Transparency by Design: Closing the Gap Between Performance and Interpretability in Visual Reasoning

Supplementary Material

1. Module Details

Here we describe each module in detail and provide motivation for specific architectural choices. In all descriptions, 'image features' refer to features that have been extracted using a pretrained model [1] and passed through our stem network, which is shared across modules. In all the following tables, $\delta(\cdot)$ will indicate a rectified linear function and $\sigma(\cdot)$ will indicate a sigmoid activation. The input size $R \times C$ indicates R rows and C columns in the input. In our original model, $R = C = 14$, while our high-resolution model uses $R = C = 28$.

The architecture of the **Attention** modules can be seen in Table 1. These modules take stem features and an attention mask as input and produce an attention mask as output. We first perform an elementwise multiplication of the input features and attention mask, broadcasting the attention mask along the channel dimension of the input features. We refer to this process as 'attending to' the features. We then process the attended features with two 3x3 convolutions, each with 128 filters and a ReLU, then use a single 1x1 convolution followed by a sigmoid to project down to an attention mask. This architecture is motivated by the design of the unary module from Johnson *et al.* [3].

The **And** and **Or** modules, seen in Table 2 and Table 3, respectively, perform set intersection and union operations. These modules return the elementwise minimum and maximum, respectively, of two input attention masks. This is motivated by the logical operations that Hu *et al.* [2] implement, which seems a natural expression of these operations. Such simple and straightforward operations need not be learned, since they can be effectively and efficiently implemented by hand.

Table 1. Architecture of an `Attention` module, which takes as input Features and an Attention and produces an Attention.

Index	Layer	Output Size
(1)	Features	$128 \times R \times C$
(2)	Previous module output	$1 \times R \times C$
(3)	Elementwise multiply (1) and (2)	$128 \times R \times C$
(4)	$\delta(\text{Conv}(3 \times 3, 128 \rightarrow 128))$	$128 \times R \times C$
(5)	$\delta(\text{Conv}(3 \times 3, 128 \rightarrow 128))$	$128 \times R \times C$
(6)	$\sigma(\text{Conv}(1 \times 1, 128 \rightarrow 1))$	$1 \times R \times C$

Table 2. Architecture of an `And` module. This module receives as input two Attentions and produces an Attention.

Index	Layer	Output Size
(1)	Previous module output	$1 \times R \times C$
(2)	Previous module output	$1 \times R \times C$
(3)	Elementwise minimum (1) and (2)	$1 \times R \times C$

Table 3. Architecture of an `Or` module. This module receives as input two Attentions and produces an Attention.

Index	Layer	Output Size
(1)	Previous module output	$1 \times R \times C$
(2)	Previous module output	$1 \times R \times C$
(3)	Elementwise maximum (1) and (2)	$1 \times R \times C$

The **Relate** module, shown in Table 4, needs global context to shift attention across an entire image. Motivated by this, we use a series of dilated 3x3 convolutions, with dilation factors 1, 2, 4, 8, and 1, to expand the receptive field to the entire image. The choice of dilation factors is informed by the work of Yu and Koltun [4]. These modules receive stem features and an attention mask and produce an attention mask. Each convolution in the series has 128 filters and is followed by a ReLU. A final convolution then reduces the feature map to a single-channel attention mask, and a sigmoid nonlinearity is applied.

The **Same** module is the most complex of our modules, and the most complex operation we perform. To illustrate this, consider the `Same[shape]` module. It must determine the shape of the attended object, compare that shape with the shape of every other object in the scene (which requires global information propagation), and attend to all the objects that share that shape. Initially, we used a design similar to the `Relate` module to perform this operation, but found it did not perform well. After further reflection, we posited this was because the `Relate` module does not have a mechanism for remembering which object we are

Table 4. Architecture of a `Relate` module. These modules receive as input Features and an Attention and produce an Attention.

