

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.

```
1 from torch import nn
2 class DETR(nn.Module):
3     def __init__(self, num_classes, hidden_dim=256, nheads=8,
4                  num_encoder_layers=6, num_decoder_layers=6, num_queries=100):
5         super().__init__()
6
7         # create ResNet-50 backbone
8         self.backbone = resnet50()
9         del self.backbone.fc
10
11        # create conversion layer
12        self.conv = nn.Conv2d(2048, hidden_dim, 1)
13
14        # create a default PyTorch transformer
15        self.transformer = nn.Transformer(
16            hidden_dim, nheads, num_encoder_layers, num_decoder_layers)
17
18        # prediction heads, one extra class for predicting non-empty slots
19        # note that in baseline DETR linear_bbox layer is 3-layer MLP
20        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
21        self.linear_bbox = nn.Linear(hidden_dim, 4)
22
23        # output positional encodings (object queries)
24        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
25
26        # spatial positional encodings
27        # note that in baseline DETR we use sine positional encodings
28        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
29        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
30
31    def forward(self, inputs):
32        # propagate inputs through ResNet-50 up to avg-pool layer
33        x = self.backbone.conv1(inputs)
34        x = self.backbone.bn1(x)
35        x = self.backbone.relu(x)
36        x = self.backbone.maxpool(x)
37
38        x = self.backbone.layer1(x)
39        x = self.backbone.layer2(x)
40        x = self.backbone.layer3(x)
41        x = self.backbone.layer4(x)
42
43        # convert from 2048 to 256 feature planes for the transformer
44        h = self.conv(x)
45
46        # construct positional encodings
47        H, W = h.shape[-2:]
48        pos = torch.cat([
49            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
50            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
51        ], dim=-1).flatten(0, 1).unsqueeze(1)
52
53        # propagate through the transformer
54        h = self.transformer(pos + 0.1 * h.flatten(2).permute(2, 0, 1),
55                            self.query_pos.unsqueeze(1)).transpose(0, 1)
56
57
58        # finally project transformer outputs to class labels and bounding boxes
59        pred_logits = self.linear_class(h)
60        pred_boxes = self.linear_bbox(h).sigmoid()
61
62        return {'pred_logits': pred_logits,
63                'pred_boxes': pred_boxes}
```

▼ 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 drier',
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_resnet101', pretrained=True)
2 model.eval():
3
4 url = 'http://farm2.staticflickr.com/1061/757365702_8889181c24_z.jpg'
5 im = Image.open(requests.get(url, stream=True).raw) # put your own image
6
7 # mean-std normalize the input image (batch-size: 1)
8 img = transform(im).unsqueeze(0)
9
10 # propagate through the model
11 outputs = model(img)
12
13 # keep only predictions with 0.7+ confidence
14 probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
15 keep = probas.max(-1).values > 0.9
16
17 # convert boxes from [0: 1] to image scales
18 bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)
19
20 # mean-std normalize the input image (batch-size: 1)
21 img = transform(im).unsqueeze(0)
22
23 # propagate through the model
24 outputs = model(img)
25
26 # keep only predictions with 0.7+ confidence
27 probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
28 keep = probas.max(-1).values > 0.9
29
30 # convert boxes from [0: 1] to image scales
31 bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)
32
33 # mean-std normalize the input image (batch-size: 1)
34 img = transform(im).unsqueeze(0)
35
36 # propagate through the model
37 outputs = model(img)
38
39 # keep only predictions with 0.7+ confidence
40 probas = outputs['pred_logits'].softmax(-1)[0, :, :-1]
41 keep = probas.max(-1).values > 0.9
42
43 # convert boxes from [0: 1] to image scales
44 bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)
45
46 plot_results(im, probas[keep], bboxes_scaled)
```

```

→ Downloading: "https://github.com/facebookresearch/detr/zipball/main" to /root/.cache/torch/hub/main.zip
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or 'None' for 'weights' are deprecated
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet101-63fe2227.pth" to /root/.cache/torch/hub/checkpoints/resnet101-63fe2227.pth
100% [██████████] | 171M/171M [00:01<00:00, 103MB/s]
Downloading: "https://dl.fbaipublicfiles.com/detr/detr-r101-2c7b67e5.pth" to /root/.cache/torch/hub/checkpoints/detr-r101-2c7b67e5.pth
100% [██████████] | 232M/232M [00:01<00:00, 155MB/s]

```



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.

