Text2Scene: Generating Compositional Scenes from Textual Descriptions
Supplementary Material

Fuwen Tan[1] Song Feng[2] Vicente Ordonez[1]
[1]University of Virginia, [2]IBM Thomas J. Watson Research Center.
fuwen.tan@virginia.edu, sfeng@us.ibm.com, vicente@virginia.edu

1. Network Architecture

Here we describe the network architectures for the components of our model in different tasks.

1.1. Text Encoder

We use the same network architecture for the text encoders in all our experiments, which consists of a single layer bidirectional recurrent network with Gated Recurrent Units (GRUs). It takes a linear embedding of each word as input and has a hidden dimension of 256 for each direction. We initialize the word embedding network with the pre-trained parameters from GloVe [7]. The word embedding vectors are kept fixed for abstract scene and semantic layout generations but finetuned for synthetic image generation.

1.2. Scene Encoder

The scene encoder Ω for abstract scene generation is an Imagenet (ILSVRC) pre-trained ResNet-34 [1]. Its parameters are fixed in all the experiments on Abstract Scene [11]. For layout and synthetic image generations, we develop our own scene encoders as the inputs for these tasks are not RGB images.

Table 1 and 2 show the architecture details. Here $|\mathcal{V}|$ is the size of the categorical vocabulary. In the layout generation task, $|\mathcal{V}|$ is 83, including 80 object categories in COCO [5] and three special categorical tokens: sos, eos, pad, representing the start and end points for sequence generation and the padding token. For synthetic image generation, $|\mathcal{V}|$ is 98, including 80 object categories in COCO [5], 15 supercategories for stuffs in COCO-stuff [2] and the special categorical tokens: sos, eos, pad.

As described in the main paper, the input for synthetic image generation has a layer-wise structure where every three channels contain the color patches of a specific category from the background canvas image. In this case, the categorical information of the color patches can be easily learned. On the other hand, since the input is a large but sparse volume with very few non-zero values, to reduce the number of parameters and memory usage, we use a depthwise separable convolution as the first layer of Ω (index (2)), where each group of three channels (g3) is convolved to one single channel in the output feature map.

Index	Input	Operation	Output Shape
(1)	-	Input	$\|\mathcal{V}\| \times 64 \times 64$
(2)	(1)	Conv(7×7, $\|\mathcal{V}\| \rightarrow 128$, s2)	$128 \times 32 \times 32$
(3)	(2)	Residual($128 \rightarrow 128$, s1)	$128 \times 32 \times 32$
(4)	(3)	Residual($128 \rightarrow 256$, s2)	$256 \times 16 \times 16$
(5)	(4)	Bilateral upsampling	$256 \times 28 \times 28$

Table 1. Architecture of our scene encoder Ω for layout generation. We follow the notation format used in [3]. Here $|\mathcal{V}|$ is the size of the categorical vocabulary. The input and output of each layer have a shape of $C \times H \times W$, where C is the number of channels and H and W are the height and width. The notation *Conv*($K \times K$, $C_{in} \rightarrow C_{out}$) represents a convolutional layer with $K \times K$ kernels, C_{in} input channels and C_{out} output channels. The notation s2 means the convolutional layer has a stride of 2. The notation *Residual*($C_{in} \rightarrow C_{out}$) is a residual module consisting of two 3×3 convolutions and a skip-connection layer. In the first residual block (index (3)), the skip-connection is an identity function and the first convolution has a stride of 1 (s1). In the second residual block (index (4)), the skip-connection is a 1×1 convolution with a stride of 2 (s2) and the first convolution also has a stride of 2 to downsample the feature map. Here all the convolutional layers are followed by a ReLU activation.

