

# Hierarchical Compact Clustering Attention (COCA) for Unsupervised Object-Centric Learning

## Supplementary Material

We provide a complexity analysis of our method in Section 6. Subsequently, we present architectural details in Section 7, organized similarly to Section 3 of the main paper. Implementation details for COCA-Net and the baselines are provided in Section 8. Additional results, including initial experiments on real-world datasets 9.1, additional quantitative 9.2 and qualitative 9.4 results are presented throughout Section 9. Extensions of the results from our ablation studies are also included in Section 9.3.

### 6. Complexity Analysis

Suppose that COCA-Net is to process a feature image consisting of  $N \times N$  pixels. At each layer, this feature image is partitioned into  $U \times U$  non-overlapping windows. The complexity burden of a COCA layer resides on the generation of affinity masks and their compactness measurement. In the worst case, COCA generates an affinity mask per node, which introduces  $O(U^2 \cdot U^2)$  operations for the overall affinity masks generation process. In the first layer, we have a total of  $\frac{N}{U} \cdot \frac{N}{U} = \frac{N^2}{U^2}$  windows, hence the complexity is  $O(U^2 \cdot N^2)$  for affinity masks generation. Our compactness measurement involves finding the node pair with minimum density in each affinity mask (see Eq. 3). The complexity associated with this operation increases above affinity mask generation, resulting in a complexity of  $O(U^2 \cdot U^3)$ . Therefore, the first layer of COCA has a complexity of  $O(U^3 \cdot N^2)$ . At the second layer, we repeat the same partitioning strategy to generate affinity masks and measure their compactness. Hence, we again need  $O(U^2 \cdot U^3)$  operations within a window and we have  $\frac{N}{U^2} \cdot \frac{N}{U^2} = \frac{N^2}{U^4}$  of these windows at the second layer. Therefore, the overall complexity for this layer is  $O(U \cdot N^2)$ . Similarly at the third layer, we have a complexity of  $O(\frac{N^2}{U})$  and so on.

In total, we have  $\log_U N$  layers, where  $N$  is a power of  $U$ . Assuming that  $U$  is kept constant and small, e.g.,  $U = 4$ , throughout the hierarchy, we have  $O(N^2)$  complexity for each layer. Altogether, these  $\log_U N$  layers add up to  $O(N^2 \cdot \log N)$  complexity. Comparing to our baselines, the complexity of SA (slot attention) model [38] can be considered as  $O(N^2)$ . SA conducts its query normalized cross attention operation on the full resolution feature image, which has a complexity of  $O(N^2)$ . On the contrary, instead of operating on this large  $N \times N$  image directly, COCA-Net operates on  $U \times U$  windows and handles  $\frac{N^2}{U^2}$  of windows in parallel. This structure makes COCA-Net inherently suitable for parallel implementation, as the operations within

each window are independent. It should be noted that the number of elements that COCA-Net processes decreases by a factor of  $U^2$  at each layer, hence it has the potential to be scaled up to deeper hierarchies and address object discovery in higher resolution images.

### 7. Architectural Details

#### 7.1. Pixel Feature Encoder

We use a simple backbone to initialize pixel features that encode both appearance and positional information. Specifically, the input image is first processed through a single convolutional layer that preserves resolution to generate appearance-based features. Positional information for each pixel is then incorporated through a positional encoding scheme borrowed from [38], which normalizes the initial pixel coordinates in four cardinal directions and learns a projection that maps these 4-dimensional position vectors to the output feature dimensions of the convolutional layer. After summing the positional embedding with the convolutional features of the pixels, we apply a GroupNorm normalization across features and finally use a Multi-Layer Perceptron (MLP) with a single hidden layer to fuse the position and appearance information. The resulting tensor forms the initial image features  $\mathbf{X}^0 \in \mathbb{R}^{h_0 \times w_0 \times d_0}$ , that we feed into our COCA-Net hierarchy at layer  $l = 1$ .

#### 7.2. COCA Layer (Section 3.1)

COCA layer can be viewed as a clustering variant of ViT-22B. It begins with a stack of original ViT-22B layers to refine the features of input nodes and then continues as a clustering variant of ViT-22B. While the overall structure of COCA resembles a ViT-22B, it differs significantly.

Rather than relating two distinct projections—query and key projections to compute attention weights as in the original self-attention formulation—we project the representations of input nodes onto the same high-dimensional space to compute affinities, e.g., using only query projection. In addition, we use these affinity masks to identify clusters among input nodes and apply feature pooling based on the masks of these clusters, unlike ViT-22B, which maintains the same number of nodes from input to output across its layers.

Apart from these fundamental differences, most of the remaining architecture resembles a ViT-22B, as also illustrated in Figure 2b, we also learn a parallel FFN, a values projection, an output projection and employ pre-norm skip

connection from the features that are just partitioned into non-overlapping windows.

### 7.2.1. Feature Refiner ViT-22B (Section 3.1.1)

A single ViT-22B layer [13] in COCA adopts Multi-Head Self-Attention (MHSA) [46] in parallel with the Feed Forward Network (FFN) in a Pre-Norm skip connection configuration. For the details, reader can refer to the original paper [13]. In COCA, the only modification to the original recipe is that we leverage GroupNorm operations over LayerNorm, where the former is often preferred over the latter in computer vision architectures.

The number of ViT-22B layers in the feature refinement stack of a single COCA layer may vary depending on the COCA layer’s position in the hierarchy. Specifically, we adopt an increasing number of ViT-22B layers as we move up through the hierarchy. For instance, the first COCA layer contains a single ViT-22B layer in its feature refinement stack, whereas the second COCA layer includes two ViT-22B layers, and so on.

The spatial extent of the ViT-22B layers’ operations also evolves throughout the hierarchy. In the earlier layers, the attention context is constrained to smaller windows (e.g., a kernel size of  $3 \times 3$ ), enabling feature refinement to focus on localized neighborhoods. Conversely, in the final layers, the attention context expands to a global scale, facilitating the integration of broader contextual information. Throughout this process, the stack of ViT-22B layers preserves the cardinality of the input node set; that is, a query is generated for every node, and an appropriate attention context is constructed within the spatial neighborhood for each query.

### 7.2.2. Partitioning Hierarchy (Section 3.1.2)

A visual illustration of the non-overlapping windows partitioning strategy is shared in Figure 4. After feature refinement, at each layer  $l$ , COCA contains an input node set with  $h'_{l-1} \times w'_{l-1} \times k_{l-1}$  elements. COCA partitions this input node set into non-overlapping windows, where each window has  $h_l \times w_l \times k_{l-1}$  nodes and there are a total of  $t_l^2$  of such windows. In other words, input nodes are divided into  $t_l$  groups along both spatial axes, i.e.,  $h_l = h'_{l-1}/t_l$  and  $w_l = w'_{l-1}/t_l$ <sup>1</sup>. Note that  $k_0 = 1$  and  $h'_0 = w'_0$  are equal to input image resolution.

With this partitioning strategy, COCA operates on a total of  $n_l = h_l \times w_l \times k_{l-1}$  input nodes in each window, then proceeds with affinity masks generation, compactness scoring and sequential clustering. After clustering is completed, COCA has produced an output clusters set comprising  $1 \times 1 \times k_l$  elements. Thus, effectively collapsing spatial dimensions of the each window to obtain  $k_l$  output clusters per window. This means that, for the next COCA layer at

layer  $l + 1$ , the input node set will contain  $t_l \times t_l \times k_l$  elements, i.e.,  $h'_l = t_l$  and  $w'_l = t_l$ . This COCA layer behaves in the exact same manner, first partitions this input set of  $h'_l \times w'_l \times k_l$  elements into  $t_{l+1}^2$  windows, where each window contains  $h_{l+1} \times w_{l+1} \times k_l$  nodes.

