Image Generation from Scene Graphs

Justin Johnson[1,2*] Agrim Gupta[1] Li Fei-Fei[1,2]
[1]Stanford University [2]Google Cloud AI

Abstract

To truly understand the visual world our models should be able not only to recognize images but also generate them. To this end, there has been exciting recent progress on generating images from natural language descriptions. These methods give stunning results on limited domains such as descriptions of birds or flowers, but struggle to faithfully reproduce complex sentences with many objects and relationships. To overcome this limitation we propose a method for generating images from scene graphs, enabling explicitly reasoning about objects and their relationships. Our model uses graph convolution to process input graphs, computes a scene layout by predicting bounding boxes and segmentation masks for objects, and converts the layout to an image with a cascaded refinement network. The network is trained adversarially against a pair of discriminators to ensure realistic outputs. We validate our approach on Visual Genome and COCO-Stuff, where qualitative results, ablations, and user studies demonstrate our method's ability to generate complex images with multiple objects.

1. Introduction

What I cannot create, I do not understand
– Richard Feynman

The act of *creation* requires a deep understanding of the thing being created: chefs, novelists, and filmmakers must understand food, writing, and film at a much deeper level than diners, readers, or moviegoers. If our computer vision systems are to truly understand the visual world, they must be able not only *recognize* images but also to *generate* them.

Aside from imparting deep visual understanding, methods for generating realistic images can also be practically useful. In the near term, automatic image generation can aid the work of artists or graphic designers. One day, we might replace image and video search engines with algorithms that generate customized images and videos in response to the individual tastes of each user.

As a step toward these goals, there has been exciting recent progress on *text to image synthesis* [41, 42, 43, 59] by combining recurrent neural networks and Generative Adversarial Networks [12] to generate images from natural language descriptions.

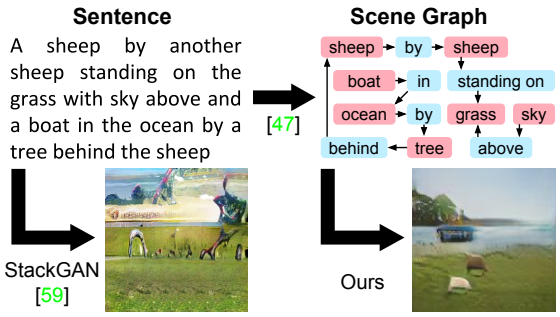

Figure 1. State-of-the-art methods for generating images from sentences, such as StackGAN [59], struggle to faithfully depict complex sentences with many objects. We overcome this limitation by generating images from *scene graphs*, allowing our method to reason explicitly about objects and their relationships.

These methods can give stunning results on limited domains, such as fine-grained descriptions of birds or flowers. However as shown in Figure 1, leading methods for generating images from sentences struggle with complex sentences containing many objects.

A sentence is a linear structure, with one word following another; however as shown in Figure 1, the information conveyed by a complex sentence can often be more explicitly represented as a *scene graph* of objects and their relationships. Scene graphs are a powerful structured representation for both images and language; they have been used for semantic image retrieval [22] and for evaluating [1] and improving [31] image captioning; methods have also been developed for converting sentences to scene graphs [47] and for predicting scene graphs from images [32, 36, 57, 58].

In this paper we aim to generate complex images with many objects and relationships by conditioning our generation on scene graphs, allowing our model to reason explicitly about objects and their relationships.

With this new task comes new challenges. We must develop a method for processing scene graph inputs; for this we use a *graph convolution network* which passes information along graph edges. After processing the graph, we must

*Work done during an internship at Google Cloud AI.

bridge the gap between the symbolic graph-structured input and the two-dimensional image output; to this end we construct a *scene layout* by predicting bounding boxes and segmentation masks for all objects in the graph. Having predicted a layout, we must generate an image which respects it; for this we use a *cascaded refinement network* (CRN) [6] which processes the layout at increasing spatial scales. Finally, we must ensure that our generated images are realistic and contain recognizable objects; we therefore train adversarially against a pair of *discriminator* networks operating on image patches and generated objects. All components of the model are learned jointly in an end-to-end manner.

We experiment on two datasets: Visual Genome [26], which provides human annotated scene graphs, and COCO-Stuff [3] where we construct synthetic scene graphs from ground-truth object positions. On both datasets we show qualitative results demonstrating our method's ability to generate complex images which respect the objects and relationships of the input scene graph, and perform comprehensive ablations to validate each component of our model.