Index	Input	Operation	Output Shape
(1)	-	Input	$3\|\mathcal{V}\| \times 128 \times 128$
(2)	(1)	Conv(7×7, $3\|\mathcal{V}\| \rightarrow \|\mathcal{V}\|$, s2, g3)	$\|\mathcal{V}\| \times 64 \times 64$
(3)	(2)	Residual($\|\mathcal{V}\| \rightarrow \|\mathcal{V}\|$, s1)	$\|\mathcal{V}\| \times 64 \times 64$
(4)	(3)	Residual($\|\mathcal{V}\| \rightarrow 2\|\mathcal{V}\|$, s1)	$2\|\mathcal{V}\| \times 64 \times 64$
(5)	(4)	Residual($2\|\mathcal{V}\| \rightarrow 2\|\mathcal{V}\|$, s1)	$2\|\mathcal{V}\| \times 64 \times 64$
(6)	(5)	Residual($2\|\mathcal{V}\| \rightarrow 3\|\mathcal{V}\|$, s2)	$3\|\mathcal{V}\| \times 32 \times 32$
(7)	(6)	Residual($3\|\mathcal{V}\| \rightarrow 3\|\mathcal{V}\|$, s1)	$3\|\mathcal{V}\| \times 32 \times 32$
(8)	(7)	Residual($3\|\mathcal{V}\| \rightarrow 4\|\mathcal{V}\|$, s1)	$4\|\mathcal{V}\| \times 32 \times 32$

Table 2. Architecture of our scene encoder Ω for synthetic image generation. The notations are in the same format of Table 1. The first convolution (index (2)) is a depthwise separable convolution where each group of three channels (g3) is convolved to one single channel in the output feature map. All the convolutional layers are followed by a LeakyReLU activation with a negative slope of 0.2.

1.3. Convolutional Recurrent Module

The scene recurrent module for all our experiments is a convolutional GRU network [12] with one ConvGRU cell. Each convolutional layer in this module have a 3×3 kernel with a stride of 1 and a hidden dimension of 512. We pad the input of each convolution so that the output feature map has the same spatial resolution as the input. The hidden state is initialized by spatially replicating the last hidden state from the text encoder.

1.4. Object and Attribute Decoders

Table 3 shows the architectures for our object and attribute decoders. Ψ^o and Ψ^a are the spatial attention modules consisting of two convolutional layers. Θ^o is a two-layer perceptron predicting the likelihood of the next object using a softmax function. Θ^a is a four-layer CNN predicting the likelihoods of the location and attributes of the object. As explained in the main paper, the output of Θ^a has $1 + \sum_k |R^k|$ channels, where $|R^k|$ denotes the discretized range of the k-th attribute, or the dimension of the appearance vector Q_t used as the query for patch retrieval for synthetic image generation. The first channel of the output from Θ^a predicts the location likelihoods which are normalized over the spatial domain using a softmax function. The rest channels predict the attributes for every grid location. During training, the likelihoods from the ground-truth locations are used to compute the loss. At each step of the test time, the top-1 location is first sampled from the model. The attributes are then collected from this sampled location. The text-based attention modules are defined similarly as in [6]. When denoting $d_i = [h_i^E; x_i]$, $s_t^o = [u_t^o; o_{t-1}]$, and $s_t^a = o_t$, Φ^o and Φ^a are defined as:

$$
\begin{aligned}
c_t^* = \Phi^*(s_t^*, \{d_i\}) &= \sum_i \frac{\exp(\text{score}(s_t^*, d_i))}{\sum_j \exp(\text{score}(s_t^*, d_j))} \cdot d_i \\
\text{score}(s_t^*, d_k) &= (s_t^*)^\intercal W_\Phi^* d_k, \quad * \in o, a
\end{aligned}
$$

Here, W_Φ^o and W_Φ^a are trainable matrices which learn to compute the attention scores for collecting the context vectors c_t^o and c_t^a.

These architecture designs are used for all the three generation tasks. The only difference is the grid resolution (H, W). For abstract scene and layout generations, (H, W) = (28, 28). For synthetic image generation, (H, W) = (32, 32). Note that, although our model uses a fixed grid resolution, the composition can be performed on canvases of different sizes.

1.5. Foreground Patch Embedding

The foreground segment representation we use is similar with the one in [8], where each segment P is represented by a tuple (P^{color}, P^{mask}, $P^{context}$). Here $P^{color} \in \mathbb{R}^{3 \times H \times W}$ is a color patch containing the segment, $P^{mask} \in \{0, 1\}^{1 \times H \times W}$ is a binary mask indicating the foreground region of P^{color}, $P^{context} \in \{0, 1\}^{|\mathcal{V}| \times H \times W}$ is a semantic map representing the semantic context around P. The context region of P is obtained by computing the bounding box of the segment and enlarging it by 50% in each direction.