### 7.2.3. Affinity Masks Generation (Section 3.1.3)

Inspired by ViT-22B, we employ pre-norm skip connections within a COCA layer, hence, the input refined and partitioned features tensor  $\hat{\mathbf{X}}^l \in \mathbb{R}^{n_l \times d_l}$  is used for skip connection with pooling after clustering is completed. Next, we apply the GroupNorm normalization to this features tensor and then use it as input to three learnable projections, two linear and one non-linear; queries projection  $\mathbf{Q}^l \in \mathbb{R}^{d_l \times d_l}$ , values projection  $\mathbf{V}^l \in \mathbb{R}^{d_l \times d_l}$  and the parallel multi-layer perceptron  $\mathbf{MLP}_1^l \in \mathbb{R}^{d_l \times d_l}$ ,  $\mathbf{MLP}_2^l \in \mathbb{R}^{d_l \times d_l}$ :

$$\hat{\mathbf{X}}_{ij}^l = \text{GroupNorm}_j(\hat{\mathbf{X}}_{ij}^l) \quad (6a)$$

$$\hat{\mathbf{Q}}_i^l = \hat{\mathbf{X}}_i^l \mathbf{Q}^l \quad (6b)$$

$$\hat{\mathbf{V}}_i^l = \hat{\mathbf{X}}_i^l \mathbf{V}^l \quad (6c)$$

$$\hat{\Psi}_i^l = (\text{GELU}(\hat{\mathbf{X}}_i^l \mathbf{MLP}_1^l)) \mathbf{MLP}_2^l \quad (6d)$$

where  $\text{GroupNorm}_j$  indicates that normalization is performed across the last axis of the features tensor. We use the query projected features as our common high-dimensional space to compute affinities on, whereas the value projected features and the features embedded by the parallel MLP will be used after generating cluster assignments (See Fig. 2b).

In order to compute affinities between each pair of nodes within a window, we apply GroupNorm normalization to query projected features:

$$\mathbf{Y}_{ij}^l = \text{GroupNorm}_j(\hat{\mathbf{Q}}_{ij}^l) \quad (7)$$

where  $\mathbf{Y}^l \in \mathbb{R}^{n_l \times d_l}$  denotes the projected and normalized feature vectors of all nodes. Following ViT-22B, we do not learn any parameters for the normalization operations involved throughout COCA layer. After all node features are projected to the same high-dimensional space and normalized, to produce the affinity masks, we measure Euclidean distances between features of each node pair, as given in Eq. 1 of the main paper.

This naive computation of  $\mathbf{E}^l \in \mathbb{R}^{n_l \times n_l}$  uses “all-to-all” affinities to generate candidate masks, however, to improve efficiency, we can optionally reduce the density of the affinity masks using a spatially dilated sampling strategy, producing sparser masks with size  $u_l \times n_l$  where  $u_l \ll n_l$ . As also mentioned in the main paper, we use soft-argmin to convert these distances to dynamic and adaptive affinities and employ min-max scaling to map these affinities to a proper range, i.e. in  $[0, 1]$  for compactness computation

<sup>1</sup>For notational simplicity, we assume that the input image is square rather than rectangle, thus we do not need  $t_l^h$  and  $t_l^w$



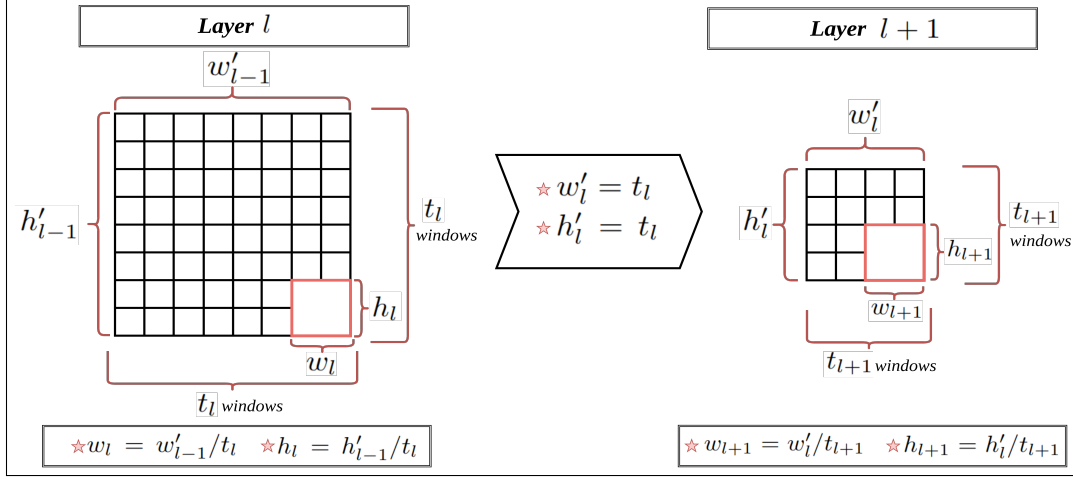


Figure 4. Visualization of the Non-Overlapping partitioning strategy.

and SBC clustering:

$$\tilde{\Lambda}_{ij}^l = \text{soft-argmin}_j (E_{ij}^l) \quad (8a)$$

$$\Lambda_{ij}^l = \frac{\tilde{\Lambda}_{ij}^l - \min_j(\tilde{\Lambda}_{ij}^l)}{\max_j(\tilde{\Lambda}_{ij}^l) - \min_j(\tilde{\Lambda}_{ij}^l)} \quad (8b)$$

#### 7.2.4. Compactness Scoring (Section 3.1.4)

Recall that a single affinity mask  $\Lambda_i^l$ , can be considered as a flattened two dimensional grid of affinities that constitute a non-uniform density shape and that we utilize five physical attributes for each node throughout the hierarchy, i.e., area  $\mathbf{A}^l \in \mathbb{R}^{n_l \times 1}$ , mass  $\mathbf{M}^l \in \mathbb{R}^{n_l \times 1}$ , density  $\mathbf{D}^l \in \mathbb{R}^{n_l \times 1}$ , moment of inertia  $\mathbf{I}^l \in \mathbb{R}^{n_l \times 1}$  and mean position in the original image resolution  $\mathbf{P}^l \in \mathbb{R}^{n_l \times 2}$ . The positions of each input node to the first layer  $\mathbf{P}_i^1$  are initialized with the corresponding pixel coordinates, while the area  $A_i^1$ , mass  $M_i^1$ , and density  $D_i^1$  of nodes are set to ones. In line with [52], the moment of inertia  $I_i^1$  for each pixel is set to  $1/6$ .

As noted in Section 3.1.4, we compute the intermediate attributes by first broadcasting then computing an element-wise product with the affinity masks  $\Lambda^l \in \mathbb{R}^{n_l \times n_l}$ :

$$\tilde{\mathbf{A}}^l = \mathbf{1}^{n_l} (\mathbf{A}^l)^\top \odot \Lambda^l \quad (9a)$$

$$\tilde{\mathbf{D}}^l = \mathbf{1}^{n_l} (\mathbf{D}^l)^\top \odot \Lambda^l \quad (9b)$$

$$\tilde{\mathbf{I}}^l = \mathbf{1}^{n_l} (\mathbf{I}^l)^\top \odot \Lambda^l \quad (9c)$$

$$\tilde{\mathbf{M}}^l = \tilde{\mathbf{A}}^l \odot \tilde{\mathbf{D}}^l \quad (9d)$$

where  $\mathbf{1}^{n_l}$  is a column vector of ones of size  $n_l$ ,  $\odot$  denotes the Hadamard product and  $\tilde{\mathbf{A}}^l, \tilde{\mathbf{D}}^l, \tilde{\mathbf{I}}^l, \tilde{\mathbf{M}}^l \in \mathbb{R}^{n_l \times n_l}$ . Notably, the mass attribute is effectively scaled with the square of the affinities (See Eq. 9b). Empirically, we found that this formulation discourages uncertain affinities in output cluster masks (e.g., an affinity of 0.4 is scaled to 0.16),

while affinity values close to 1 remain only modestly affected. After intermediate attributes computation, we compute the compactness scores  $\mathbf{C}^l \in \mathbb{R}^{n_l}$  of affinity masks using Eq. 3 in the main paper.

#### 7.2.5. Sequential Clustering (Section 3.1.5)

A single cluster mask generation scheme in a COCA layer was discussed in detail in Section 3.1.5 of the main paper. Here, with Algorithm 1, we provide the pseudo-code for the overall compactness-based sequential clustering algorithm.

