

## Homework 3

### Instructions

- This homework focuses on understanding and applying DETR for object detection and attention visualization. It consists of **three questions** designed to assess both theoretical understanding and practical application.
- Please organize your answers and results for the questions below and submit this jupyter notebook as a **.pdf file**.
- **Deadline: 11/14 (Thur) 23:59**

### Reference

- End-to-End Object Detection with Transformers (DETR):  
<https://github.com/facebookresearch/detr>

### Q1. Understanding DETR model

- Fill-in-the-blank exercise to test your understanding of critical parts of the DETR model workflow.

[ ] ↴ 숨겨진 셀 1개

### Q2. Custom Image Detection and Attention Visualization

In this task, you will upload an **image of your choice** (different from the provided sample) and follow the steps below:

- Object Detection using DETR
  - Use the DETR model to detect objects in your uploaded image.
- Attention Visualization in Encoder
  - Visualize the regions of the image where the encoder focuses the most.
- Decoder Query Attention in Decoder
  - Visualize how the decoder's query attends to specific areas corresponding to the detected objects.

```
1 import math
2
3 from PIL import Image
4 import requests
5 import matplotlib.pyplot as plt
```

```
6 %config InlineBackend.figure_format = 'retina'
7
8 import ipywidgets as widgets
9 from IPython.display import display, clear_output
10
11 import torch
12 from torch import nn
13
14
15 from torchvision.models import resnet50
16 import torchvision.transforms as T
17 torch.set_grad_enabled(False);
18
19 # COCO classes
20 CLASSES = [
21     'N/A', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
22     'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A',
23     'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse',
24     'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack',
25     'umbrella', 'N/A', 'N/A', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis',
26     'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove',
27     'skateboard', 'surfboard', 'tennis racket', 'bottle', 'N/A', 'wine glass',
28     'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich',
29     'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake',
30     'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table', 'N/A',
31     'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard',
32     'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A',
33     'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair dr ier',
34     'toothbrush'
35 ]
36
37 # colors for visualization
38 COLORS = [[0.000, 0.447, 0.741], [0.850, 0.325, 0.098], [0.929, 0.694, 0.125],
39             [0.494, 0.184, 0.556], [0.466, 0.674, 0.188], [0.301, 0.745, 0.933]]
40 # standard PyTorch mean-std input image normalization
41 transform = T.Compose([
42     T.Resize(800),
43     T.ToTensor(),
44     T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
45 ])
46
47 # for output bounding box post-processing
48 def box_cxcywh_to_xyxy(x):
49     x_c, y_c, w, h = x.unbind(1)
50     b = [(x_c - 0.5 * w), (y_c - 0.5 * h),
51           (x_c + 0.5 * w), (y_c + 0.5 * h)]
52     return torch.stack(b, dim=1)
53
54 def rescale_bboxes(out_bbox, size):
55     img_w, img_h = size
56     b = box_cxcywh_to_xyxy(out_bbox)
57     b = b * torch.tensor([img_w, img_h, img_w, img_h], dtype=torch.float32)
58     return b
59
60 def plot_results(pil_img, prob, boxes):
```

```

61     plt.figure(figsize=(16,10))
62     plt.imshow(pil_img)
63     ax = plt.gca()
64     colors = COLORS * 100
65     for p, (xmin, ymin, xmax, ymax), c in zip(prob, boxes.tolist(), colors):
66         ax.add_patch(plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,
67                                   fill=False, color=c, linewidth=3))
68         cl = p.argmax()
69         text = f'{CLASSES[cl]}: {p[cl]:0.2f}'
70         ax.text(xmin, ymin, text, fontsize=15,
71                 bbox=dict(facecolor='yellow', alpha=0.5))
72     plt.axis('off')
73     plt.show()
74
75

```

In this section, we show-case how to load a model from hub, run it on a custom image, and print the result. Here we load the simplest model (DETR-R50) for fast inference. You can swap it with any other model from the model zoo.