Module	Index	Input	Operation	Output Shape
Ψ^o	(1)	-	Conv(3×3, 512→256)	256 × H × W
	(2)	(1)	Conv(3×3, 256→1)	1 × H × W
Ψ^a	(1)	-	Conv(3×3, 1324→256)	256 × H × W
	(2)	(1)	Conv(3×3, 256→1)	1 × H × W
Θ^o	(1)	-	Linear((1324 + $\lvert\mathcal{V}\rvert$)→512)	512
	(2)	(1)	Linear(512→ $\lvert\mathcal{V}\rvert$)	$\lvert\mathcal{V}\rvert$
Θ^a	(1)	-	Conv(3×3, (1324+$\lvert\mathcal{V}\rvert$)→512)	512 × H × W
	(2)	(1)	Conv(3×3, 512→256)	256 × H × W
	(3)	(2)	Conv(3×3, 256→256)	256 × H × W
	(4)	(3)	Conv(3×3, 256→(1 + $\sum_k \lvert R^k\rvert$))	(1 + $\sum_k \lvert R^k\rvert$) × H × W

Table 3. Architectures for the object and attribute decoders. The notation *Linear*($C_{in} \rightarrow C_{out}$) represents a fully connected layer with C_{in} input channels and C_{out} output channels. All layers, except the last layer of each module, are followed by a ReLU activation.

Index	Input	Operation	Output Shape
(1)	-	Input layout	($\lvert\mathcal{V}\rvert$ + 4) × 64 × 64
(2)	(1)	Conv(2 × 2, ($\lvert\mathcal{V}\rvert$ + 4) → 256, s2)	256 × 32 × 32
(3)	(2)	Conv(2 × 2, 256 → 256, s2)	256 × 16 × 16
(4)	(3)	Conv(2 × 2, 256 → 256, s2)	256 × 8 × 8
(5)	(4)	Conv(2 × 2, 256 → 256, s2)	256 × 4 × 4
(6)	(5)	Conv(2 × 2, 256 → 128, s2)	256 × 2 × 2
(7)	(6)	Global average pooling	256
(8)	-	Input patch feature	2048
(9)	(7)(8)	Linear((256 + 2048) → 128)	128

Table 4. Architecture of our foreground patch embedding network for synthetic image generation. All the convolutional layers are followed by a LeakyReLU activation with a negative slope of 0.2.

Table 4 shows the architecture of our foreground patch embedding network. Here, the concatenation of (P^{color}, P^{mask}, $P^{context}$) is fed into a five-layer convolutional network which reduces the input into a 1D feature vector F_s (index (7)). As this convolutional backbone is relatively shallow, F_s is expected to encode the shape, appearance, and context, but may not capture the fine-grained semantic attributes of P. In our experiments, we find that incorporating the knowledge from the pre-trained deep features of P^{color} can help retrieve segments associated with strong semantics, such as the "person" segments. Therefore, we also use the pre-trained features F_d (index (8)) of P^{color} from the mean pooling layer of ResNet152 [1], which has 2048 features. The final vector F_t is predicted from the concatenation of (F_s, F_d) by a linear regression.

1.6. Inpainting Network

Our inpainting network has the same architecture as the image synthesis module proposed in [8], except that we exclude all the layer-normalization layers. To generate the simulated canvases on COCO, we follow the procedures proposed in [8], but make minor modifications: (1) we use

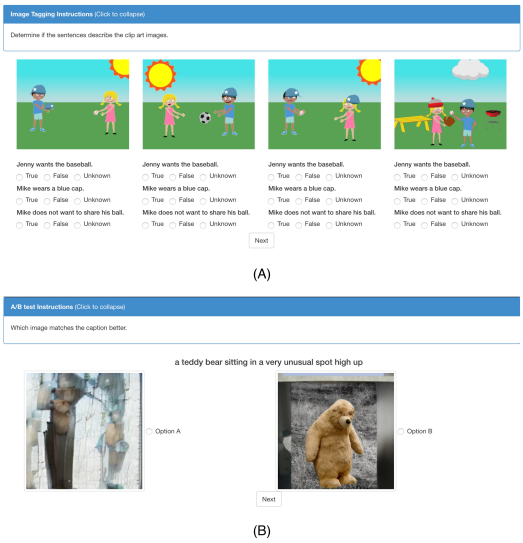

Figure 1. Screen shots of the user interfaces for our human subject studies on Amazon Mechanical Turk. (A) User interface for the evaluation study of the abstract scene generation experiment; (B) User interface for the evaluation study of the synthetic image generation experiment.