---

##### Algorithm 1: Sequential Clustering

---

**Input:** Compactness Scores  $\mathbf{C}^l \in \mathbb{R}^{n_l}$ , Affinity Masks  $\Lambda^l \in \mathbb{R}^{n_l \times n_l}$

**Output:** Cluster Masks  $\Pi^l \in \mathbb{R}^{k_l \times n_l}$

```

1 Initialise: Cluster Masks  $\Pi^l = \emptyset$ , Scope  $\mathbf{Z}^l = \mathbf{1}^{n_l}$ 
2 while not stopping condition( $\mathbf{Z}^l$ ) do
3    $\mathbf{C}^l = \mathbf{C}^l \odot \mathbf{Z}^l$ 
4    $\omega = \text{argmax}_i (\mathbf{C}_i^l)$ 
5    $\Pi^l.\text{append}(\Lambda_\omega^l \odot \mathbf{Z}^l)$ 
6    $\mathbf{Z}^l = \mathbf{Z}^l \odot (1 - \Pi_{-1}^l)$ 
7 end
8  $\Pi^l.\text{append}(\mathbf{Z}^l)$ 
```

---

Note that in line 2 of Alg. 1, the ‘stopping condition’ for the original COCA-Net is met when the maximum number of objects in the dataset has been generated. For COCA-Net-Dyna, this condition is satisfied when a stopping condition is met, e.g., the sum of the scope variable drops below 2.5% of its total size. In line 3, we first erode the compactness scores of nodes with an element-wise multiplication with the current scope. Next, in line 4, we find the most compact node among the remaining ones. With line 5, we first fetch the affinity mask that belongs to the most com-

Table 4. Hyper-parameters for Pixel Feature Encoder Backbone across six datasets

Dataset	Channels	Pos. Emb. Channels	MLP Out Channels	Kernel Size	Stride	Padding
Tetrominoes	32	32	64	$3 \times 3$	$1 \times 1$	$1 \times 1$
Multi-dSprites	64	64	64	$3 \times 3$	$1 \times 1$	$1 \times 1$
ShapeStacks	64	64	64	$3 \times 3$	$1 \times 1$	$1 \times 1$
ObjectsRoom	64	64	64	$3 \times 3$	$1 \times 1$	$1 \times 1$
CLEVR6	96	96	96	$5 \times 5$	$1 \times 1$	$2 \times 2$
CLEVRTex	128	128	96	$5 \times 5$	$1 \times 1$	$2 \times 2$

Table 5. Hyper-parameters for Spatial Broadcast Decoder employed in COCA-Net, across six datasets

Dataset	Dataset Resolution	Broadcast Resolution	Channels	Kernel Sizes	Strides	Paddings	Output Paddings
Tetrominoes	$32 \times 32$	$32 \times 32$	$[32, 32, 32, 4]$	$[5, 5, 5, 3]$	$[1, 1, 1, 1]$	$[2, 2, 2, 1]$	$[0, 0, 0, 0]$
Multi-dSprites	$64 \times 64$	$64 \times 64$	$[32, 32, 32, 4]$	$[5, 5, 5, 3]$	$[1, 1, 1, 1]$	$[2, 2, 2, 1]$	$[0, 0, 0, 0]$
ShapeStacks	$64 \times 64$	$64 \times 64$	$[32, 32, 32, 4]$	$[5, 5, 5, 3]$	$[1, 1, 1, 1]$	$[2, 2, 2, 1]$	$[0, 0, 0, 0]$
ObjectsRoom	$64 \times 64$	$64 \times 64$	$[32, 32, 32, 4]$	$[5, 5, 5, 3]$	$[1, 1, 1, 1]$	$[2, 2, 2, 1]$	$[0, 0, 0, 0]$
CLEVR6	$128 \times 128$	$8 \times 8$	$[48, 48, 48, 48, 48, 4]$	$[5, 5, 5, 5, 5, 3]$	$[2, 2, 2, 2, 1, 1]$	$[2, 2, 2, 2, 2, 1]$	$[1, 1, 1, 1, 0, 0]$
CLEVRTex	$128 \times 128$	$8 \times 8$	$[64, 64, 64, 64, 64, 4]$	$[5, 5, 5, 5, 5, 3]$	$[2, 2, 2, 2, 1, 1]$	$[2, 2, 2, 2, 2, 1]$	$[1, 1, 1, 1, 0, 0]$

pact node, conceal it with the current scope and then add it to the list of generated cluster masks. We then update our scope, with line 6, to discard the nodes that are explained in this iteration. After the stopping condition is met, and the loop is finished, we finally add the remaining scope to our cluster masks list and complete the algorithm.

### 7.2.6. Pool, Aggregate and Skip-Connect (Section 3.1.6)

**Pool and Aggregate** The output cluster masks generated at layer  $l$ ,  $\Pi_l \in \mathbb{R}^{k_l \times n_l}$ , provide the COCA layer with the necessary assignments between input nodes and output clusters. Hence, each COCA layer utilizes these output cluster masks to pool attributes from input nodes and construct output cluster attributes. Specifically, we use  $\Pi_l$  to pool a feature vector and five physical attributes per output cluster as:

$$\hat{\mathbf{X}}_i^l = \frac{\Pi_i^l \hat{\mathbf{X}}^l + (\Pi_i^l \hat{\mathbf{V}}^l) \mathbf{O}^l + \Pi_i^l \hat{\mathbf{\Psi}}^l}{\sum_j \Pi_{ij}^l} \quad (10a)$$

$$\mathbf{I}_i^{l+1} = \Pi_i^l \mathbf{I}^l \quad (10b)$$

$$\mathbf{A}_i^{l+1} = \Pi_i^l \mathbf{A}^l \quad (10c)$$

$$\mathbf{M}_i^{l+1} = \Pi_i^l \mathbf{M}^l \quad (10d)$$

$$\mathbf{D}_i^{l+1} = \mathbf{M}_i^{l+1} / \mathbf{A}_i^{l+1} \quad (10e)$$

$$\mathbf{P}_i^{l+1} = \frac{\Pi_i^l \mathbf{P}^l}{\sum_j \Pi_{ij}^l} \quad (10f)$$

where  $\mathbf{O}^l \in \mathbb{R}^{d_l \times d_l}$  represents the output projection, as illustrated in Figure 2b, similar to the one used in ViT-22B layer. Note that  $\hat{\mathbf{X}}_i^l$  still contains the layer index  $l$  since final features of the output clusters will be produced after the inter-layer skip connection operation that will be detailed next.

**Inter-layer Skip Connections** To enhance unsupervised hierarchical feature learning, we leverage inter-layer skip connections starting from the second layer and applied at each layer until the hierarchy is completed. Specifically, in order to incorporate the information from node features at layer  $l - 2$  into cluster features at layer  $l$ , we first merge the cluster masks that are generated between these two layers, namely masks at layer  $l$ ;  $\Pi^l \in \mathbb{R}^{t_l^2 \times k_l \times n_l}$  and masks at layer  $l - 1$ ;  $\Pi^{l-1} \in \mathbb{R}^{t_{l-1}^2 \times k_{l-1} \times n_{l-1}}$ . The masks from the preceding layer,  $l - 1$ , are initially un-flattened, reshaped, and partitioned into non-overlapping windows, with each window sized to match the configuration of the current layer  $l$ . Simultaneously, the masks from the current layer  $l$ , undergo a process of un-flattening and reshaping to align appropriately. These operations yield the intermediate masks,  $\tilde{\Pi}^l \in \mathbb{R}^{t_l^2 \times (h_l w_l) \times k_l \times k_{l-1}}$  and  $\tilde{\Pi}^{l-1} \in \mathbb{R}^{t_l^2 \times (h_l w_l) \times k_{l-1} \times (h_{l-1} w_{l-1} k_{l-2})}$ , ready for merging:

$$\tilde{\Pi}^{(l-1) \rightarrow l} = \tilde{\Pi}^l \tilde{\Pi}^{l-1} \quad (11)$$

where  $\tilde{\Pi}^{(l-1) \rightarrow l}$  can be represented as  $\tilde{\Pi}^{(l-1) \rightarrow l} \in \mathbb{R}^{t_l^2 \times k_l \times (h_l w_l n_{l-1})}$  after few reshape and transpose operations.