```

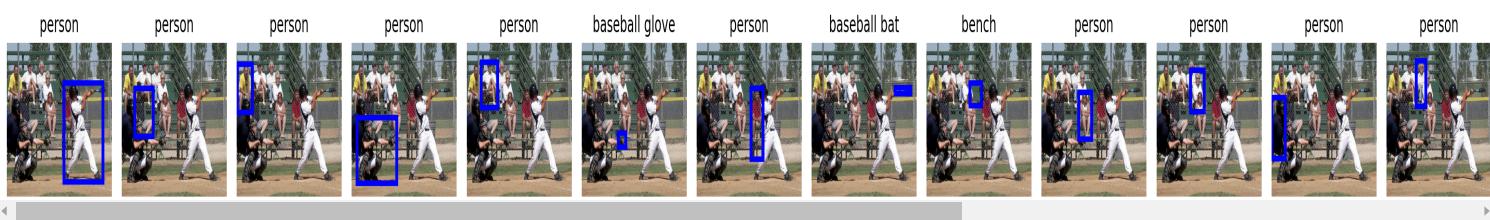
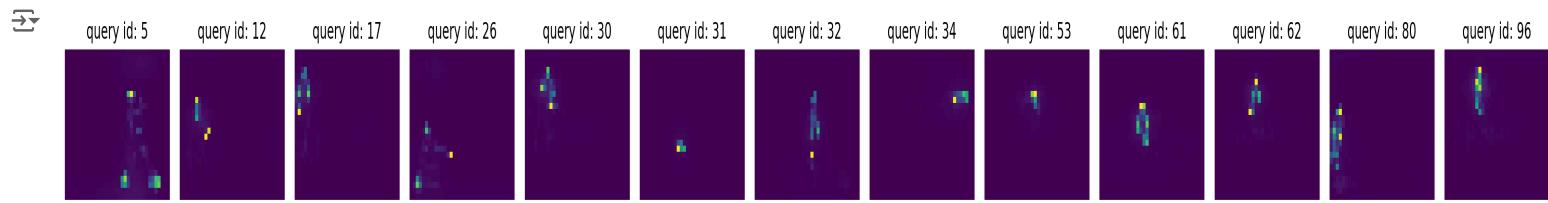
1 # use lists to store the outputs via up-values
2 conv_features, enc_attn_weights, dec_attn_weights = [], [], []
3
4 hooks = [
5     model.backbone[-2].register_forward_hook(
6         lambda self, input, output: conv_features.append(output)
7     ),
8     model.transformer.encoder.layers[-1].self_attn.register_forward_hook(
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) # put your own image
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]
26
27 # get the feature map shape
28 h, w = conv_features['0'].tensors.shape[-2:]
29
30 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(pplt.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([850, 850])
Feature map:            torch.Size([1, 2048, 25, 34])

```

```

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, 34, 25, 34])

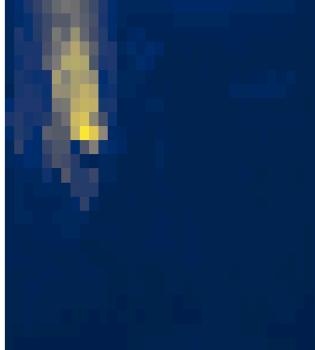
```

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

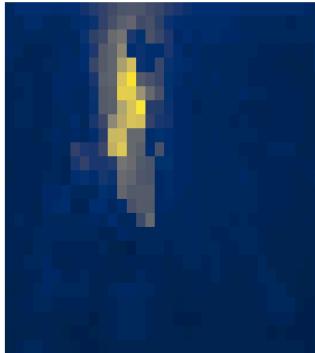
```



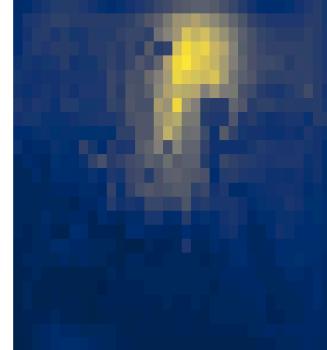
self-attention(200, 200)



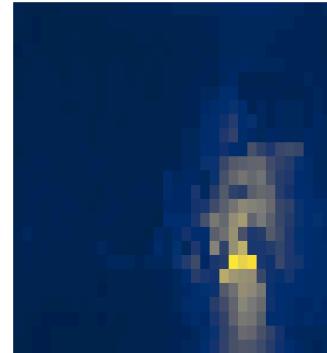
self-attention(280, 400)



self-attention(200, 600)



self-attention(440, 800)



▼ Q3. Understanding Attention Mechanisms

In this task, you focus on understanding the attention mechanisms present in the encoder and decoder of DETR.

- Briefly describe the types of attention used in the encoder and decoder, and explain the key differences between them.
- Based on the visualized results from Q2, provide an analysis of the distinct characteristics of each attention mechanism in the encoder and decoder. Feel free to express your insights.
- Encoder에서 attention은 'self-attention'으로, 이미지 feature들의 관계 파악을 강화해준다. 즉, global context를 학습한다. 반면에 Decoder에서 attention은 'self-attention'뿐만 아니라 'cross-attention'도 수행한다. 이는 인코더의 출력 정보에 해당하는 이미지 피쳐와 디코더의 쿼리를 연결하여 각 객체에 대한 구체적인 정보를 얻게 해준다. 간단히 말해, 인코더의 attention은 이미지의 글로벌한 비전을 제공하고, 디코더의 attention은 개별 객체에 대한 구체적인 정보를 예측한다.
- 이미지를 보면 인코더의 attention 매커니즘은 전체적인 이미지로부터 해당 포인트를 attention하여 전반적으로 넓은 영역에 하이라이트가 들어간 반면, 디코더의 attention 매커니즘은 쿼리로부터 attention을 수행하여 좁은 영역에 하이라이트가 들어가 있다. 이는 위에서 언급한 인코더는 글로벌한 비전을 제공하고, 디코더는 로컬한 비전을 제공한다는 내용과 상통한다.

더블클릭 또는 Enter 키를 눌러 수정