the trained embedding patch features to retrieve alternative segments to stencil the canvas, instead of the intersection-over-union based criterion used in [8]. (2) we do not perform boundary elision for the segments as it may remove fine grained details of the segments such as human faces.

2. Optimization

For optimization we use Adam [4] with an initial learning rate of $5e-5$. The learning rate is decayed by 0.8 every 3 epochs. We clip the gradients in the back-propagation such that the norm of the gradients is not larger than 10. Models are trained until validation errors stop decreasing. For abstract scene generation, we set the hyperparameters (w_o, w_l, w_{pose}, $w_{expression}$, w_{size}, $w_{direction}$, w_a^O, w_a^A) to (8,2,2,2,1,1,1,1). For semantic layout generation, we set the hyperparameters (w_o, w_l, w_{size}, w_{aratio}, w_a^O, w_a^A) to (5,2,2,2,1,0). For synthetic image generation, we set the hyperparameters (w_o, w_l, w_{size}, w_{aratio}, w_a^O, w_a^A, w_e, α) to (5,2,2,2,1,0,10,0.5). The hyperparameters are chosen to make the losses of different components comparable. Exploration of the best hyperparameters is left for future work.

3. User Study

We conduct two user studies on Amazon Mechanical Turk (AMT).

The first user study is to evaluate if the generated clip-art scenes match the input sentences. To this end, we randomly select 100 groups of images generated from the sentences in the test set. Each group consists of three images generated by different models, and the ground truth reference image. During the study, these images and the corresponding sentences are presented in random orders. The human annotators are asked to determine if the entailment between the generated scene and the sentence is `true`, `false` or `uncertain`. Each group of images is seen by three annotators. We ignore the `uncertain` responses and report the results using majority opinions. Figure 1 (A) shows the user interface of this study.

The second user study is on the synthetic image generation task, where we compare the generated images from our model and three state-of-the-art approaches: SG2IM [3], HDGAN [10], and AttnGAN [9]. In each round of the study, the human annotator is presented with one sentence and two generated images: one from our model, the other from an alternative approach. The orders of the images are randomized. We ask the human annotator to select the image which matches the sentence better. In total, we collect results for 500 sentences randomly selected from the test set, using five annotators for each. Figure 1 (B) shows the user interface of this study.

4. More qualitative examples

4.1. Abstract Scene

We present more qualitative examples on Abstract Scene [11] in Fig. 2. The examples show that our model does not simply replicate the ground truth reference scenes, but generates dramatically different clip-arts which still match the input textual descriptions.

4.2. Layout Generation

We present more qualitative examples for layout generation in Fig. 3. The examples include various scenes containing different object categories. Our model manages to learn important semantic concepts from the language, such as the presence and count of the objects, and their spatial relations.

4.3. Synthetic Image Generation

To demonstrate our model does not learn an image-level retrieval on the training set, we present in Fig. 4 the generated images and the corresponding source images from which the patch segments are retrieved for compositing. For each generated image, we show three source images for clarity. The examples illustrate that our model learns not only the presence and spatial layout of objects, but also the semantic knowledge that helps retrieve segments in similar contexts. Fig. 5 shows more qualitative examples of our model for synthetic image generation.