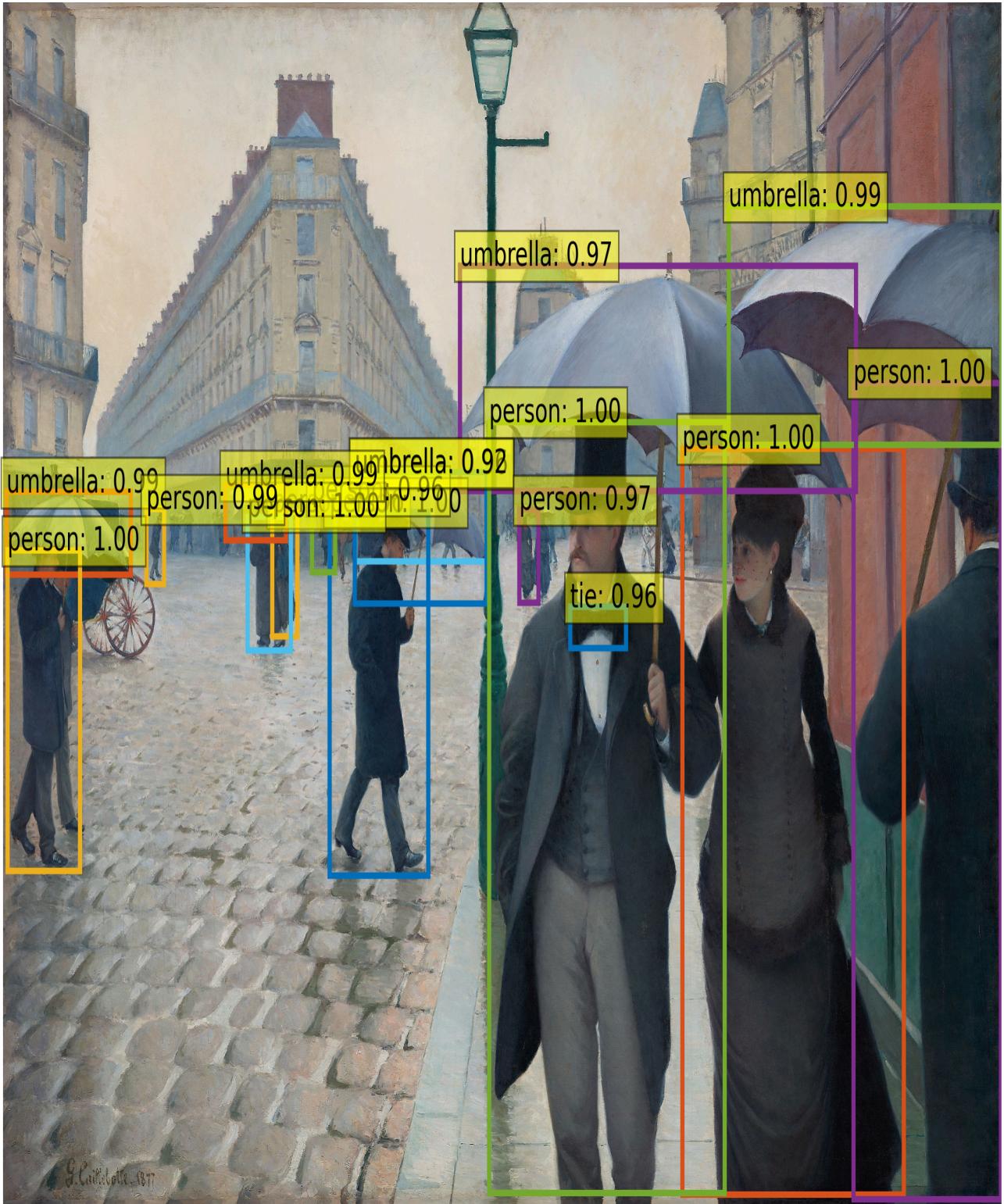
```

1 model = torch.hub.load('facebookresearch/detr', 'detr_resnet50', pretrained=True)
2 model.eval();
3
4 url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
5 im = Image.open("Gustave_Caillebotte_-_Paris_Street,_Rainy_Day_-_1964.336_-_Art_Institute_of_Chicago,_Chicago,_Illinois,_USA.jpg")
6 # mean-std normalize the input image (batch-size: 1)
7 img = transform(im).unsqueeze(0)
8
9 # propagate through the model
10 outputs = model(img)
11
12 # keep only predictions with 0.7+ confidence
13 probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
14 keep = probas.max(-1).values > 0.9
15
16 # convert boxes from [0; 1] to image scales
17 bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)
18
19 # mean-std normalize the input image (batch-size: 1)
20 img = transform(im).unsqueeze(0)
21
22 # propagate through the model
23 outputs = model(img)
24
25 # keep only predictions with 0.7+ confidence
26 probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
27 keep = probas.max(-1).values > 0.9
28
29 # convert boxes from [0; 1] to image scales
30 bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)
31
32 # mean-std normalize the input image (batch-size: 1)