To complete the inter-layer skip connections from layer  $l - 2$  to  $l$ , we first skip connect the output cluster features from layer  $l - 2$ ;  $\hat{\mathbf{X}}^{l-2} \in \mathbb{R}^{t_{l-2}^2 \times k_{l-2} \times d_{l-2}}$  and apply appropriate partitioning—two times to match layer  $l$ —and reshaping operations to arrive at  $\hat{\mathbf{X}}^{l-2} \in \mathbb{R}^{t_l^2 \times (h_l w_l n_{l-1}) \times d_{l-2}}$ . Note that we keep the feature dimensions of nodes consistent throughout the hierarchy, similar to ViT-22B and shared in Section 8, thus we have  $d_{l-2} = d_l$ . In addition, note that the output cluster features from layer  $l = 0$  is simply input pixel features generated by the backbone and  $h_0 = 1, w_0 = 1$ . Having generated the necessary merged

Table 6. Hyper-parameters for COCA-Net encoder across six datasets. Here, Q Kernel, Q Stride and Q Pad denote the parameters of unfolding operation applied on the queries of ViT-22B layer. Similarly, K Kernel, K Stride and K Pad stand for unfolding parameters for keys.

Dataset	COCA-Net					ViT-22B Stack						
Name	$L$	$d_l$	$h_l, w_l$	$\tau_l$	$k_l$	# of Layers	Q Kernel	Q Stride	Q Pad	K Kernel	K Stride	K Pad
Tetrominoes	2	[64, 64]	[[4, 4], [8, 8]]	[1.00, 2.00]	[3, 4]	[2, 2]	[4, 8]	[4, 1]	[0, 0]	[4, 8]	[4, 1]	[0, 0]
Multi-dSprites	2	[64, 64]	[[8, 8], [8, 8]]	[1.00, 1.25]	[4, 6]	[3, 3]	[8, 8]	[8, 1]	[0, 0]	[8, 8]	[8, 1]	[0, 0]
ShapeStacks	2	[64, 64]	[[8, 8], [8, 8]]	[0.75, 1.00]	[4, 7]	[3, 3]	[8, 8]	[8, 1]	[0, 0]	[8, 8]	[8, 1]	[0, 0]
ObjectsRoom	2	[64, 64]	[[8, 8], [8, 8]]	[1.00, 1.50]	[4, 7]	[3, 3]	[8, 8]	[8, 1]	[0, 0]	[8, 8]	[8, 1]	[0, 0]
CLEVR6	3	[96, 96, 96]	[[4, 4], [4, 4], [8, 8]]	[1.0, 0.5, 1.0]	[2, 4, 7]	[1, 2, 3]	[4, 4, 8]	[4, 4, 1]	[0, 0, 0]	[4, 4, 8]	[4, 4, 1]	[0, 0, 0]
CLEVRTex	3	[96, 96, 96]	[[4, 4], [4, 4], [8, 8]]	[2.0, 1.0, 0.5]	[2, 4, 11]	[1, 2, 5]	[1, 1, 1]	[1, 1, 1]	[0, 0, 0]	[3, 5, 8]	[1, 1, 1]	[1, 2, 0]

masks and reshaped node features, we compute the skip connection from layer  $l - 2$  to layer  $l$  as:

$$\tilde{\mathbf{X}}^{(l-2) \rightarrow l} = \frac{\tilde{\Pi}^{(l-1) \rightarrow l} \tilde{\mathbf{X}}^{l-2}}{\sum_j \Pi_{ij}^{(l-1) \rightarrow l}} \quad (12)$$

where  $\tilde{\mathbf{X}}^{(l-2) \rightarrow l} \in \mathbb{R}^{t_l^2 \times k_l \times d_l}$ . Finally, we apply Group-Norm normalization to  $\tilde{\mathbf{X}}^{(l-2) \rightarrow l}$ , pass it through an FFN and sum it with the cluster features obtained at layer  $l$  (from Eq. 10a), hence complete cluster feature aggregation from layer  $l$  to  $l + 1$ .

### 7.3. Dendrogram Generation by Merging Cluster Masks

To construct our dendrogram, which represents the object masks eventually generated by the encoder sub-network, we resort to mask merging, similar to the one described in the previous section. Here we progressively merge cluster masks obtained at each level  $l$  of the hierarchy,  $l = 1, \dots, L$ . Specifically, in order to merge the cluster masks that are generated during layer  $l$ ;  $\Pi^l \in \mathbb{R}^{t_l^2 \times k_l \times n_l}$  and merged masks up until layer  $l - 1$ ;  $\Pi^{1 \rightarrow (l-1)} \in \mathbb{R}^{t_{l-1}^2 \times k_{l-1} \times (\prod_{j=1}^{l-1} h_j w_j)}$ , the merged cluster masks from the earlier layer  $l - 1$  is first un-flattened, reshaped and then divided into non-overlapping windows, with a window size that matches the one at layer  $l$ . Meanwhile, the masks from the current layer  $l$  is also un-flattened and reshaped in an appropriate way. These operations yield the reshaped output cluster mask from layer  $l$  as  $\tilde{\Pi}^l \in \mathbb{R}^{t_l^2 \times (h_l w_l) \times k_l \times k_{l-1}}$  whereas the reshaped merged masks in layer  $l - 1$  become  $\tilde{\Pi}^{1 \rightarrow (l-1)} \in \mathbb{R}^{t_l^2 \times (h_l w_l) \times k_{l-1} \times (\prod_{j=1}^{l-1} h_j w_j)}$ . Now both masks contain the appropriate dimensions to carry out the merging operation as:

$$\tilde{\Pi}^{1 \rightarrow l} = \tilde{\Pi}^l \tilde{\Pi}^{1 \rightarrow (l-1)} \quad (13)$$

where  $\tilde{\Pi}^{1 \rightarrow l} \in \mathbb{R}^{t_l^2 \times (h_l w_l) \times k_l \times (\prod_{j=1}^{l-1} h_j w_j)}$ . After appropriate post processing, this merged cluster masks has the dimensions:  $\Pi^{1 \rightarrow l} \in \mathbb{R}^{t_l^2 \times k_l \times (\prod_{j=1}^{l-1} h_j w_j)}$ .

## 8. Implementation Details

For all experiments included in this work, we build on the comprehensive OCL library that is provided by [15]. This OCL library includes the six datasets that we share results on, in addition to code scripts for training, testing and evaluation. Details are included in [15]. We extend this library by adding the implementations of our proposed architecture COCA-Net and three state-of-the-art methods, which are also our baselines: GEN-v2 [18], INVSA [5] and BO-QSA [28]. To add these baselines to our library, we rely on the official implementation of each baseline and adapt these implementations to work within the OCL library. For a fair comparison, we use a Spatial Broadcast Decoder version for each method.

We train all methods, COCA-Net and the baselines, three times using three random seeds. These three seeds are once generated at random before all experiments and then fixed for all the methods. Following the training and evaluation criteria laid out in [15], we train all methods on the same training and validation splits of each dataset, reserving 2000 test images per dataset for evaluation. In each dataset, the maximum number of objects and background segments is used as the number of output slots generated for all methods.

### 8.1. Training the COCA-Net

To optimize the image reconstruction objective on all six datasets, COCA-Net uses the Adam optimizer, employs learning rate linear warm-up and exponential decay scheduling, just as described in the original SA paper [38]. We set the learning rate as 0.0003 and adopt weight decay regularization with the coefficient 0.00001. We train COCA-Net for 500K iterations for each dataset, similar to SA.

For all experiments of the COCA-Net, we use data augmentation as to “pad and random crop” input images. This augmentation first pads the image with a small number of repeated pixels (e.g., three) on each edge and randomly crops this padded image with a window that has the same resolution as the original image. The motivation behind this particular augmentation strategy is to present each COCA

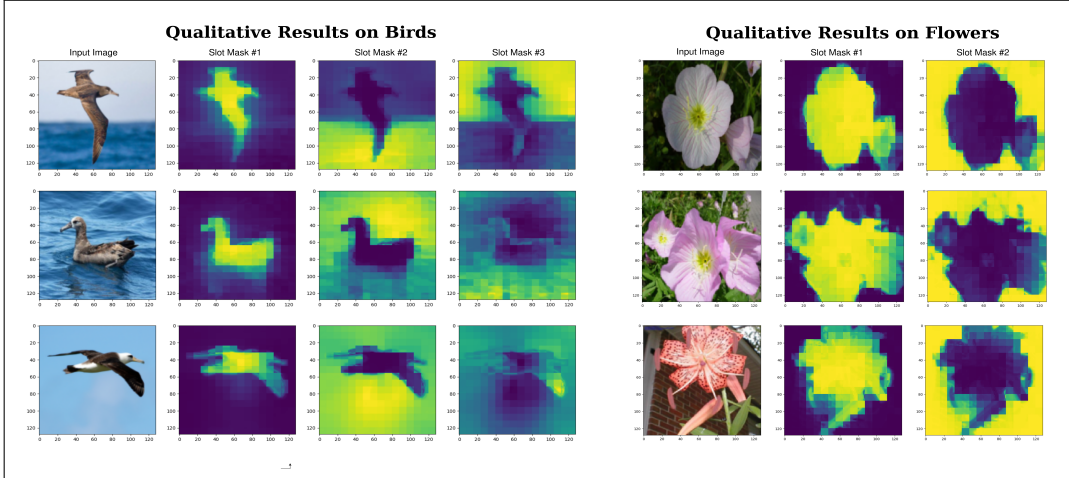


Figure 5. Qualitative results obtained for real-world datasets. Slot masks predicted by COCA-Net Encoder are shared for Birds (left) and Flowers (right).

layer with a randomly shifted non-overlapping partitions, effectively increasing the window variability that COCA-Net learns from.