Automated evaluation of generative images models is a challenging problem unto itself [52], so we also evaluate our results with two user studies on Amazon Mechanical Turk. Compared to StackGAN [59], a leading system for text to image synthesis, users find that our results better match COCO captions in 68% of trials, and contain 59% more recognizable objects.

2. Related Work

Generative Image Models fall into three recent categories: Generative Adversarial Networks (GANs) [12, 40] jointly learn a *generator* for synthesizing images and a *discriminator* classifying images as real or fake; Variational Autoencoders [24] use variational inference to jointly learn an *encoder* and *decoder* mapping between images and latent codes; autoregressive approaches [38, 53] model likelihoods by conditioning each pixel on all previous pixels.

Conditional Image Synthesis conditions generation on additional input. GANs can be conditioned on category labels by providing labels as an additional input to both generator and discriminator [10, 35] or by forcing the discriminator to predict the label [37]; we take the latter approach.

Reed *et al*. [42] generate images from text using a GAN; Zhang *et al*. [59] extend this approach to higher resolutions using multistage generation. Related to our approach, Reed *et al*. generate images conditioned on sentences and keypoints using both GANs [41] and multiscale autoregressive models [43]; in addition to generating images they also predict locations of unobserved keypoints using a separate generator and discriminator operating on keypoint locations.

Chen and Koltun [6] generate high-resolution images of street scenes from ground-truth semantic segmentation using a cascaded refinement network (CRN) trained with a perceptual feature reconstruction loss [9, 21]; we use their CRN architecture to generate images from scene layouts.

Related to our layout prediction, Chang *et al*. have investigated text to 3D scene generation [4, 5]; other approaches to image synthesis include stochastic grammars [20], probabalistic programming [27], inverse graphics [28], neural de-rendering [55], and generative ConvNets [56].

Scene Graphs represent scenes as directed graphs, where nodes are objects and edges give relationships between objects. Scene graphs have been used for image retrieval [22] and to evaluate image captioning [1]; some work converts sentences to scene graphs [47] or predicts grounded scene graphs for images [32, 36, 57, 58]. Most work on scene graphs uses the Visual Genome dataset [26], which provides human-annotated scene graphs.

Deep Learning on Graphs. Some methods learn embeddings for graph nodes given a single large graph [39, 51, 14] similar to word2vec [34] which learns embeddings for words given a text corpus. These differ from our approach, since we must process a new graph on each forward pass.

More closely related to our work are Graph Neural Networks (GNNs) [11, 13, 46] which generalize recursive neural networks [8, 49, 48] to operate on arbitrary graphs. GNNs and related models have been applied to molecular property prediction [7], program verification [29], modeling human motion [19], and premise selection for theorem proving [54]. Some methods operate on graphs in the spectral domain [2, 15, 25] though we do not take this approach.

3. Method

Our goal is to develop a model which takes as input a *scene graph* describing objects and their relationships, and which generates a realistic image corresponding to the graph. The primary challenges are threefold: first, we must develop a method for processing the graph-structured input; second, we must ensure that the generated images respect the objects and relationships specified by the graph; third, we must ensure that the synthesized images are realistic.

We convert scene graphs to images with an *image generation network* f, shown in Figure 2, which inputs a scene graph G and noise z and outputs an image $\hat{I} = f(G, z)$.

The scene graph G is processed by a *graph convolution network* which gives embedding vectors for each object; as shown in Figures 2 and 3, each layer of graph convolution mixes information along edges of the graph.

We respect the objects and relationships from G by using the object embedding vectors from the graph convolution network to predict bounding boxes and segmentation masks for each object; these are combined to form a *scene layout*, shown in the center of Figure 2, which acts as an intermediate between the graph and the image domains.

