1 Details on Faster R-CNN

Our model is implemented in TensorFlow (Abadi et al., 2015). We follow Anderson et al. (2018) in terms of model architecture, data splits, and processing steps. We describe major components and differences below. In particular, we use the latest version of Visual Genome (v1.4), with 1600 object and 400 attribute categories. We also have the “background” class for objects and the “no attribute” class for attributes. We limit the number of attributes per object to 16. We resize the image to so that the maximum of height or width is 896. We train our model with a batch size of 64 for 50K steps, using SGD with momentum on an 8-core Google Cloud TPU\(^1\). We clip the gradient if the norm is greater than 10. We use the cosine learning rate schedule with 1K warm-up steps, increasing the learning rate from 0.003 to 0.04 and reducing it to 0.01 at step 20K and to 0.005 at step 40K. We apply random crops to images and use batch normalization (Ioffe and Szegedy, 2015) as well as DropBlock (Ghiasi et al., 2018) on block 3 and block 4 of the ResNet-101 during training. Our features come from fc6 after ReLU.

2 Details on Image Captioning

Our model is implemented in TensorFlow (Abadi et al., 2015). Our Transformer-based architecture has a stack of 6 layers for both the encoder and the decoder. The number of attention heads is set to 8. We do not use positional encoding. We have an additional dense projection layer for each type of input features (see Figure 1 for examples). Moreover, for Faster-RCNN features, we observe the best performance when first transforming the 2048D input feature vector to a 64D one (as in Ultra) using another projection layer, and thus report accuracy numbers in this setting.

At the same time, we also have these projection layers in our VQA architecture when using Ultra features (see the next section). We use Adam optimizer (Kingma and Ba, 2015) with a warm-up style learning rate schedule, linearly increasing the learning rate in the first 20 epochs until it reaches 0.000032 and then use a decay rate of 0.95 for every 25 epochs. We tuned the initial learning rate over \{0.000016, 0.000032, 0.000064\}. We train our model with a batch size of 4096 on a 32-core Google Cloud TPU for a total of 2 million steps. Each training run takes approximately 4 days.

In Figure 1, we show how we convert an image (pixels) to an input sequence of image features to the Transformer-based model described in the main text.

3 Details on VQA

Our model is implemented in TensorFlow (Abadi et al., 2015). As mentioned in the main text, the architecture is a simplified “up-down” model of (Anderson et al., 2018). This architecture has two major differences. First, it uses weight normalization (Salimans and Kingma, 2016) followed by ReLU instead of the more expensive gated hyperbolic tangent activation. Second, it uses multimodal combination by element-wise multiplication instead of by feature concatenation.

The only minor differences from Pythia v0.1 are that we use Adam (Kingma and Ba, 2015), not its variant AdaMax, and that we use a single classifier layer instead of two vision and language layers in all of our experiments. We use Pythia v0.1 (Jiang et al., 2018) to preprocess VizWiz dataset and retain 3135 top answers. Analogous to what we observe in our image captioning model, for Ultra features, we see the best performance when scaling and expanding the 64D input feature vector to a 2048D one (as in FRCNN) using another projec-
Figure 1: Pipeline for converting an image to a sequence of image features in our highest performing image captioning model on the Conceptual Captions benchmark, used as input to the Transformer-based model.

<table>
<thead>
<tr>
<th></th>
<th>val</th>
<th>test-dev</th>
<th>test-standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>VizWiz \cite{Gurari18}</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BAN \cite{Kim18}</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ours (FRCCN)</td>
<td>55.2</td>
<td>53.6</td>
<td>72.7</td>
</tr>
<tr>
<td>Ours (Ultra)</td>
<td>56.8</td>
<td>55.1</td>
<td>71.7</td>
</tr>
</tbody>
</table>

Table 1: Accuracy (%) for the VQA task on the VizWiz dataset. Additionally, we provide accuracy per answer type on the test-dev and test-standard splits: yes/no (y/n), number (num), unanswerable (unans), and the rest (other).

4 Full results on the VizWiz benchmark

Table 1 reports accuracy on additional splits of VizWiz, complementing the one in the main text.

References