Tables 4 and 5 list the hyper-parameters used across all six datasets for the Pixel Feature Encoder Backbone and the Spatial Broadcast Decoder, respectively. Table 6 presents the hyper-parameters employed for COCA-Net on each dataset.

## 8.2. Training the Baseline Models

For all baselines, we use the default configurations and hyper-parameters that are laid out in their original papers. Unlike some of our baselines, we train all methods to produce output object masks that match the ground-truth maximum number of objects in a dataset. In addition, due to computational constraints, we fix the batch size as 64 for all methods.

## 9. Additional Results

### 9.1. Initial Experiments on Real-World Datasets; Birds and Flowers

We evaluated COCA-Net on Birds<sup>2</sup> and Flowers<sup>3</sup> (following BOQSA’s protocol) using the SLATE<sup>4</sup> architecture which leverages a transformer decoder but no pretrained backbone. Entire architecture is trained from scratch. Both datasets contain real-world images with compositionality (objects contain a spectrum of colors and deformable shapes). Table 7 shows COCA-Net is on par or better than BOQSA across IoU and Dice metrics. Figure 5 displays qualitative results, with COCA-Net achieving robust

segmentation masks for both foreground objects and background segments. Since these are our initial experiments with COCA-Net integrated into the SLATE architecture, we believe that the performance can be further boosted with more elaborate hyperparameter tuning.

### 9.2. Additional Quantitative Results

Here, we present Table 8, an extension of Table 1 that was introduced in the main paper. Table 8 provides additional quantitative results of COCA-Net and its baselines obtained on the Tetrominoes and Multi-dSprites datasets.

### 9.3. Ablation Studies

We now present additional quantitative results to complement those in Table 3 of the main paper. Table 9 extends our comparison between COCA-Net and its random anchor node selection variant, COCA-Net-RAS. Further results for COCA-Net and COCA-Net-Dyna are provided in Table 10.

### 9.4. Additional Qualitative Results

We share the supplementary qualitative results of COCA-Net, starting from Figure 6 to Figure 11 for all six datasets. In each figure, we share a sample batch of images from the corresponding dataset and include COCA-Net’s reconstructions of these images next to it. In addition, we visualize the segmentation masks that COCA-Net produces in its encoder and decoder sub-networks.

<sup>2</sup>Welinder, P. et al. “Caltech-UCSD Birds 200.” 2010.

<sup>3</sup>Nilsback, M. and A. Zisserman. “Delving into the Whorl of Flower Segmentation.” BMVC 2007.

<sup>4</sup>Singh, G. et al. “Illiterate DALL-E Learns to Compose.” ICLR 2022.



Table 7. Quantitative Results for COCA-Net and BOQSA on Birds and Flowers datasets. Background segments included, single seed set to BOQSA’s open-source implementation.

Model	Flowers		Birds	
	IoU↑	Dice↑	IoU↑	Dice↑
BOQSA	0.6477	0.7603	0.5565	0.7032
COCA-Net (ours)	<b>0.6482</b>	<b>0.7612</b>	<b>0.5778</b>	<b>0.7238</b>

Table 8. Supplementary unsupervised scene segmentation results of three baseline models and proposed COCA-Net on Tetrominoes and Multi-dSprites datasets, based on nine performance evaluation metrics, assessed across four different configurations. Scores are reported as mean  $\pm$  standard deviation for 3 seeds.

Name	DEC-FG Only		DEC-BG Included		ENC-FG Only		ENC-BG Included		MSE↓
	ARI↑	mSC↑	ARI↑	mSC↑	ARI↑	mSC↑	ARI↑	mSC↑	
Tetrominoes									
GEN-v2 [18]	0.33±0.47	0.17±0.10	0.03±0.04	0.27±0.03	0.13±0.18	0.13±0.05	0.02±0.02	0.24±0.02	0.004±0.000
INV-SA [5]	0.98±0.01	0.97±0.00	0.97±0.00	0.98±0.00	0.38±0.08	0.59±0.05	0.61±0.04	0.65±0.04	0.001±0.000
BOQ-SA [28]	<b>0.99±0.01</b>	0.31±0.00	0.11±0.01	0.33±0.01	0.60±0.03	0.42±0.01	0.29±0.02	0.47±0.00	<b>0.000±0.000</b>
COCA-Net (ours)	<b>0.99±0.01</b>	<b>0.98±0.01</b>	<b>0.99±0.01</b>	<b>0.99±0.01</b>	<b>0.74±0.04</b>	<b>0.70±0.07</b>	<b>0.71±0.09</b>	<b>0.75±0.07</b>	0.001±0.000
Multi-dSprites									
GEN-v2 [18]	0.80±0.03	0.58±0.04	0.71±0.02	0.66±0.03	0.60±0.02	0.15±0.02	0.04±0.01	0.22±0.01	0.007±0.000
INV-SA[5]	0.90±0.00	0.84±0.01	0.71±0.35	0.84±0.05	0.68±0.04	0.47±0.00	0.41±0.26	0.53±0.06	<b>0.001±0.000</b>
BOQ-SA [28]	0.89±0.00	0.65±0.07	0.42±0.17	0.65±0.10	0.75±0.01	0.55±0.01	0.34±0.06	0.56±0.02	<b>0.001±0.000</b>
COCA-Net (ours)	<b>0.95±0.00</b>	<b>0.91±0.01</b>	<b>0.84±0.19</b>	<b>0.91±0.03</b>	<b>0.96±0.01</b>	<b>0.95±0.01</b>	<b>0.98±0.00</b>	<b>0.96±0.00</b>	0.002±0.000

Table 9. Extension of Table 3 from the main paper, comparing the unsupervised scene segmentation performance of COCA-Net and COCA-Net-RAS on two datasets. The comparison is based on nine performance evaluation metrics assessed across four different configurations, with scores reported for the same single seed.

Name	DEC-FG Only		DEC-BG Included		ENC-FG Only		ENC-BG Included		MSE↓
	ARI↑	mSC↑	ARI↑	mSC↑	ARI↑	mSC↑	ARI↑	mSC↑	
ObjectsRoom									
COCA-Net-RAS	0.832	0.739	0.561	0.581	0.763	0.277	0.469	0.368	<b>0.001</b>
COCA-Net	<b>0.894</b>	<b>0.832</b>	<b>0.960</b>	<b>0.889</b>	<b>0.879</b>	<b>0.823</b>	<b>0.952</b>	<b>0.881</b>	<b>0.001</b>
ShapeStacks									
COCA-Net-RAS	0.834	0.735	<b>0.267</b>	0.728	0.747	0.385	0.157	0.419	0.005
COCA-Net	<b>0.916</b>	<b>0.857</b>	0.230	<b>0.782</b>	<b>0.843</b>	<b>0.865</b>	<b>0.209</b>	<b>0.772</b>	<b>0.004</b>

Table 10. Extension of Table 3 from the main paper, comparing the unsupervised scene segmentation performance of COCA-Net and COCA-Net-Dyna on two datasets. The comparison is based on nine performance evaluation metrics assessed across four different configurations. All scores are based on results obtained using the same random seed. The average number of slots used is provided in the second column. Note that for the CLEVR dataset, COCA-Net is trained and evaluated on CLEVR6, whereas COCA-Net-Dyna is trained on CLEVR6 but evaluated on CLEVR10.