The output image $\hat{I}$ is generated from the layout using a *cascaded refinement network* (CRN) [6], shown in the right

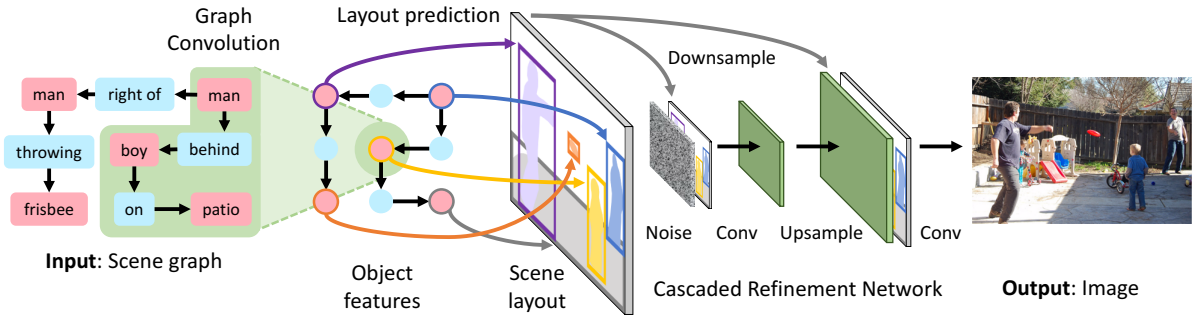

Figure 2. Overview of our image generation network f for generating images from scene graphs. The input to the model is a *scene graph* specifying objects and relationships; it is processed with a *graph convolution network* (Figure 3) which passes information along edges to compute embedding vectors for all objects. These vectors are used to predict bounding boxes and segmentation masks for objects, which are combined to form a *scene layout* (Figure 4). The layout is converted to an image using a *cascaded refinement network* (CRN) [6]. The model is trained adversarially against a pair of *discriminator networks*. During training the model observes ground-truth object bounding boxes and (optionally) segmentation masks, but these are predicted by the model at test-time.

half of Figure 2; each of its modules processes the layout at increasing spatial scales, eventually generating the image $\hat{I}$.

We generate realistic images by training f adversarially against a pair of *discriminator* networks D_{img} and D_{obj} which encourage the image $\hat{I}$ to both appear realistic and to contain realistic, recognizable objects.

Each of these components is described in more detail below; the supplementary material describes the exact architecures used in our experiments.

Scene Graphs. The input to our model is a *scene graph* [22] describing objects and relationships between objects. Given a set of object categories $\mathcal{C}$ and a set of relationship categories $\mathcal{R}$, a scene graph is a tuple (O, E) where $O = \{o_1, \ldots, o_n\}$ is a set of objects with each $o_i \in \mathcal{C}$, and $E \subseteq O \times \mathcal{R} \times O$ is a set of directed edges of the form (o_i, r, o_j) where $o_i, o_j \in O$ and $r \in \mathcal{R}$.

As a first stage of processing, we use a learned embedding layer to convert each node and edge of the graph from a categorical label to a dense vector, analogous to the embedding layer typically used in neural language models.

Graph Convolution Network. In order to process scene graphs in an end-to-end manner, we need a neural network module which can operate natively on graphs. To this end we use a *graph convolution network* composed of several *graph convolution layers*.

A traditional 2D convolution layer takes as input a spatial grid of feature vectors and produces as output a new spatial grid of vectors, where each output vector is a function of a local neighborhood of its corresponding input vector; in this way a convolution aggregates information across local neighborhoods of the input. A single convolution layer can operate on inputs of arbitrary shape through the use of *weight sharing* across all neighborhoods in the input.

Our graph convolution layer performs a similar function: given an input graph with vectors of dimension D_{in} at each node and edge, it computes new vectors of dimension D_{out} for each node and edge. Output vectors are a function of a neighborhood of their corresponding inputs, so that each graph convolution layer propagates information along edges of the graph. A graph convolution layer applies the same function to all edges of the graph, allowing a single layer to operate on graphs of arbitrary shape.

Concretely, given input vectors $v_i, v_r \in \mathbb{R}^{D_{in}}$ for all objects $o_i \in O$ and edges $(o_i, r, o_j) \in E$, we compute output vectors for $v_i', v_r' \in \mathbb{R}^{D_{out}}$ for all nodes and edges using three functions g_s, g_p, and g_o, which take as input the triple of vectors (v_i, v_r, v_j) for an edge and output new vectors for the subject o_i, predicate r, and object o_j respectively.

