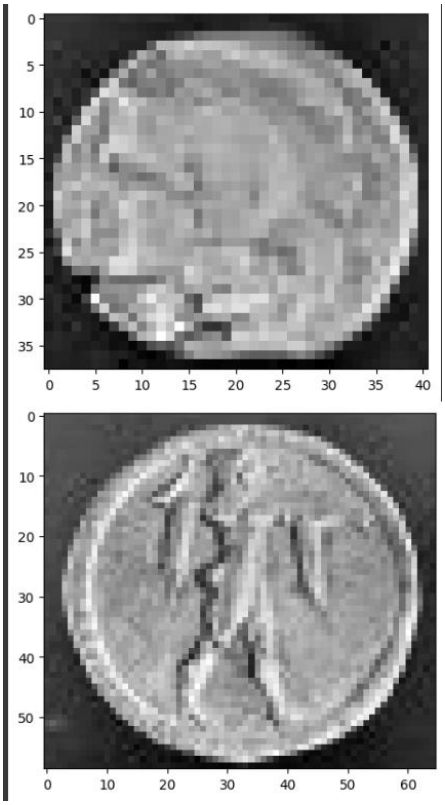
```
# Importing the mtm module and printing its version.
import mtm
print("mtm version: ", mtm.__version__)

# Importing specific functions from the mtm module.
from mtm import matchTemplates
from mtm.detection import plotDetections

# enabling inline plotting with matplotlib.
%matplotlib inline

# Importing the matplotlib.pyplot module, which is used for plotting and visualizing data.
import matplotlib.pyplot as plt
```

2. نمایش عکس داده شده:

```python
# Importing necessary libraries
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

# Define the path to the image
image_path = os.path.join('/content', 'coins.png')

# Load the image
image = mpimg.imread(image_path)

# Display the image in grayscale
plt.imshow(image, cmap='gray')
plt.axis('off')  # Hide the axis
plt.show()
```

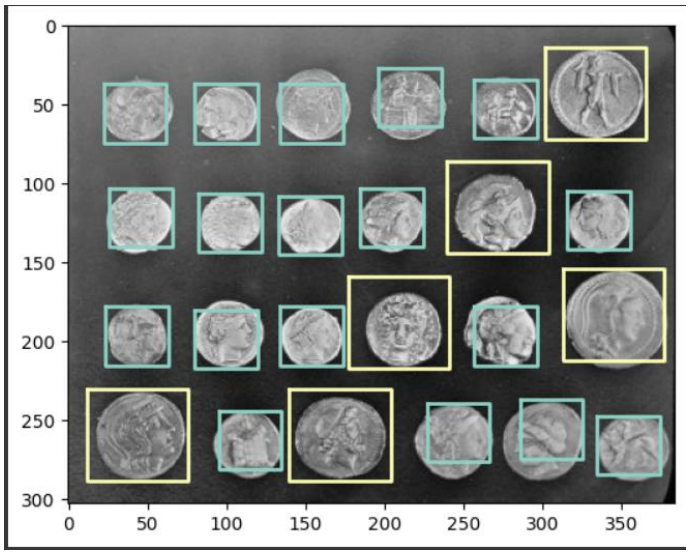
3. ساخت تصویر الگو: برای جستجو در تصویر به تعدادی تصویر الگو نیاز داریم که در کد زیر دوتا از سکه‌ها (یکی بزرگ و دیگری کوچک) بعنوان الگو درنظر گرفتیم:

```python
small_Coin = image[37:37+38, 80:80+41]
large_Coin = image[14:14+59,302:302+65]
plt.figure(0)
plt.imshow(small_Coin, cmap="gray")
plt.figure(1)
plt.imshow(large_Coin, cmap="gray")
```



4. حال با استفاده از تابع matchTemplates و تصاویر الگو و مقدار threshold کل سکه‌های تصویر داده شده را می‌یابیم:

```
listTemplates = [small_Coin, large_Coin]
listDetections = matchTemplates(image, listTemplates, scoreThreshold=0.4, maxOverlap=0)
plotDetections(image, listDetections)
```



سوال 4: نوتبوک داده شده تکمیل شد.

سوال 5: مراحل کد خواسته شده به شرح زیر است:

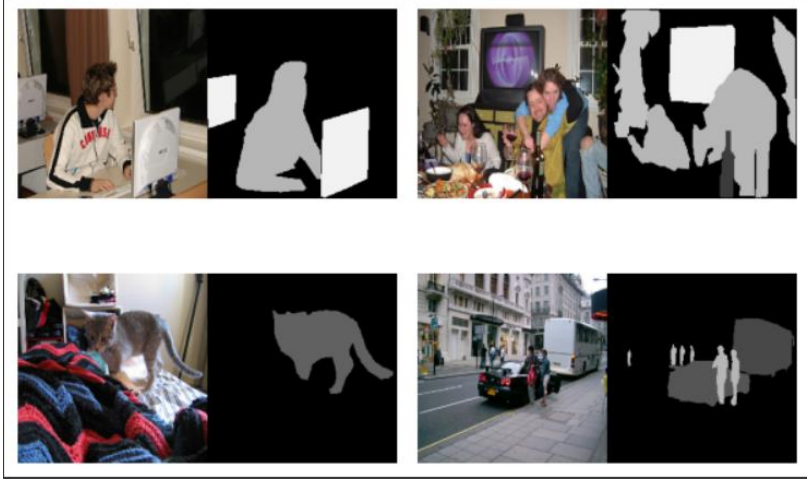
- ابتدا کتابخانه‌ها و پکیج‌های مورد نیاز را نصب کردیم.
- دیتاست PasCal را لود کردیم:

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

- پیش پردازش دیتا: با resize و batch کردن:

```
# Preprocess the dataset
def preprocess_tfds_inputs(inputs):
    def unpackage_tfds_inputs(tfds_inputs):
        return {
            "images": tfds_inputs["image"],
            "segmentation_masks": tfds_inputs["class_segmentation"],
        }

    outputs = inputs.map(unpackage_tfds_inputs)
    outputs = outputs.map(keras_cv.layers.Resizing(height=224, width=224))
    outputs = outputs.batch(32, drop_remainder=True)
    return outputs
```

- نشان دادن نمونه دیتا از دیتاست:



- چند تکنیک data augmentation مثل random flipping و random rotation روی داده‌ها انجام شد:

```python
# Data augmentation
train_ds = train_ds.map(keras_cv.layers.RandomFlip())
train_ds = train_ds.map(keras_cv.layers.RandomRotation(factor=.1,segmentation_classes=21))

batch = train_ds.take(1).get_single_element()

keras_cv.visualization.plot_segmentation_mask_gallery(
    batch["images"],
    value_range=(0, 255),
    num_classes=21,
    y_true=batch["segmentation_masks"],
    scale=3,
    rows=2,
    cols=2,
)
```

- طراحی شبکه Unet از صفر: مدل شامل encoder که از چندین بلوک شامل دو لایه کانولوشن و به دنبال آن یک لایه حداکثر ادغام و decoder که رمزگذار را منعکس می کند اما حداکثر ادغام را با unsampling جایگزین می کند و bridge برای اتصال آن دو، تشکیل شده است.
- تعریف توابع ضرر:

```python
# Define losses and metrics
dice_loss = sm.losses.DiceLoss()
focal_loss = sm.losses.CategoricalFocalLoss()
total_loss = dice_loss + (1 * focal_loss)
```

- تعریف متریک جدید: یک معیار دلخواه برای ارزیابی عملکرد یک مدل تقسیم‌بندی تصویر است. این شاخص Jaccard را محاسبه می‌کند که به نام IoU نیز شناخته می‌شود، که یک معیار پرکاربرد در وظایف تقسیم‌بندی تصویر است.

```python
def jaccard_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (intersection + 1.0) / (K.sum(y_true_f) + K.sum(y_pred_f) - intersection + 1.0)
```

- کامپایل کردن مدل: خلاصه مدل در کدها آورده شده است.

```python
# Compile the model
metrics=['accuracy', jaccard_coef]
#compile the model
####################
#your code goes here
model.compile(optimizer='adam', loss=total_loss, metrics=metrics)
####################
```

- اضافه کردن call back برای ذخیره بهترین مدل:

```
# Callbacks to save the best model
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath='best_model_unet_scratch.keras',
        save_best_only=True,
        monitor='val_loss',
        mode='min'
    )
]
```

- سپس با مرحله بعدی از پیش پردازش داده، مطمئن میشویم که دیتا دارای فرمت درست برای مدل ما است و همچنین one-hot vectors میسازیم:

```
# Preprocess dataset for training
def dict_to_tuple(x):


    return x["images"], tf.one_hot(
        tf.cast(tf.squeeze(x["segmentation_masks"], axis=-1), "int32"), 21
    )


train_ds = train_ds.map(dict_to_tuple)
eval_ds = eval_ds.map(dict_to_tuple)
```

- آموزش مدل
- حال سراغ مدل از پیش آموزش دیده شده میرویم که رمزگذار mobilenetv2 دارد:

```
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())
```

- ابتدا encoder فریز شده و فقط decoder آموزش میبینند: خلاصه مدل در کد قابل مشاهده است.

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

- اضافه کردن call back برای ذخیره بهترین مدل:

```
# Callbacks to save the best model
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath='best_model_unet_pretrained.keras',
        save_best_only=True,
        monitor='val_loss',
        mode='min'
    )
]
```

- آموزش مدل
- پس از آموزش اولیه، تمام لایه‌ها منجمد می‌شوند و کل مدل با نرخ یادگیری پایین‌تر تنظیم می‌شود:

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

```
for l in model1.layers:
    l.trainable = True
```

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

- آموزش نهایی مدل