Q1. Understanding DETR model

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

```
from torch import nn
class DETR(nn.Module):
  def __init__(self, num_classes, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder
    super().__init__()

    self.backbone = resnet50()
    del self.backbone.fc #resnet의 구조에서 마지막 레이어인 fc는 필요하지 않음

    self.conv = nn.Conv2d(2048, hidden_dim, 1)

    self.transformer = nn.Transformer(hidden_dim, nheads, num_encoder_layers, num_decoder_lay

    self.linear_class = nn.Linear(hidden_dim, num_classes+1) # 분류되는 클래스에는 no object도
    self.linear_bbox = nn.Linear(hidden_dim, 4) # bbox를 결정하는 인자는 4개(cx, cy, w, h)

    self.query_pos = nn.Parameter(torch.rand(num_queries, hidden_dim)) # 디코더의 object quer

    #인코더에 입력으로 사용되는 쿼리(즉 추출된 이미지 feature)에 추가되는 positional encoding
    self.row_embed = nn.Parameter(torch.rand(50, hidden_dim//2))
    self.col_embed = nn.Parameter(torch.rand(50, hidden_dim//2))

  def forward(self, inputs):
    x = self.backbone.conv1(inputs)
    x = self.backbone.bn1(x)
    x = self.backbone.relu(x)
    x = self.backbone.maxpool(x)

    x = self.backbone.layer1(x)
    x = self.backbone.layer2(x)
    x = self.backbone.layer3(x)
    x = self.backbone.layer4(x)

    h = self.conv(x) # resnet 결과인 2048 채널의 텐서를 256 채널로 변환함

    H, W = h.shape[-2:]
    pos = torch.cat([
        self.col_embed[:W].unsqueeze(0).repeat(H,1,1),
        self.row_embed[:H].unsqueeze(1).repeat(1,W,1),
    ], dim=-1).flatten(0,1).unsqueeze(1) # [H*W, 1, hidden_dim]으로 positional encoding 크기를

    h = self.transformer(pos + 0.1*h.flatten(2).permute(2,0,1),
                         self.query_pos.unsqueeze(1)).transpose(0,1)

    pred_logits = self.linear_class(h)
    pred_boxes = self.linear_bbox(h).sigmoid()

    return {'pred_logits': pred_logits,
            '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.

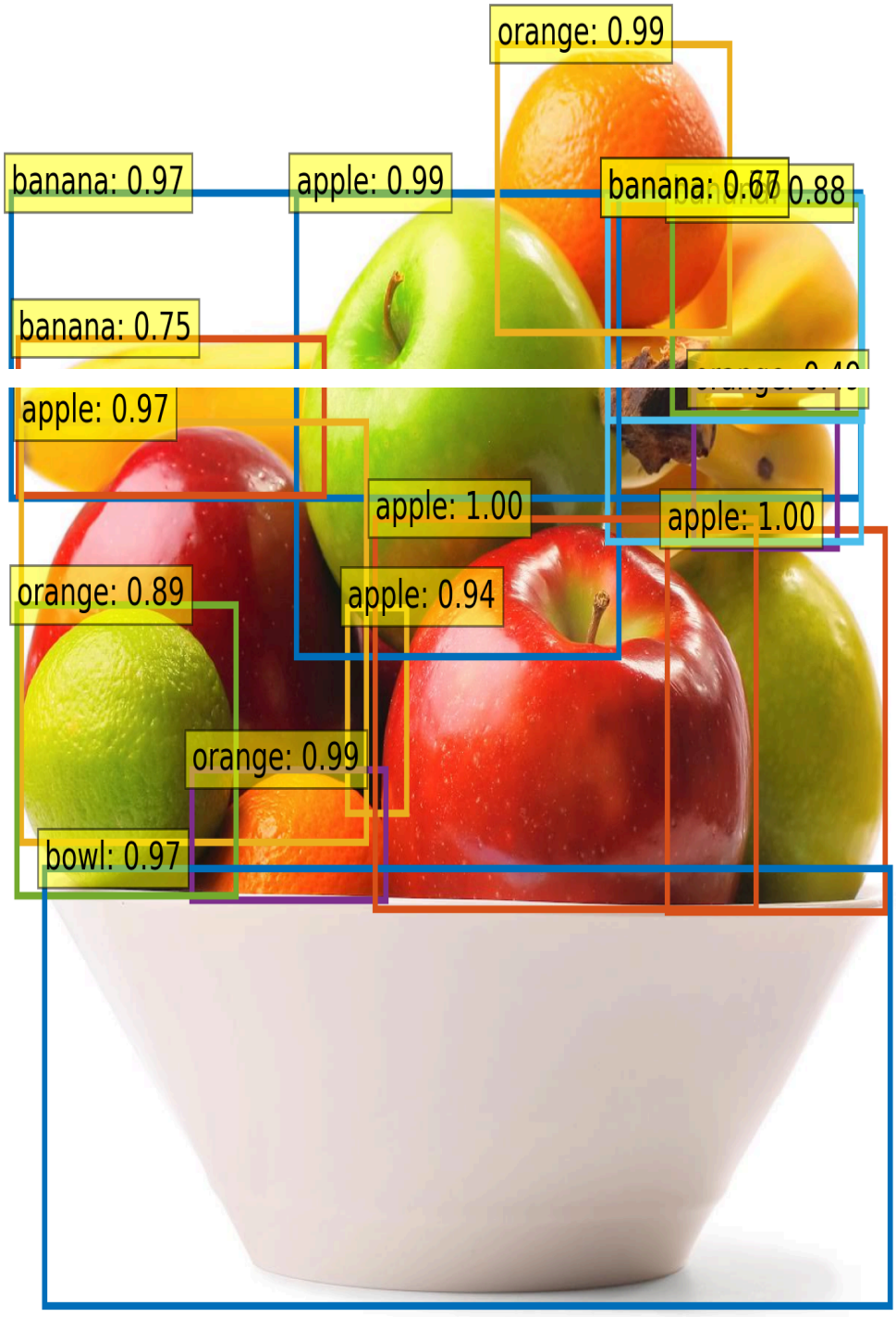
```
import math