To compute the output vectors v_r' for edges we simply set $v_r' = g_p(v_i, v_r, v_j)$. Updating object vectors is more complex, since an object may participate in many relationships; as such the output vector v_i' for an object o_i should depend on all vectors v_j for objects to which o_i is connected via graph edges, as well as the vectors v_r for those edges. To this end, for each edge starting at o_i we use g_s to compute a *candidate vector*, collecting all such candidates in the set V_i^s; we similarly use g_o to compute a set of candidate vectors V_i^o for all edges terminating at o_i. Concretely,

$$V_i^s = \{g_s(v_i, v_r, v_j) : (o_i, r, o_j) \in E\} \quad (1)$$

$$V_i^o = \{g_o(v_j, v_r, v_i) : (o_j, r, o_i) \in E\}. \quad (2)$$

The output vector for v_i' for object o_i is then computed as $v_i' = h(V_i^s \cup V_i^o)$ where h is a symmetric function which pools an input set of vectors to a single output vector. An example computational graph for a single graph convolution layer is shown in Figure 3.

In our implementation, the functions g_s, g_p, and g_o are implemented using a single network which concatenates its three input vectors, feeds them to a multilayer perceptron (MLP), and computes three output vectors using fully-connected output heads. The pooling function h averages its input vectors and feeds the result to a MLP.

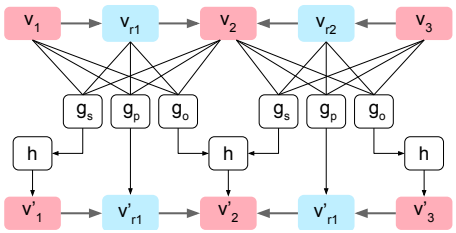

Figure 3. Computational graph illustrating a single graph convolution layer. The graph consists of three objects o_1, o_2, and o_3 and two edges (o_1, r_1, o_2) and (o_3, r_2, o_2). Along each edge, the three input vectors are passed to functions g_s, g_p, and g_o; g_p directly computes the output vector for the edge, while g_s and g_o compute *candidate vectors* which are fed to a symmetric pooling function h to compute output vectors for objects.

Scene Layout. Processing the input scene graph with a series of graph convolution layers gives an embedding vector for each object which aggregates information across all objects and relationships in the graph.

In order to generate an image, we must move from the graph domain to the image domain. To this end, we use the object embedding vectors to compute a *scene layout* which gives the coarse 2D structure of the image to generate; we compute the scene layout by predicting a segmentation mask and bounding box for each object using an *object layout network*, shown in Figure 4.

The object layout network receives an embedding vector v_i of shape D for object o_i and passes it to a *mask regression network* to predict a soft binary mask $\hat{m}_i$ of shape $M \times M$ and a *box regression network* to predict a bounding box $\hat{b}_i = (x_0, y_0, x_1, y_1)$. The mask regression network consists of several transpose convolutions terminating in a sigmoid nonlinearity so that elements of the mask lies in the range $(0, 1)$; the box regression network is a MLP.

We multiply the embedding vector v_i elementwise with the mask $\hat{m}_i$ to give a masked embedding of shape $D \times M \times M$ which is then warped to the position of the bounding box using bilinear interpolation [18] to give an object layout. The scene layout is then the sum of all object layouts.

During training we use ground-truth bounding boxes b_i to compute the scene layout; at test-time we instead use predicted bounding boxes $\hat{b}_i$.

Cascaded Refinement Network. Given the scene layout, we must synthesize an image that respects the object positions given in the layout. For this task we use a Cascaded Refinement Network [6] (CRN). A CRN consists of a series of convolutional refinement modules, with spatial resolution doubling between modules; this allows generation to proceed in a coarse-to-fine manner.

Each module receives as input both the scene layout (downsampled to the input resolution of the module) and the output from the previous module. These inputs are concatenated channelwise and passed to a pair of 3×3 convolution layers; the output is then upsampled using nearest-neighbor interpolation before being passed to the next module.

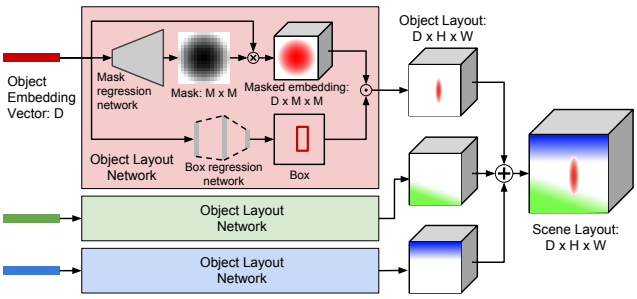

Figure 4. We move from the graph domain to the image domain by computing a *scene layout*. The embedding vector for each object is passed to an object layout network which predicts a layout for the object; summing all object layouts gives the scene layout. Internally the object layout network predicts a soft binary segmentation mask and a bounding box for the object; these are combined with the embedding vector using bilinear interpolation to produce the object layout.