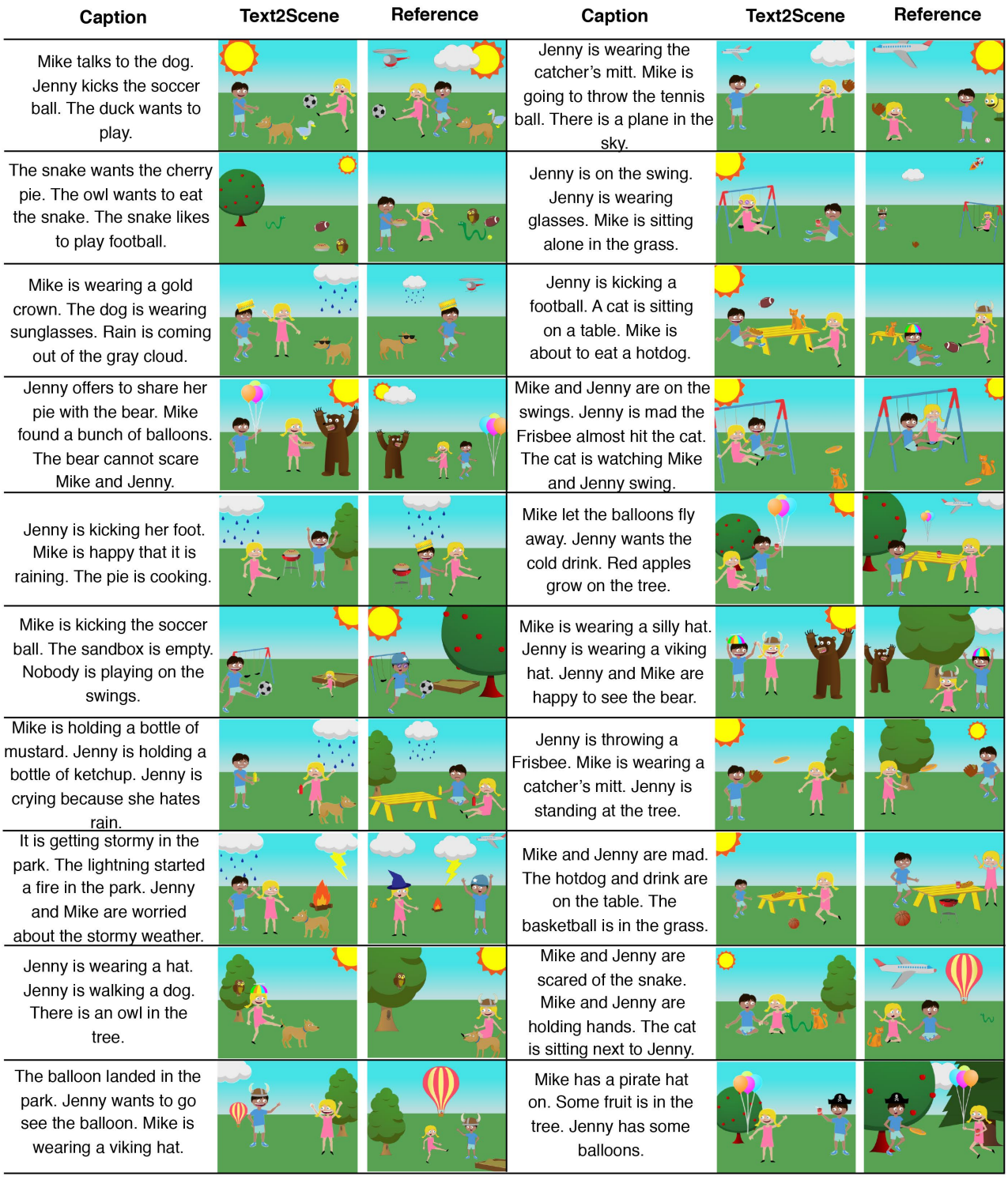

Figure 2. More qualitative examples of the abstract scene generation experiment.

