Supplementary Material

A Prior on z_{pres}

Here we provide a derivation for the prior on z_{pres} (a vector of length HW made up of all z^{ij}_{pres}), discussed briefly in Section 4.2. Let C be a random variable giving the number of non-zero entries in z_{pres}. Under the prior, z_{pres} is generated by first sampling C, and then drawing z_{pres} uniformly from all binary vectors that have C non-zero entries.

$$p(z_{\text{pres}}) = p(C = nz(z_{\text{pres}}))p(z_{\text{pres}}|C = nz(z_{\text{pres}}))$$

where $nz(\cdot)$ gives the number of non-zero entries in a vector. There are $\binom{HW}{C}$ binary vectors of length HW with C non-zero entries, so:

$$p(z_{\text{pres}}) = p(C = nz(z_{\text{pres}}))\binom{HW}{nz(z_{\text{pres}})}^{-1}$$

For the distribution over C we use a Geometric distribution with parameter s. We use the Geometric interpretation that puts C as the number of failures before a success, and s as the success probability. By setting s to a high value, we put most of the probability mass at low values, thereby putting pressure on the network to explain images using as few objects as possible.

$$p(C = nz(z_{\text{pres}})) = s(1-s)^{nz(z_{\text{pres}})}$$

We then truncate and normalize to ensure C has support $\{0, 1, \ldots, HW\}$:

$$\begin{aligned} p(C = nz(z_{\text{pres}})) &= \frac{s(1-s)^{nz(z_{\text{pres}})}}{\sum_{c=0}^{HW} s(1-s)^c} \\ &= \frac{s(1-s)^{nz(z_{\text{pres}})}}{s\frac{(1-(1-s)^{HW+1})}{1-(1-s)}} \\ &= \frac{s(1-s)^{nz(z_{\text{pres}})}}{1-(1-s)^{HW+1}} \end{aligned}$$

Overall, the prior is:

$$p(z_{\text{pres}}) = \frac{s(1-s)^{nz(z_{\text{pres}})}}{(1-(1-s)^{HW+1})\binom{HW}{nz(z_{\text{pres}})}}$$

B KL Divergence for z_{pres}

Having derived the prior, we now turn to efficient computation of the KL divergence between the distribution over z_{pres} yielded by the network, namely $q_\phi(z_{\text{pres}}|x)$ (embodied in the β^{ij}_{pres} variables from Section 4.3) and $p(z_{\text{pres}})$. Efficient computation of this KL divergence is necessary for computing the second term of the VAE training objective (Equation 3). For convenience, we switch to indexing the entries of z_{pres} using a single index k rather than the grid indices ij. We will use the notation $z^{m:n}_{\text{pres}}$ to mean the sub-vector ranging from index m to n, inclusive.

First, from basic properties of KL divergence we have:

$$D_{KL}(q(z_{\text{pres}}|x) \parallel p(z_{\text{pres}})) = \sum_{k=1}^{n} E\left[D_{KL}(q(z^k_{\text{pres}}|z^{1:k-1}_{\text{pres}}, x) \parallel p(z^k_{\text{pres}}|z^{1:k-1}_{\text{pres}}))\right]$$

where the expectation for index k is taken over variables $z^{1:k-1}_{\text{pres}}$ sampled from the marginal $q(z^{1:k-1}_{\text{pres}}|x)$. We estimate each expectation using a single sample:

$$\approx \sum_{k=1}^{n} D_{KL}(q(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}, x) \parallel p(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}))$$

where $\hat{z}$ indicates values that have been sampled. Note that $q(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}, x)$ is just Bernoulli(β^k_{pres}) from Section 4.3.

We now turn to computation of $p(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}})$. Recalling that C is a random variable giving the number of non-zero entries in z_{pres}, we have:

$$p(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}) = \sum_{c=0}^{HW} p(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}, C)p(C|\hat{z}^{1:k-1}_{\text{pres}}) \quad \text{(B1)}$$

We focus first on the factor $p(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}, C)$. Given that we have sampled $\hat{z}^{1:k-1}_{\text{pres}}$, we still need $C - nz(\hat{z}^{1:k-1}_{\text{pres}})$ non-zeros, and we have $HW - (k-1)$ "slots" to get them with. Recalling that given C, we are assuming uniform sampling of all binary vectors with that number of non-zeros, we have:

$$p(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}}, C) = \frac{C - nz(\hat{z}^{1:k-1}_{\text{pres}})}{HW - (k-1)}$$