from PIL import Image
import requests
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'

import ipywidgets as widgets
from IPython.display import display, clear_output

import torch
from torch import nn


from torchvision.models import resnet50
import torchvision.transforms as T
torch.set_grad_enabled(False);

# COCO classes
CLASSES = [
    'N/A', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A',
    'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse',
    'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack',
    'umbrella', 'N/A', 'N/A', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis',
    'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove',
    'skateboard', 'surfboard', 'tennis racket', 'bottle', 'N/A', 'wine glass',
    'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich',
    'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake',
    'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table', 'N/A',
    'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard',
    'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A',
    'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier',
    'toothbrush'
]
```

```
COLORS = [[0.000, 0.447, 0.741], [0.850, 0.325, 0.098], [0.929, 0.694, 0.125],
          [0.494, 0.184, 0.556], [0.466, 0.674, 0.188], [0.301, 0.745, 0.933]]

transform = T.Compose([
    T.Resize(800),
    T.ToTensor(),
    T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

def box_cxcywh_to_xyxy(x):
    x_c, y_c, w, h = x.unbind(1)
    b = [(x_c - 0.5 * w), (y_c - 0.5 * h),
         (x_c + 0.5 * w), (y_c + 0.5 * h)]
    return torch.stack(b, dim=1)

def rescale_bboxes(out_bbox, size):
    img_w, img_h = size
    b = box_cxcywh_to_xyxy(out_bbox)
    b = b * torch.tensor([img_w, img_h, img_w, img_h], dtype=torch.float32)
    return b

def plot_results(pil_img, prob, boxes):
    plt.figure(figsize=(16,10))
    plt.imshow(pil_img)
    ax = plt.gca()
    colors = COLORS * 100
    for p, (xmin, ymin, xmax, ymax), c in zip(prob, boxes.tolist(), colors):
        ax.add_patch(plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,
                                   fill=False, color=c, linewidth=3))
        cl = p.argmax()
        text = f'{CLASSES[cl]}: {p[cl]:0.2f}'
        ax.text(xmin, ymin, text, fontsize=15,
                bbox=dict(facecolor='yellow', alpha=0.5))
    plt.axis('off')
    plt.show()
```

```
model = torch.hub.load('facebookresearch/detr', 'detr_resnet50', pretrained=True)
model.eval() # 평가 모드로 모델을 실행

url="https://www.careclubusa.com/News/wp-content/uploads/2014/02/bowl-of-fruit_mini.jpg"
im = Image.open(requests.get(url, stream=True).raw)

img = transform(im).unsqueeze(0)

outputs = model(img)

probas = outputs['pred_logits'].softmax(-1)[0, :, :-1] #class의 마지막은 no object이므로 배제
keep = probas.max(-1).values > 0.1

bboxes_scaled = rescale_bboxes(outputs['pred_boxes'][0, keep], im.size)