Name	Avg. Slots	DEC-FG Only		DEC-BG Included		ENC-FG Only		ENC-BG Included		MSE↓
		ARI↑	mSC↑	ARI↑	mSC↑	ARI↑	mSC↑	ARI↑	mSC↑	
CLEVR										
COCA-Net	11	0.982	0.881	0.929	0.900	0.975	0.756	0.849	0.795	0.000
COCA-Net-Dyna	7.49	0.978	0.840	0.911	0.859	0.966	0.741	0.846	0.773	0.001
ShapeStacks										
COCA-Net	7	0.916	0.857	0.230	0.782	0.843	0.865	0.209	0.772	0.004
COCA-Net-Dyna	5.35	0.896	0.818	0.232	0.754	0.827	0.823	0.210	0.746	0.005

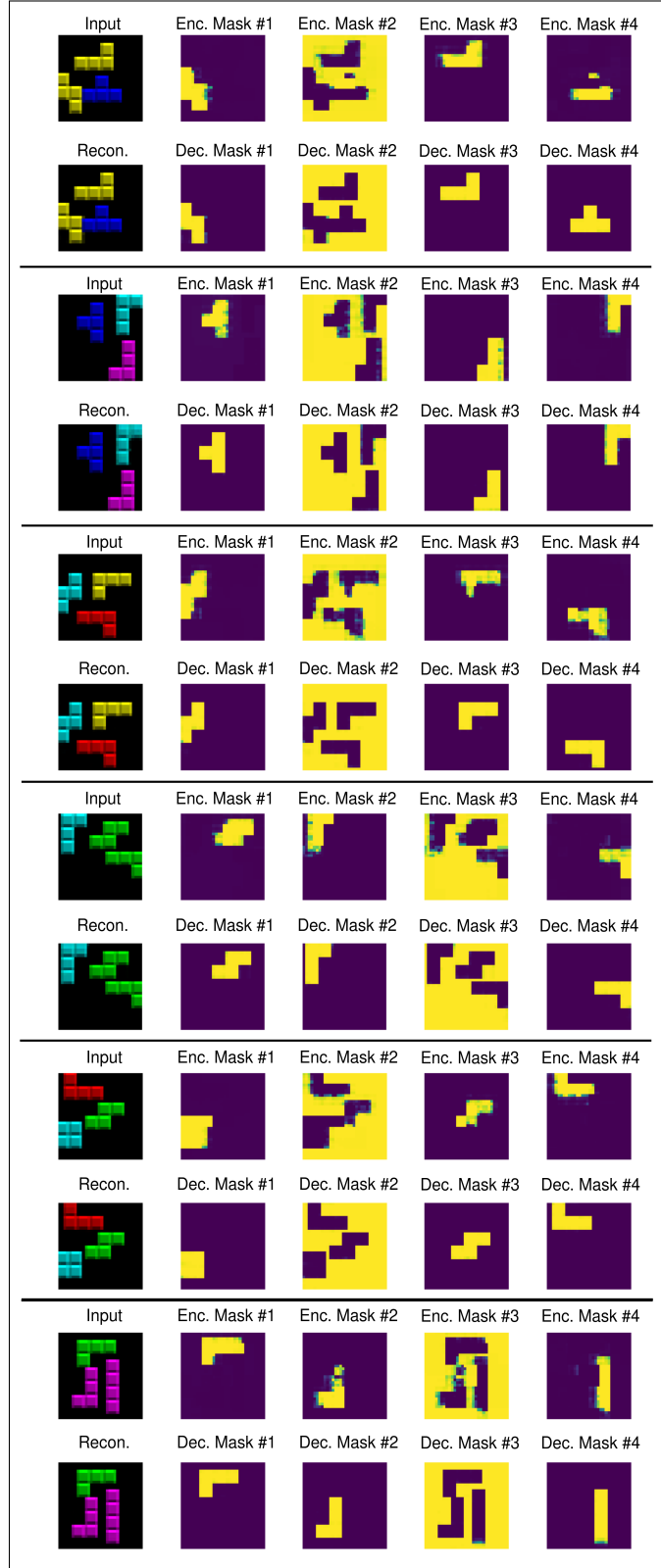


Figure 6. Qualitative results of COCA-Net on Tetrominoes dataset. Accompanying each input image, we visualize COCA-Net’s reconstruction, segmentation masks generated by its encoder sub-network, as well as segmentation masks predicted by the decoder sub-network.

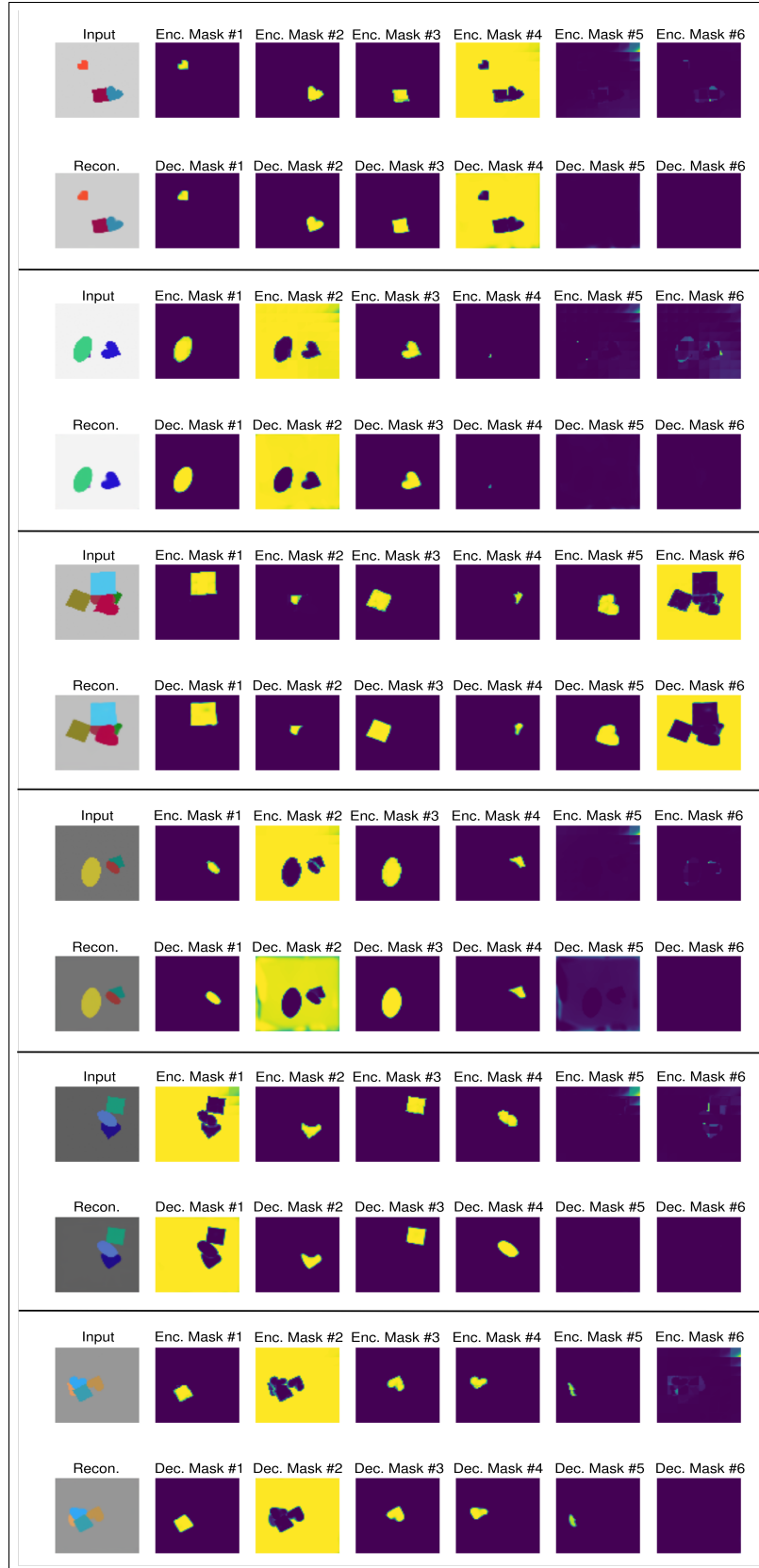


Figure 7. Qualitative results of COCA-Net on Multi-dSprites dataset. Accompanying each input image, COCA-Net’s reconstruction, segmentation masks generated by its encoder sub-network, as well as segmentation masks predicted by the decoder sub-network are visualized.