The first module takes Gaussian noise $z \sim p_z$ as input, and the output from the last module is passed to two final convolution layers to produce the output image.

Discriminators. We generate realistic output images by training the image generation network f adversarially against a pair of *discriminator* networks D_{img} and D_{obj}.

A discriminator D attempts to classify its input x as real or fake by maximizing the objective [12]

$$\mathcal{L}_{GAN} = \mathop{\mathbb{E}}_{x \sim p_{\text{real}}} \log D(x) + \mathop{\mathbb{E}}_{x \sim p_{\text{fake}}} \log(1 - D(x)) \quad (3)$$

where $x \sim p_{\text{fake}}$ are outputs from the generation network f. At the same time, f attempts to generate outputs which will fool the discriminator by minimizing $\mathcal{L}_{GAN}$.[1]

The patch-based *image discriminator* D_{img} ensures that the overall appearance of generated images is realistic; it classifies a regularly spaced, overlapping set of image patches as real or fake, and is implemented as a fully convolutional network, similar to the discriminator used in [17].

The *object discriminator* D_{obj} ensures that each object in the image appears realistic; its input are the pixels of an object, cropped and rescaled to a fixed size using bilinear interpolation [18]. In addition to classifying each object as real or fake, D_{obj} also ensures that each object is recognizable using an *auxiliary classifier* [37] which predicts the object's category; both D_{obj} and f attempt to maximize the probability that D_{obj} correctly classifies objects.

Training. We jointly train the generation network f and the discriminators D_{obj} and D_{img}. The generation network is trained to minimize the weighted sum of six losses:

[1] In practice, to avoid vanishing gradients f typically maximizes the surrogate objective $\mathbb{E}_{x \sim p_{\text{fake}}} \log D(x)$ instead of minimizing $\mathcal{L}_{GAN}$ [12].