img = transform(im).unsqueeze(0)
```

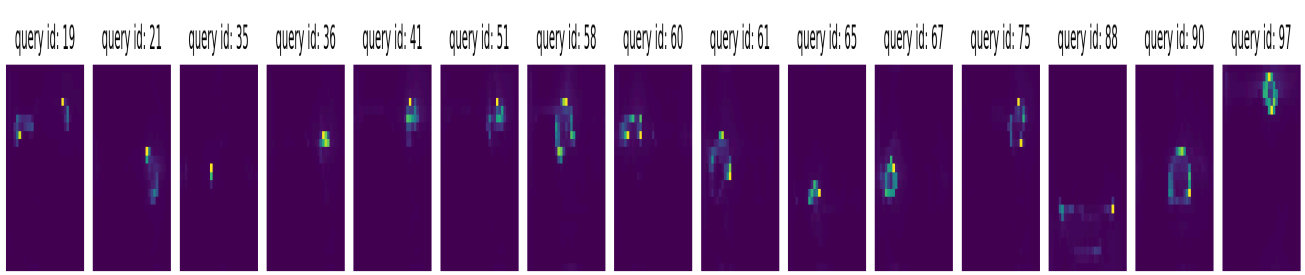
```
outputs = model(img)

plot_results(im, probas[keep], bboxes_scaled)
```

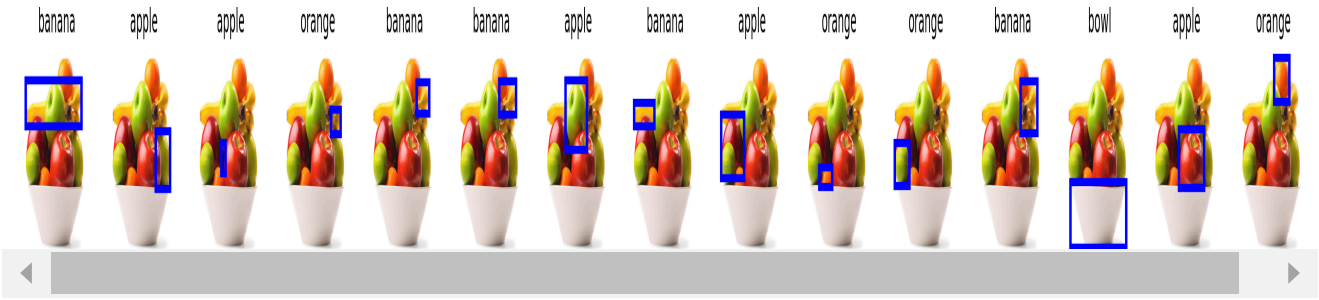
```
Using cache found in /root/.cache/torch/hub/facebookresearch_detr_main
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning:
  warnings.warn(msg)
```




```
conv_features, enc_attn_weights, dec_attn_weights = [], [], [] #hook으로 가중치 등을 캡쳐해서
```

```
hooks = [
    model.backbone[-2].register_forward_hook( #backbone[-2]는 conv 레이어이므로 conv에서 출력
        lambda self, input, output: conv_features.append(output)
    ),
    model.transformer.encoder.layers[-1].self_attn.register_forward_hook( #encoder의 어텐션
        lambda self, input, output: enc_attn_weights.append(output[1])
    ),
    model.transformer.decoder.layers[-1].multihead_attn.register_forward_hook( #decoder의 O
        lambda self, input, output: dec_attn_weights.append(output[1])
    )
]

outputs = model(img)

for hook in hooks:
  hook.remove() #hook을 제거하여 계속해서 forward_pass를 할 때마다 hook이 실행되지 않도록 한다

# 이 경우에는 forward_pass를 한 번만 수행한다. (outpus = model(img)만 수행함)
conv_features = conv_features[0]
enc_attn_weights = enc_attn_weights[0]
dec_attn_weights = dec_attn_weights[0]


h, w = conv_features['0'].tensors.shape[-2:]

fig, axs = plt.subplots(ncols=len(bboxes_scaled), nrows=2, figsize=(22, 7))
colors = COLORS * 100
for idx, ax_i, (xmin, ymin, xmax, ymax) in zip(keep.nonzero(), axs.T, bboxes_scaled):
    ax = ax_i[0]
    ax.imshow(dec_attn_weights[0, idx].view(h, w))
    ax.axis('off')
    ax.set_title(f'query id: {idx.item()}')
    ax = ax_i[1]
    ax.imshow(im)
    ax.add_patch(plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,
                               fill=False, color='blue', linewidth=3))
    ax.axis('off')
    ax.set_title(CLASSES[probas[idx].argmax()])
fig.tight_layout()
```






```
f_map = conv_features['0']
print("Encoder attention:     ", enc_attn_weights[0].shape)
print("Feature map:           ", f_map.tensors.shape)
```

```
Encoder attention:      torch.Size([775, 775])
Feature map:            torch.Size([1, 2048, 25, 31])
```

```
shape = f_map.tensors.shape[-2:]
sattn = enc_attn_weights[0].reshape(shape+shape)
print("Reshaped self-attention:", sattn.shape)
```