```

```
33 img = transform(im).unsqueeze(0)
34
35 # propagate through the model
36 outputs = model(img)
37
38 # keep only predictions with 0.7+ confidence
39 probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
40 keep = probas.max(-1).values > 0.9
41
42 # convert boxes from [0; 1] to image scales
43 bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)
44
45 plot_results(im, probas[keep], bboxes_scaled)
```

→ Using cache found in /root/.cache/torch/hub/facebookresearch\_detr\_main



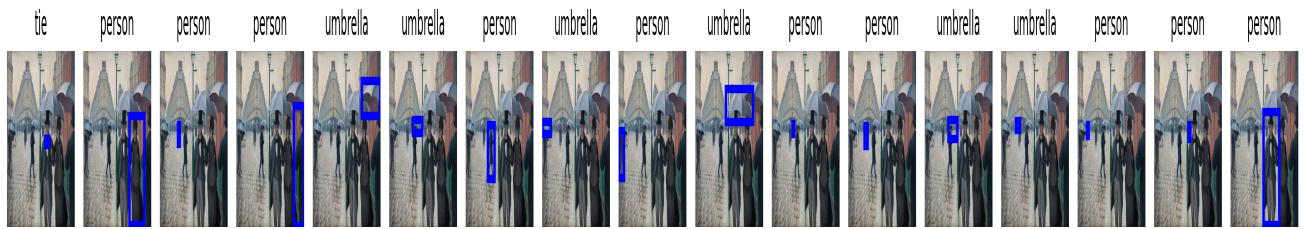
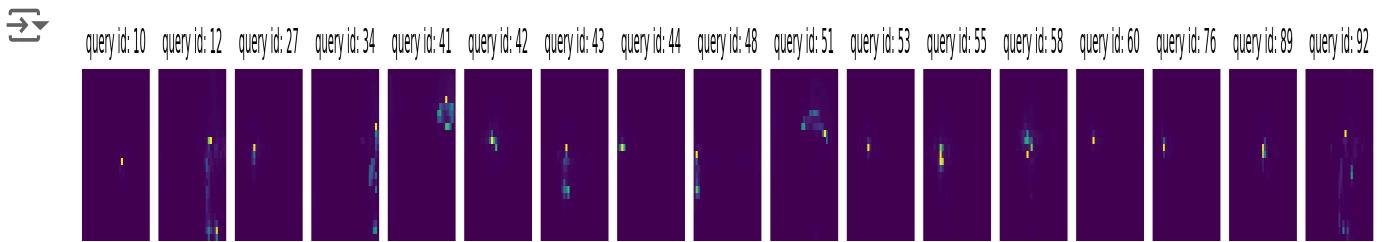
Here we visualize attention weights of the last decoder layer. This corresponds to visualizing, for each detected objects, which part of the image the model was looking at to predict this specific bounding box and class.

```

1 # use lists to store the outputs via up-values
2 conv_features, enc_attn_weights, dec_attn_weights = [], [], []
3
4 hooks = [

