Object Detection with DETR

Group 14:
Tuan Nguyen
Nikhil Darwin Bollepalli
Erum Hooda
DETR Architecture

- DETR: Object DEtection with TRansformers
- Goal:
  - Evaluate our own implementation of DETR against original DETR implementation by Facebook
Transformer Encoder-Decoder

- Encoder
  - Input = feature map + positional encodings
  - Has multi-head self-attention module and feed forward network
  - Encodes image features
- Decoder
  - Input = encoder output + queries
  - N queries learned in training
  - Each query results in bounding box + class label
  - Some queries map to no object
  - Output fed into feed forward network
Loss Function

- Bipartite Matching
  - Assign each predicted bounding box + class label to a ground truth bounding box + class label
  - 1:1 matching
- Hungarian Algorithm
  - Finds optimal bipartite matching
  - Minimize total loss
Backbone: MobileNetV2

- **Backbone:**
  - pretrained CNN
  - Outputs feature representation of input image
- **MobileNetV2 chosen over ResNet50 because it is smaller,** which means:
  - shorter running time
  - Less memory used
- **Outputted feature map fed into Encoder-Decoder**

![Diagram of MobileNetV2 architecture](image)

(d) MobileNet V2
Loss vs Accuracy Curve: MobileNetV2

- Training time: 8+ hours
- Top-1 accuracy: 75%
- Epochs trained: 200
- Batch size: 128
- Dataset: ImageNet Subset
Dataset

COCO dataset

- 91 classes, including “N/A”
- 328K images
Training

- Training DETR is extremely resource intensive even using a smaller backbone
  - 41M parameters VS 16M parameters
- Impractical to train on consumer hardware
  - Original paper trained on 16 V100 GPUs
  - We had 1 P100 (Kaggle)
  - 1 epoch of full transformer took 10 hours on Kaggle
  - 1 epoch of the scaled down transformer took 4 hours

<table>
<thead>
<tr>
<th>Hyperparameters</th>
<th>Queries</th>
<th>Hidden</th>
<th>Heads</th>
<th>Encoder</th>
<th>Decoder</th>
<th>Feedforward</th>
<th>Learning Rate</th>
<th>Batch Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>100</td>
<td>512</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>2048</td>
<td>1.00E-04</td>
<td>64</td>
</tr>
<tr>
<td>1st Attempt</td>
<td>100</td>
<td>512</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>2048</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>2nd Attempt</td>
<td>50</td>
<td>512</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1024</td>
<td>0.1</td>
<td>1</td>
</tr>
</tbody>
</table>
Results

- Ability to train was limited
  - Kaggle kept timing out
  - 1 epoch on full transformer
  - 5 epochs on scaled down transformer
- Transformer proved to be a much bigger bottleneck compared to the backbone
Questions?
References