```
Reshaped self-attention: torch.Size([25, 31, 25, 31])
```

```
fact = 32

idxs = [(400, 500), (400, 200), (100, 600), (250, 700),]

fig = plt.figure(constrained_layout=True, figsize=(25*0.7, 8.5*0.7))
gs = fig.add_gridspec(2, 4) # 2행 4열의 그리드 공간을 만듦
axs = [ #self-attention을 시각화한 그리드 셀이 위치하는 곳
    fig.add_subplot(gs[0,0]),
    fig.add_subplot(gs[1,0]),
    fig.add_subplot(gs[0,-1]),
    fig.add_subplot(gs[1,-1]),
]

for idx_o, ax in zip(idxs, axs):
  idx = (idx_o[0] // fact, idx_o[1] // fact)
  ax.imshow(sattn[..., idx[0], idx[1]], cmap='cividis', interpolation='nearest')
  ax.axis('off')
  ax.set_title(f'self-attention{idx_o}')
```

```
fcenter_ax = fig.add_subplot(gs[:, 1:-1]) # 중앙에 나타나는 원본 이미지 영역
fcenter_ax.imshow(im)
for (y,x) in idxs:
  scale = im.height / img.shape[-2]
  x = ((x // fact) + 0.5) * fact
  y = ((y // fact) + 0.5) * fact
  fcenter_ax.add_patch(plt.Circle((x*scale, y*scale), fact//2, color='r'))
  fcenter_ax.axis('off')
```

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.

인코더와 디코더는 구성하는 attention 종류가 다르다.

인코더 : self-attention만 포함 디코더 : cross-attention, self-attention 둘 다 포함

self attention : query, key, value가 모두 같은 데이터로부터 derived cross attention : key, value가 derived된 데이터와 query가 derived된 데이터가 다름.

인코더는 입력 데이터의 특징을 추출하는 것이 주된 역할이고, 디코더는 인코더가 추출한 특징을 수행하고자 하는 작업에 활용하는 역할. 이 때 query가 수행하고자 하는 작업과 관련된 정보를 담고 있고 key, value는 인코더가 추출한 정보를 담고 있다. 디코더는 self-attention을 포함하는데 이는 디코더의 입력 시퀀스 간의 관계를 학습하는 데에 사용되고 이 결과가 다음에 연결되는 cross-attention의 query로 들어가게 된다. DETR의 경우로 설명을 하자면 encoder가 이미지의 정보를 추출하고 decoder의 입력이 self-attention을 통해 검출된 object query간의 관계가 학습되고, 학습된 object query가 cross-attention의 query로 들어가고, encoder의 output이 key, value로 들어가게 되어 학습된 object query에 대한 image feature를 담고 있는 답이 만들어지게 되는 것이다.

위에 2번째로 나오는 출력 이미지는 decoder layers[-1]의 attention weights를 시각화하고 있으므로 cross-attention을 시각화하고 있고 가장 아래에 있는 출력 결과 이미지는 encoder의 attention이므로 self-attention 결과를 시각화한 것이다.

두 이미지 출력 결과를 비교해볼 때, (400, 200), (100,600) idx에서 관찰한 self-attention weight를 봤을 때도 유사하게 객체의 형태를 비교적 잘 추출해냈음을 확인했다. 다만 (400, 500) idx에서 관찰한 self-attention weight를 봤을 때 사과의 동그란 형체와 그 아래에 있는 bowl의 형체까지 비교적 선명하게 관찰되는 것을 알 수 있으나 가까이 위치한 bowl의 영역까지 비슷한 가중치를 공유하고 있다는 사실을 통해 항상 객체를 완벽히 분리해낼 수 있는 것은 아니라는 점을 알 수 있다. 인접한 물체와 유사한 weight를 가지게 되는 이러한 현상은 spatial embedding으로 인한 효과라고 짐작할 수 있다. 반면 (250,700)에서 관찰한 weight를 봤을 때 바나나는 이미지에서 occulusion이 일어났기 때문에 객체의 형태를 잘 파악하지 못했음을 확인했다. 이는 인접한 서로 다른 객체끼리는 동일한 객체로 인식할 수 있지만, 같은 객체더라도 occulusion으로 인해 객체의 일부분이 거리를 두고 위치하게 되면 encoder가 같은 객체라고 판단하기 힘들다는 점을 입증한다. 따라서 encoder에서는 spatial embedding의 정보가 상당히 높은 비중으로 가중치를 결정하고 있음을 유추할 수 있다.

반면 decoder의 cross-attention을 봤을 때는 앞서 occulusion으로 인해 encoder에서 하나의 객체로 잘 인식하지 못했던 바나나의 전체 형태가 인식되었으며 이는 encoder가 얻어낸 이미지 속 객체들의 정보가 decoder의 cross attention으로 인해 교환되었기 때문으로 생각된다. 이외에도 딱히 인식하는 데에 어려움이 없어 보이는 사물들은 거의 완벽히 객체의 형태를 파악하고 있음을 어텐션 가중치 영역에서 확인할 수 있었다