The final piece that we need is $p(C|\hat{z}^{1:k-1}_{\text{pres}})$. This can be decomposed recursively as:

$$p(C|\hat{z}^{1:k-1}_{\text{pres}}) \propto p(\hat{z}^{k-1}_{\text{pres}}|C, \hat{z}^{1:k-2}_{\text{pres}})p(C|\hat{z}^{1:k-2}_{\text{pres}}) \quad \text{(B2)}$$

Starting from $k = 1$, we first evaluate Equation (B1), then sample z^k from $q_\phi(z^k_{\text{pres}}|\hat{z}^{1:k-1}_{\text{pres}})$, and finally update the conditional count distribution $p(C|\hat{z}^{1:k}_{\text{pres}})$ via Equation (B2).

C Object Rendering in the Decoder Network

Here we describe in detail the differentiable rendering algorithm implemented in the decoder network. For each object we have the following values:

1. Object RGB map o^{ij} with shape $(H_{obj}, W_{obj}, 3)$
2. Object transparency map α^{ij} with shape $(H_{obj}, W_{obj}, 1)$
3. Object bounding box b^{ij} derived from z^{ij}_{where} (see Section 4.1 and Figure 1).
4. Relative depth value z^{ij}_{depth}
5. Presence value z^{ij}_{pres}

We also have a background image x_{bg} with shape $(H_{img}, W_{img}, 3)$, which may have been predicted from the input image by a neural network, or may be a fixed image calculated from the dataset in some other way.

We first perform two initial computations for each object:

$$\tilde{\alpha}^{ij} = \alpha^{ij} z^{ij}_{\text{pres}}$$
$$\gamma^{ij} = \alpha^{ij} z^{ij}_{\text{pres}} \sigma(-z^{ij}_{\text{depth}})$$

where σ is the sigmoid function. $\tilde{\alpha}^{ij}$ is a transparency map that takes into account whether the object exists, γ^{ij} is an *importance* map which is used to implement a differentiable approximation of relative depth.

For each pixel in the output image, we iterate over all objects, checking whether each object affects the current pixel (i.e. whether the pixel is inside the bounding box for the object). For all objects that affect the pixel, three values are extracted from the object: RGB values o', a transparency value α' and an importance value γ'. These values are extracted by finding the position of the pixel with respect to the object's coordinate frame, and then using bilinear interpolation to extract a value from the relevant map. The RGB value o' is then mixed with the background using α'. The mixed values from all affecting objects are then mixed with one another using a convex combination consisting of the γ' values normalized to sum to 1 (within the pixel). This mixing step implements the differentiable approximation of relative depth; objects with higher importance values (lower depth) get a larger "share" of the pixels that they affect, and appear on top of objects with lower importance (higher depth). Pseudo-code for this algorithm is given in Algorithm 1.

Algorithm 1 Differentiable rendering algorithm

function CONTAINS(b, (y, x))
Return True iff pixel (y, x) is inside bounding box b
end function

function INTERPOLATE(b, v, (y, x))
Assuming pixel (y, x) is inside bounding box b, extract a value from map v for pixel (y, x) using bilinear interpolation.
end function

function DIFFERENTIABLERENDERING
    $x_{out} \leftarrow$ COPY(x_{bg})
    for pixel with position (y, x) **do**
        n-writes $\leftarrow 0$
        weighted-sum $\leftarrow 0$
        importance-sum $\leftarrow 0$
        for object with index (i, j) **do**
            if CONTAINS($b^{ij}, (y, x)$) **then**
                $\alpha' \leftarrow$ INTERPOLATE($b^{ij}, \tilde{\alpha}^{ij}, (y, x)$)
                $\gamma' \leftarrow$ INTERPOLATE($b^{ij}, \gamma^{ij}, (y, x)$)
                $o' \leftarrow$ INTERPOLATE($b^{ij}, o^{ij}, (y, x)$)
                $o' \leftarrow \alpha' o' + (1 - \alpha') x_{bg}[y, x]$
                n-writes $\leftarrow$ n-writes+1
                weighted-sum $\leftarrow$ weighted-sum + $\gamma' o'$
                importance-sum $\leftarrow$ importance-sum + γ'
            end if
        end for
        if n-writes > 0 **then**
            $x_{out}[y, x] \leftarrow \frac{\text{weighted-sum}}{\text{importance-sum}}$
        end if
    end for
    return x_{out}
end function