```

```
5     model.backbone[-2].register_forward_hook(  
6         lambda self, input, output: conv_features.append(output)  
7     ),  
8     model.transformer.encoder.layers[-1].self_attn.register_forward_hook(  
9         lambda self, input, output: enc_attn_weights.append(output[1])  
10    ),  
11    model.transformer.decoder.layers[-1].multihead_attn.register_forward_hook(  
12        lambda self, input, output: dec_attn_weights.append(output[1])  
13    ),  
14 ]  
15  
16 # propagate through the model  
17 outputs = model(img)  
18  
19 for hook in hooks:  
20     hook.remove()  
21  
22 # don't need the list anymore  
23 conv_features = conv_features[0]  
24 enc_attn_weights = enc_attn_weights[0]  
25 dec_attn_weights = dec_attn_weights[0]  
  
1 # get the feature map shape  
2 h, w = conv_features['0'].tensors.shape[-2:]  
3  
4 fig, axs = plt.subplots(ncols=len(bboxes_scaled), nrows=2, figsize=(22, 7))  
5 colors = COLORS * 100  
6 for idx, ax_i, (xmin, ymin, xmax, ymax) in zip(keep.nonzero(), axs.T, bboxes_scaled):  
7     ax = ax_i[0]  
8     ax.imshow(dec_attn_weights[0, idx].view(h, w))  
9     ax.axis('off')  
10    ax.set_title(f'query id: {idx.item()}')  
11    ax = ax_i[1]  
12    ax.imshow(im)  
13    ax.add_patch(plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,  
14                                fill=False, color='blue', linewidth=3))  
15    ax.axis('off')  
16    ax.set_title(CLASSES[probas[idx].argmax()])  
17 fig.tight_layout()
```



```

1 # output of the CNN
2 f_map = conv_features['0']
3 print("Encoder attention:      ", enc_attn_weights[0].shape)
4 print("Feature map:            ", f_map.tensors.shape)

```

→ Encoder attention: torch.Size([825, 825])  
 Feature map: torch.Size([1, 2048, 25, 33])

```

1 # get the HxW shape of the feature maps of the CNN
2 shape = f_map.tensors.shape[-2:]
3 # and reshape the self-attention to a more interpretable shape
4 sattn = enc_attn_weights[0].reshape(shape + shape)
5 print("Reshaped self-attention:", sattn.shape)

```

→ Reshaped self-attention: torch.Size([25, 33, 25, 33])

```

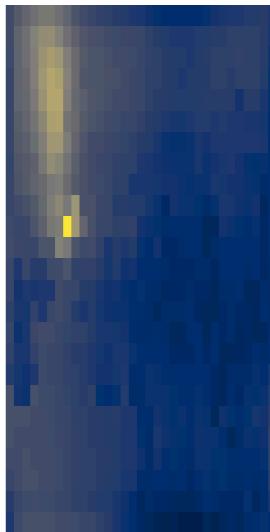
1 # downsampling factor for the CNN, is 32 for DETR and 16 for DETR DC5
2 fact = 32
3
4 # let's select 4 reference points for visualization
5 idxs = [(200, 200), (280, 400), (200, 600), (440, 800),]
6
7 # here we create the canvas
8 fig = plt.figure(constrained_layout=True, figsize=(25 * 0.7, 8.5 * 0.7))
9 # and we add one plot per reference point
10 gs = fig.add_gridspec(2, 4)
11 axs = [
12     fig.add_subplot(gs[0, 0]),
13     fig.add_subplot(gs[1, 0]),
14     fig.add_subplot(gs[0, -1]),
15     fig.add_subplot(gs[1, -1]),

```

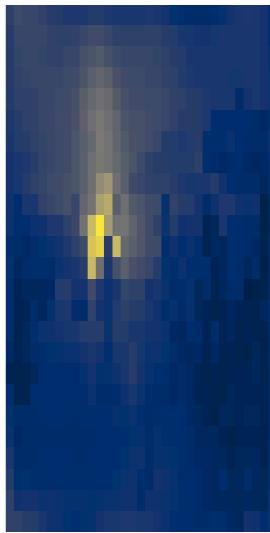
```
16 ]
17
18 # for each one of the reference points, let's plot the self-attention
19 # for that point
20 for idx_o, ax in zip(idxs, axs):
21     idx = (idx_o[0] // fact, idx_o[1] // fact)
22     ax.imshow(sattn[..., idx[0], idx[1]], cmap='cividis', interpolation='nearest')
23     ax.axis('off')
24     ax.set_title(f'self-attention{idx_o}')
25
26 # and now let's add the central image, with the reference points as red circles
27 fcenter_ax = fig.add_subplot(gs[:, 1:-1])
28 fcenter_ax.imshow(im)
29 for (y, x) in idxs:
30     scale = im.height / img.shape[-2]
31     x = ((x // fact) + 0.5) * fact
32     y = ((y // fact) + 0.5) * fact
33     fcenter_ax.add_patch(plt.Circle((x * scale, y * scale), fact // 2, color='r'))
34     fcenter_ax.axis('off')
```



self-attention(200, 200)



self-attention(280, 400)



self-attention(200, 600)



self-attention(440, 800)



### ✓ Q3. Understanding Attention Mechanisms