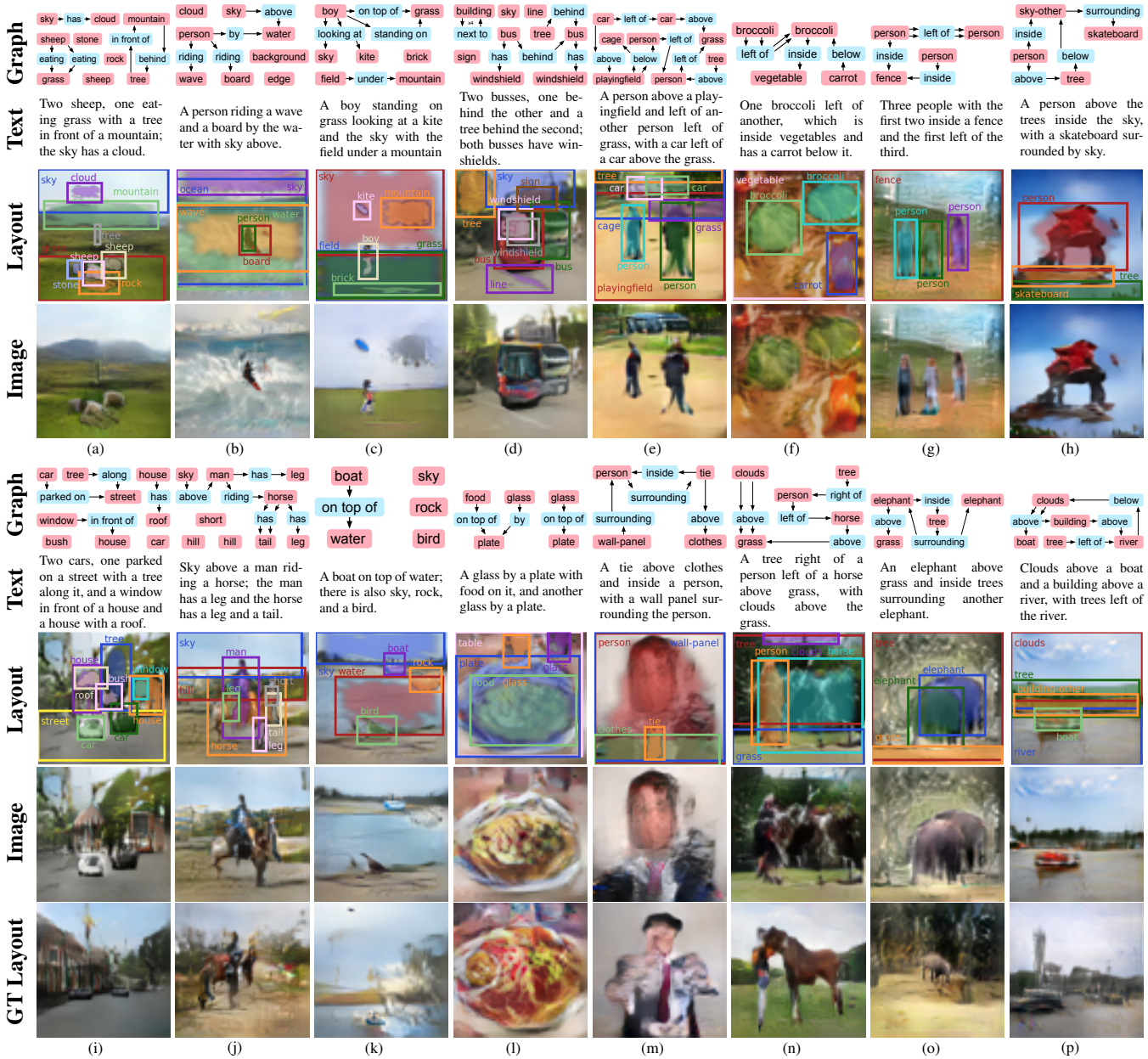

Figure 5. Examples of 64×64 generated images using graphs from the test sets of Visual Genome (left four columns) and COCO (right four columns). For each example we show the input scene graph and a manual translation of the scene graph into text; our model processes the scene graph and predicts a layout consisting of bounding boxes and segmentation masks for all objects; this layout is then used to generate the image. We also show some results for our model using ground-truth rather than predicted scene layouts. Some scene graphs have duplicate relationships, shown as double arrows. For clarity, we omit masks for some stuff categories such as sky, street, and water.

- *Box loss* $\mathcal{L}_{box} = \sum_{i=1}^{n} \|b_i - \hat{b}_i\|_1$ penalizing the L_1 difference between ground-truth and predicted boxes
- *Mask loss* $\mathcal{L}_{mask}$ penalizing differences between ground-truth and predicted masks with pixelwise cross-entropy; not used for models trained on Visual Genome
- *Pixel loss* $\mathcal{L}_{pix} = \|I - \hat{I}\|_1$ penalizing the L_1 difference between ground-truth generated images
- *Image adversarial loss* $\mathcal{L}_{GAN}^{img}$ from D_{img} encouraging generated image patches to appear realistic
- *Object adversarial loss* $\mathcal{L}_{GAN}^{obj}$ from the D_{obj} encouraging each generated object to look realistic
- *Auxiliarly classifier loss* $\mathcal{L}_{AC}^{obj}$ from D_{obj}, ensuring that each generated object can be classified by D_{obj}

Implementation Details. We augment all scene graphs with a special *image* object, and add special *in image* relationships connecting each true object with the *image* object; this ensures that all scene graphs are connected.

Figure 6. Images generated by our method trained on Visual Genome. In each row we start from a simple scene graph on the left and progressively add more objects and relationships moving to the right. Images respect relationships like *car below kite* and *boat on grass*.

We train all models using Adam [23] with learning rate 10^{-4} and batch size 32 for 1 million iterations; training takes about 3 days on a single Tesla P100. For each mini-batch we first update f, then update D_{img} and D_{obj}.

We use ReLU for graph convolution; the CRN and discriminators use discriminators use LeakyReLU [33] and batch normalization [16]. Full details about our architecture can be found in the supplementary material, and code will be made publicly available.

4. Experiments

We train our model to generate 64×64 images on the Visual Genome [26] and COCO-Stuff [3] datasets. In our experiments we aim to show that our method generates images of complex scenes which respect the objects and relationships of the input scene graph.