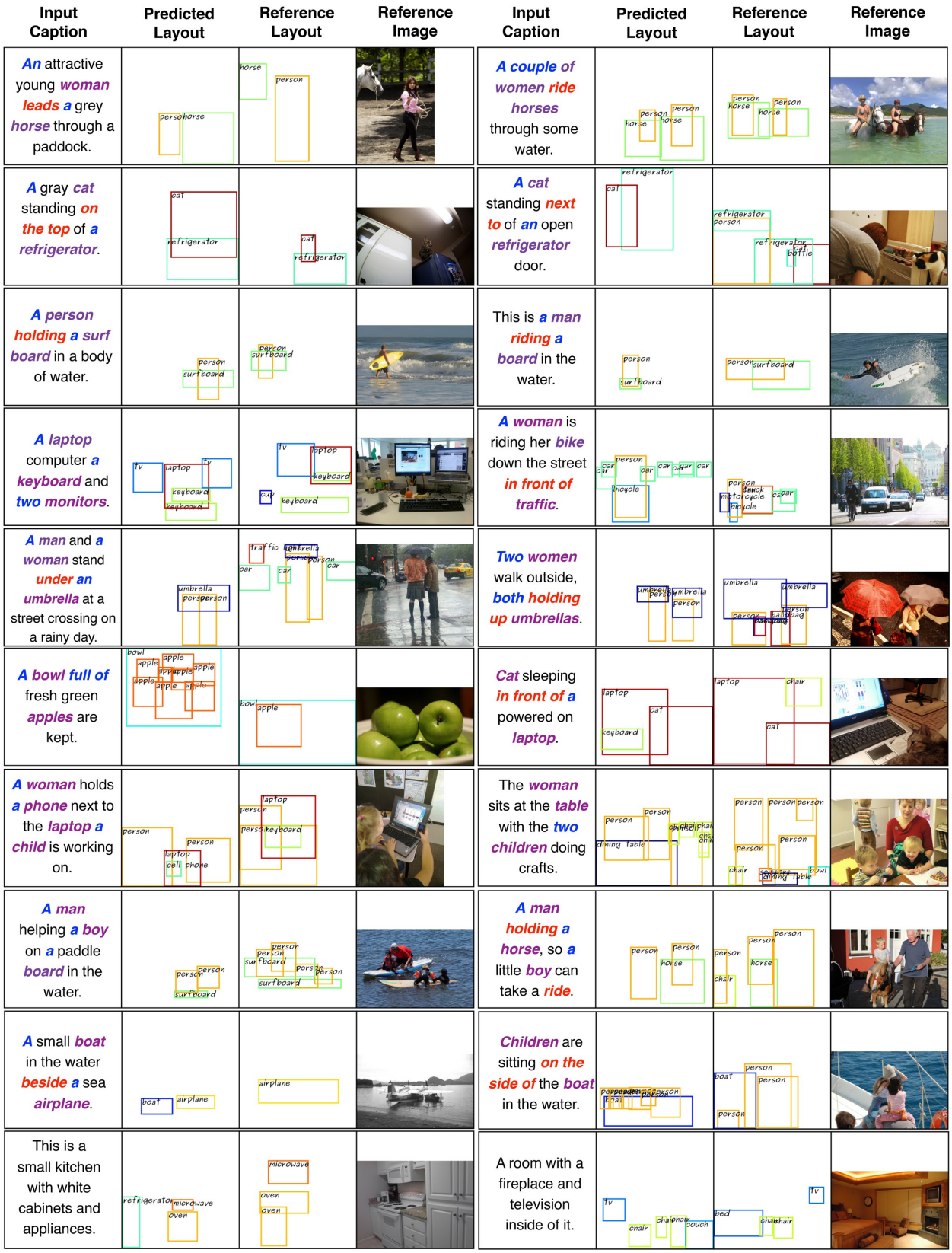

Figure 3. More qualitative examples of the layout generation experiment (best viewed in color). The presences (purple), counts (blue), and spatial relations (red) of the objects are highlighted in the captions. The last row shows the cases when the layouts are underspecified in the input captions.

Figure 4. Example synthetic images and the source images from which the patch segments are retrieved for compositing. For each synthetic image, we show three source images for clarity.