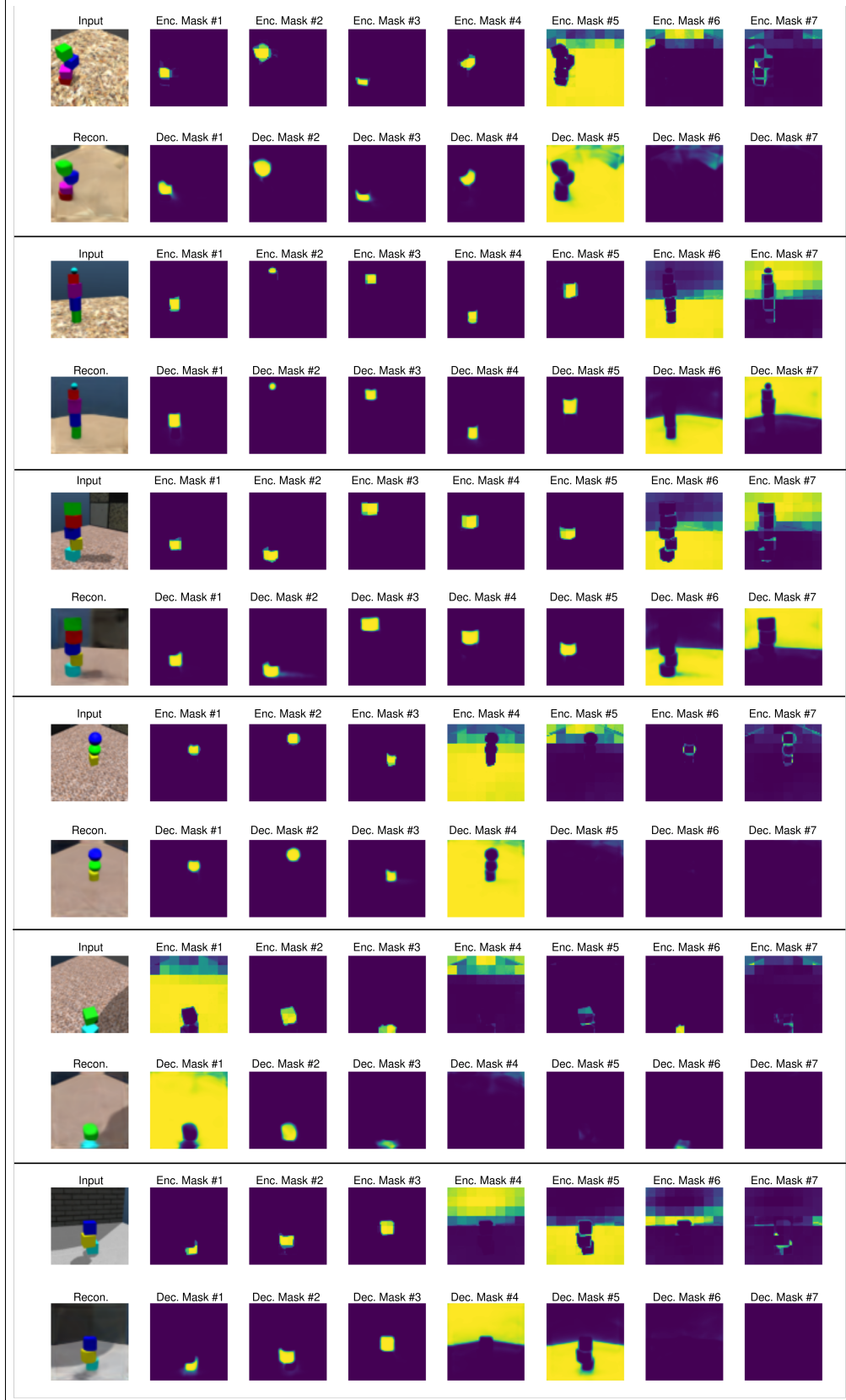


Figure 8. Qualitative results of COCA-Net on Shapestacks dataset. With each input image, we visualize COCA-Net’s reconstruction, segmentation masks generated by its encoder sub-network, as well as segmentation masks predicted by the decoder sub-network.



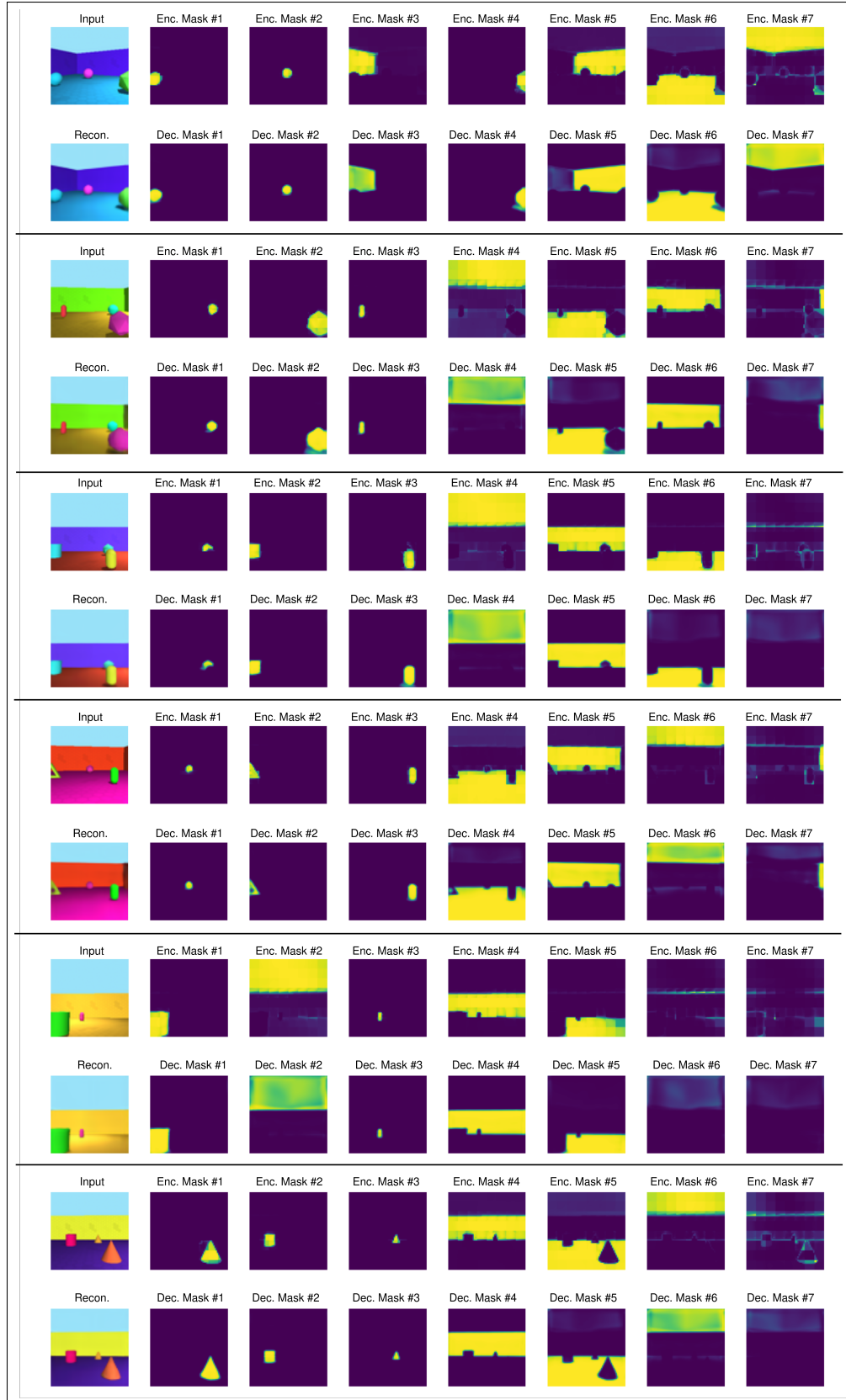


Figure 9. Qualitative results of COCA-Net on ObjectRoom dataset. With each input image, we visualize COCA-Net’s reconstruction, segmentation masks generated by its encoder sub-network, as well as segmentation masks predicted by the decoder sub-network.