4.1. Datasets

COCO. We perform experiments on the 2017 COCO-Stuff dataset [3], which augments a subset of the COCO dataset [30] with additional stuff categories. The dataset annotates 40K train and 5K val images with bounding boxes and segmentation masks for 80 *thing* categories (people, cars, *etc*.) and 91 *stuff* categories (sky, grass, *etc*.).

We use these annotations to construct synthetic scene graphs based on the 2D image coordinates of the objects, using six mutually exclusive geometric relationships: *left of*, *right of*, *above*, *below*, *inside*, and *surrounding*.

We ignore objects covering less than 2% of the image, and use images with 3 to 8 objects; we divide the COCO-Stuff 2017 val set into our own val and test sets, leaving us with 24,972 train, 1024 val, and 2048 test images.

Visual Genome. We experiment on Visual Genome [26] version 1.4 (VG) which comprises 108,077 images annotated with scene graphs. We divide the data into 80% train, 10% val, and 10% test; we use object and relationship categories occurring at least 2000 and 500 times respectively in the train set, leaving 178 object and 45 relationship types.

We ignore small objects, and use images with between 3 and 30 objects and at least one relationship; this leaves us with 62,565 train, 5,506 val, and 5,088 test images with an average of ten objects and five relationships per image.

Visual Genome does not provide segmentation masks, so we omit the mask prediction loss for models trained on VG.

4.2. Qualitative Results

Figure 5 shows example scene graphs from the Visual Genome and COCO test sets and generated images using our method, as well as predicted object bounding boxes and segmentation masks.

From these examples it is clear that our method can generate scenes with multiple objects, and even multiple instances of the same object type: for example Figure 5 (a) shows two sheep, (d) shows two busses, (g) contains three people, and (i) shows two cars.

These examples also show that our method generates images which respect the relationships of the input graph; for example in (i) we see one broccoli *left of* a second broccoli, with a carrot *below* the second broccoli; in (j) the man is *riding* the horse, and both the man and the horse have *legs* which have been properly positioned.

Figure 5 also shows examples of images generated by our method using ground-truth rather than predicted object layouts. In some cases we see that our predicted layouts can

Method	Inception	
	COCO	VG
Real Images (64×64)	16.3 ± 0.4	13.9 ± 0.5
Ours (No gconv)	4.6 ± 0.1	4.2 ± 0.1
Ours (No relationships)	3.7 ± 0.1	4.9 ± 0.1
Ours (No discriminators)	4.8 ± 0.1	3.6 ± 0.1
Ours (No D_{obj})	5.6 ± 0.1	5.0 ± 0.2
Ours (No D_{img})	5.6 ± 0.1	$\mathbf{5.7 \pm 0.3}$
Ours (Full model)	$\mathbf{6.7 \pm 0.1}$	5.5 ± 0.1
Ours (GT Layout, no gconv)	7.0 ± 0.2	6.0 ± 0.2
Ours (GT Layout)	$\mathbf{7.3 \pm 0.1}$	$\mathbf{6.3 \pm 0.2}$
StackGAN [59] (64×64)	$\mathbf{8.4 \pm 0.2}$	-

Table 1. Ablation study using Inception scores. On each dataset we randomly split our test-set samples into 5 groups and report mean and standard deviation across splits. On COCO we generate five samples for each test-set image by constructing different synthetic scene graphs. For StackGAN we generate one image for each of the COCO test-set captions, and downsample their 256×256 output to 64×64 for fair comparison with our method.

vary significantly from the ground-truth objects layout. For example in (k) the graph does not specify the position of the bird and our method renders it standing on the ground, but in the ground-truth layout the bird is flying in the sky. Our model is sometimes bottlenecked by layout prediction, such as (n) where using the ground-truth rather than predicted layout significantly improves the image quality.

In Figure 6 we demonstrate our model's ability to generate complex images by starting with simple graphs on the left and progressively building up to more complex graphs. From this example we can see that object positions are influenced by the relationships in the graph: in the top sequence adding the relationship *car below kite* causes the car to shift to the right and the kite to shift to the left so that the relationship is respected. In the bottom sequence, adding the relationship *boat on grass* causes the boat's position to shift.

4.3. Ablation Study

We demonstrate the necessity of all components of our model by comparing the image quality of several ablated versions of our model, shown in Table 1; see supplementary material for example images from ablated models.