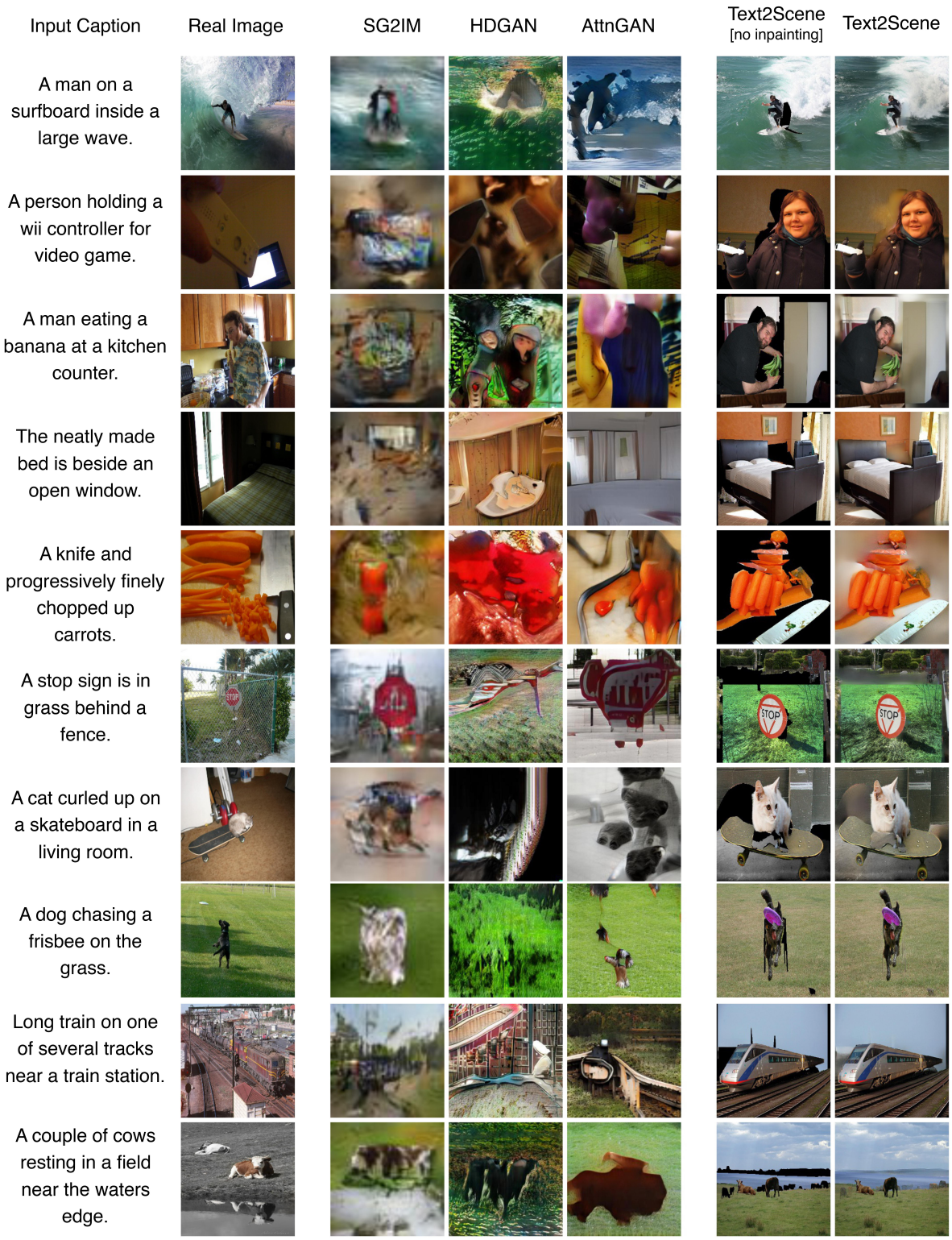

Figure 5. More qualitative examples of the synthetic image generation experiment.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Jasper Uijlings Holger Caesar and Vittorio Ferrari. Cocostuff: Thing and stuff classes in context. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[5] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *European Conference on Computer Vision (ECCV)*, 2014.

[6] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, 2015.

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[8] Xiaojuan Qi, Qifeng Chen, Jiaya Jia, and Vladlen Koltun. Semi-parametric image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[9] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[10] Zizhao Zhang, Yuanpu Xie, and Lin Yang. Photographic text-to-image synthesis with a hierarchically-nested adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[11] C. Lawrence Zitnick, Devi Parikh, and Lucy Vanderwende. Learning the visual interpretation of sentences. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[12] Z. Zuo, B. Shuai, G. Wang, X. Liu, X. Wang, B. Wang, and Y. Chen. Convolutional recurrent neural networks: Learning spatial dependencies for image representation. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 18–26, 2015.