Index	Layer	Output Size
(1)	Features	$128 \times R \times C$
(2)	Previous module output	$1 \times R \times C$
(3)	Elementwise multiply (1) and (2)	$128 \times R \times C$
(4)	δ(Conv($3 \times 3, 128 \to 128$, dilate 1))	$128 \times R \times C$
(5)	δ(Conv($3 \times 3, 128 \to 128$, dilate 2))	$128 \times R \times C$
(6)	δ(Conv($3 \times 3, 128 \to 128$, dilate 4))	$128 \times R \times C$
(7)	δ(Conv($3 \times 3, 128 \to 128$, dilate 8))	$128 \times R \times C$
(8)	δ(Conv($3 \times 3, 128 \to 128$, dilate 1))	$128 \times R \times C$
(9)	σ(Conv($1 \times 1, 128 \to 1$))	$1 \times R \times C$

interested in performing the `Same` with respect to. Table 5 provides an overview of the `Same` module. Here we explicate the notation. Provided with stem features and an attention mask as input, we take the $\arg\max$ of the feature map, spatially. This gives us the (x, y) position of the object of interest (*i.e.* the object to perform the `Same` with respect to). We then extract the feature vector at this spatial location in the input feature map, which gives us the vector encoding the property of interest (among other properties). Next, we perform an elementwise multiplication with the feature vector at every spatial dimension. This essentially performs a cross-correlation of the feature vector of interest with the feature vector of every other position in the image. Our intuition is that the vector dimensions that encode the property of interest will be 'hot' at every point sharing that property with the object of interest. At this point, a convolution could be learned that attends to the relevant regions. However, the `Same` operation, by its definition in CLEVR, must *not* attend to the original object. That is, an object is by definition not the same property as itself. Therefore, the `Same` module must learn not to attend to the original object. We thus concatenate the original input attention mask with the cross-correlated feature map, allowing the convolutional filter to know which object was the original, and thus ignore it.

Table 5. Architecture of a `Same` module. These modules receive as input Features and an Attention and produce an Attention.

Index	Layer	Output Size
(1)	Features	$128 \times R \times C$
(2)	Previous module output	$1 \times R \times C$
(3)	$\arg\max_{x,y}(2)$	$1 \times 1 \times 1$
(4)	$(1)_{(3)}$	$128 \times 1 \times 1$
(5)	Elementwise multiply (1) and (4)	$128 \times R \times C$
(6)	Concatenate (5) and (2)	$129 \times R \times C$
(7)	σ(Conv($1 \times 1, 129 \to 1$))	$1 \times R \times C$

The **`Query`** module architecture can be seen in Table 6.

Table 6. Architecture of a `Query` module. These modules receive as input Features and an Attention and produce an Encoding.

Index	Layer	Output Size
(1)	Features	$128 \times R \times C$
(2)	Previous module output	$1 \times R \times C$
(3)	Elementwise multiply (1) and (2)	$128 \times R \times C$
(4)	δ(Conv($3 \times 3, 128 \to 128$))	$128 \times R \times C$
(5)	δ(Conv($3 \times 3, 128 \to 128$))	$128 \times R \times C$

Table 7. Architecture of a `Compare` module. These modules receive as input two Features and produce an Encoding.

Index	Layer	Output Size
(1)	Previous module output	$128 \times R \times C$
(2)	Previous module output	$128 \times R \times C$
(3)	Concatenate (1) and (2)	$128 \times R \times C$
(4)	δ(Conv($1 \times 1, 128 \to 128$))	$128 \times R \times C$
(5)	δ(Conv($3 \times 3, 128 \to 128$))	$128 \times R \times C$
(6)	δ(Conv($3 \times 3, 128 \to 128$))	$128 \times R \times C$

Its design is similar to that of the `Attention` modules and is likewise inspired by the unary module design of Johnson *et al.* [3] and adapted for receiving an attention mask and stem features as input. These modules produce a feature map as output, and thus do not have a convolutional filter that performs a down-projection.

The **`Compare`** module, shown in Table 7, is inspired by the binary module of Johnson *et al.* [3]. These modules take two feature maps as input and produce a feature map as output. Their purpose is to determine whether the two input feature maps encode the same property.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. Learning to reason: End-to-end module networks for visual question answering. *CoRR*, abs/1704.05526, 2017.

[3] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. Inferring and executing programs for visual reasoning. *arXiv preprint arXiv:1705.03633*, 2017.

[4] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.