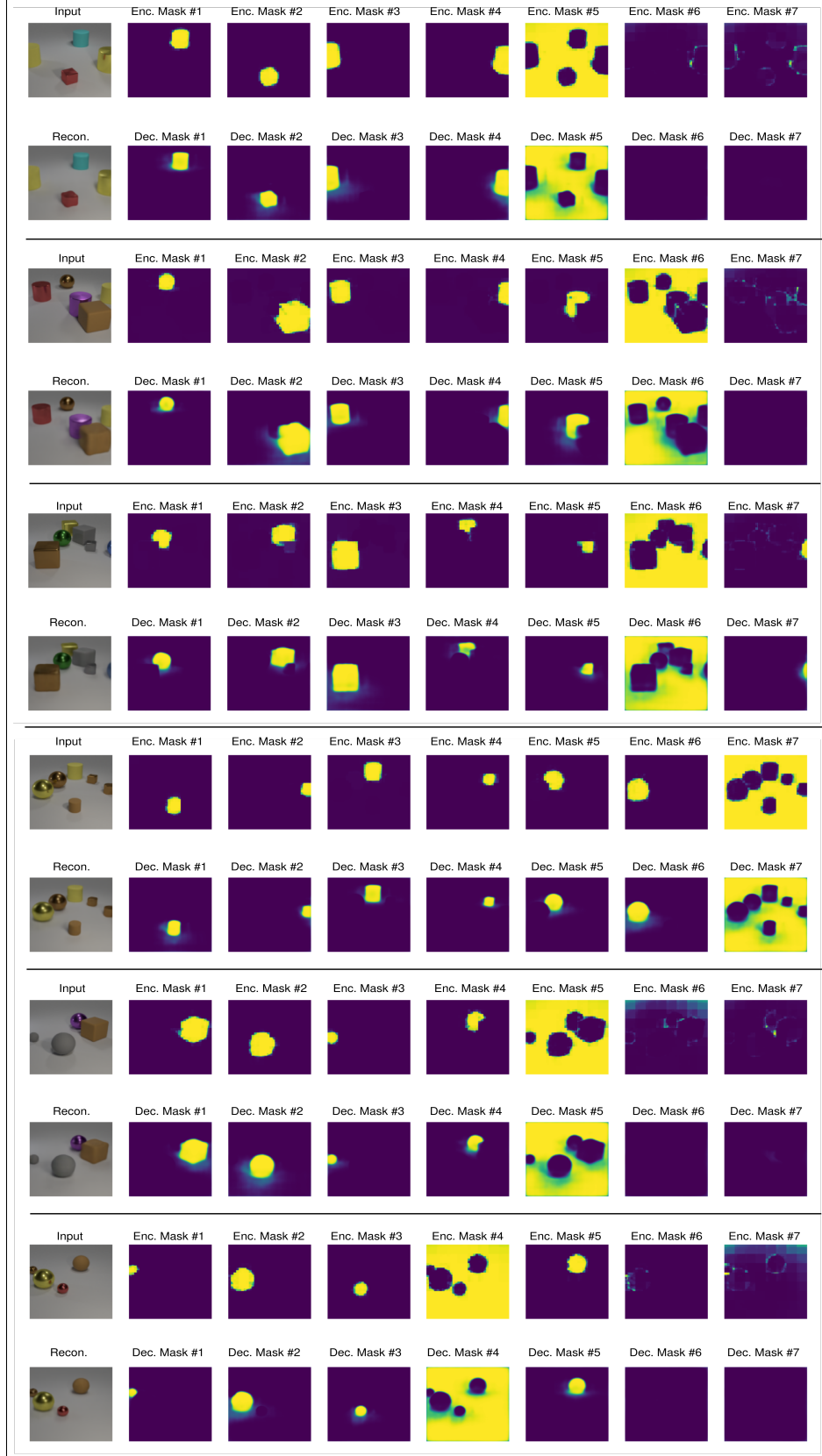


Figure 10. Qualitative results of COCA-Net on CLEVR6 dataset. With each input image, we visualize COCA-Net’s reconstruction, segmentation masks generated by its encoder sub-network, as well as segmentation masks predicted by the decoder sub-network.

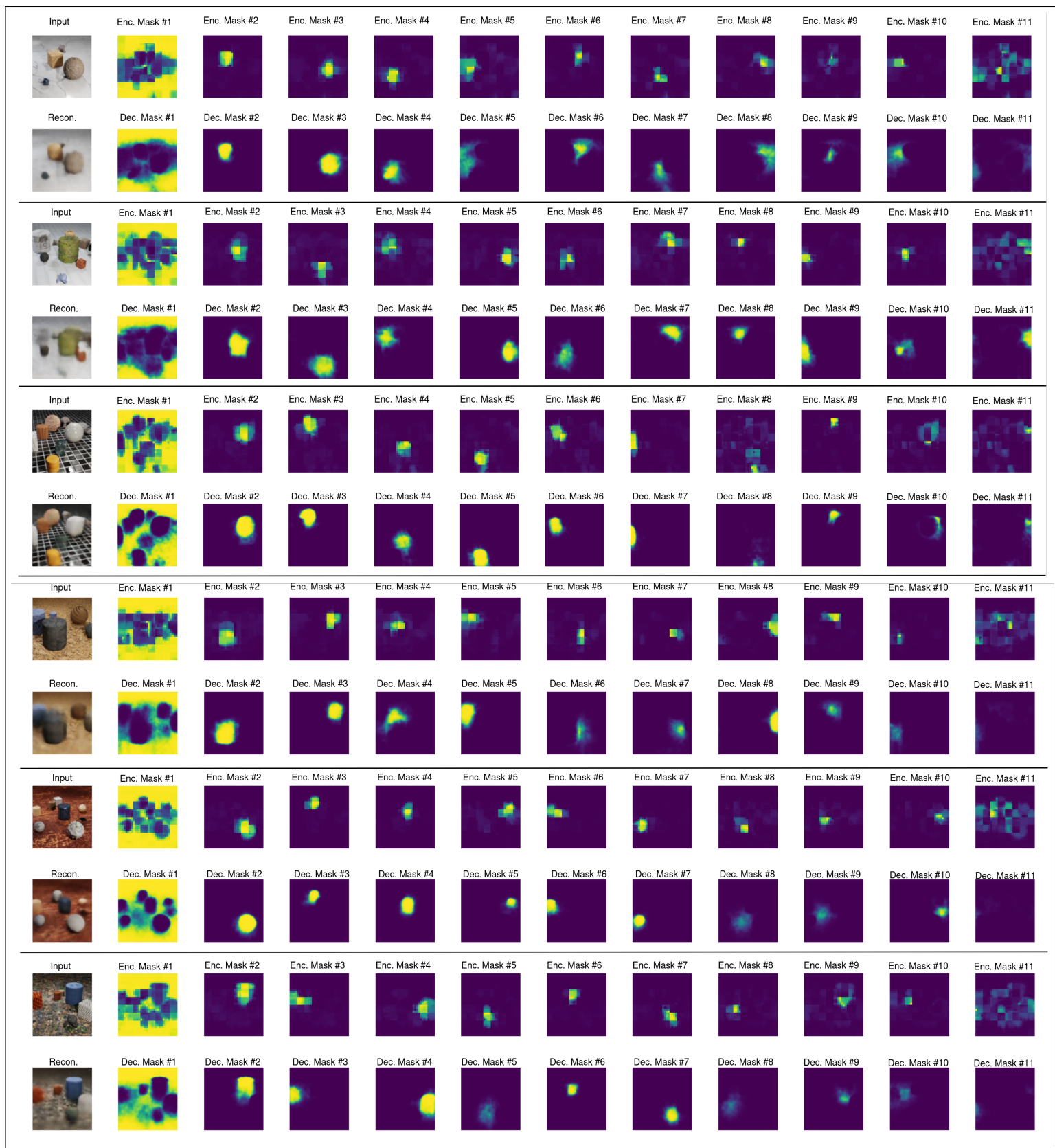


Figure 11. Qualitative results of COCA-Net on CLEVRText dataset. With each input image, we visualize COCA-Net’s reconstruction, segmentation masks generated by its encoder sub-network, as well as segmentation masks predicted by the decoder sub-network.