We measure image quality using *Inception score*[2] [45] which uses an ImageNet classification model [44, 50] to encourage recognizable objects within images and diversity across images. We test several ablations of our model:

No gconv omits graph convolution, so boxes and masks are predicted from initial object embedding vectors. It cannot reason jointly about the presence of different objects, and can only predict one box and mask per category.

No relationships uses graph convolution layers but ignores all relationships from the input scene graph except for trivial *in image* relationships; graph convolution allows this model to jointly about objects. Its poor performance demonstrates the utility of the scene graph relationships.

No discriminators omits both D_{img} and D_{obj}, relying on the pixel regression loss $\mathcal{L}_{pix}$ to guide the generation network. It tends to produce overly smoothed images.

No $\mathbf{D_{obj}}$ and **No $\mathbf{D_{img}}$** omit one of the discriminators. On both datasets, using any discriminator leads to significant improvements over models trained with $\mathcal{L}_{pix}$ alone. On COCO the two discriminators are complimentary, and combining them in our full model leads to large improvements. On VG, omitting D_{img} does not degrade performance.

In addition to ablations, we also compare with two **GT Layout** versions of our model which omit the $\mathcal{L}_{box}$ and $\mathcal{L}_{mask}$ losses, and use ground-truth bounding boxes during both training and testing; on COCO they also use ground-truth segmentation masks, similar to Chen and Koltun [6]. These methods give an upper bound to our model's performance in the case of perfect layout prediction.

Omitting graph convolution degrades performance even when using ground-truth layouts, suggesting that scene graph relationships and graph convolution have benefits beyond simply predicting object positions.

[2]Defined as $\exp(\mathbb{E}_{\hat{I}} KL(p(y|\hat{I})\|p(y)))$ where the expectation is taken over generated images $\hat{I}$ and $p(y|\hat{I})$ is the predicted label distribution.

	R@0.3		R@0.5		σ_x		σ_{area}	
	COCO	VG	COCO	VG	COCO	VG	COCO	VG
Ours (No gconv)	46.9	20.2	20.8	6.4	0	0	0	0
Ours (No rel.)	21.8	16.5	7.6	6.9	**0.1**	**0.1**	**0.2**	**0.1**
Ours (Full)	**52.4**	**21.9**	**32.2**	**10.6**	**0.1**	**0.1**	**0.2**	**0.1**

Table 2. Statistics of predicted bounding boxes. R@t is object recall with an IoU threshold of t, and measures agreement with ground-truth boxes. σ_x and σ_{area} measure box variety by computing the standard deviation of box x-positions and areas within each object category and then averaging across categories.

4.4. Object Localization

In addition to looking at images, we can also inspect the bounding boxes predicted by our model. One measure of box quality is high agreement between predicted and ground-truth boxes; in Table 2 we show the object recall of our model at two intersection-over-union thresholds.

Another measure for boxes is *variety*: predicted boxes for objects should vary in response to the other objects and relationships in the graph. Table 2 shows the mean per-category standard deviations of box position and area.

Without graph convolution, our model can only learn to predict a single bounding box per object category. This model achieves nontrivial object recall, but has no variety in its predicted boxes, as $\sigma_x = \sigma_{area} = 0$.

Using graph convolution without relationships, our model can jointly reason about objects when predicting bounding boxes; this leads to improved variety in its predictions. Without relationships, this model's predicted boxes have less agreement with ground-truth box positions.

Which image matches the caption better?

User choice	332 / 1024 (32.4%)	**692** / 1024 **(67.6%)**

Figure 7. We performed a user study to compare the semantic interpretability of our method against StackGAN [59]. **Top:** We use StackGAN to generate an image from a COCO caption, and use our method to generate an image from a scene graph constructed from the COCO objects corresponding to the caption. We show users the caption and both images, and ask which better matches the caption. **Bottom:** Across 1024 val image pairs, users prefer the results from our method by a large margin.

Our full model with graph convolution and relationships achieves both variety and high agreement with ground-truth boxes, indicating that it can use the relationships of the graph to help localize objects with greater fidelity.

4.5. User Studies

Automatic metrics such as Inception scores and box statistics give a coarse measure of image quality; the true measure of success is human judgement of the generated images. For this reason we performed two user studies on Mechanical Turk to evaluate our results.

We are unaware of any previous end-to-end methods for generating images from scene graphs, so we compare our method with StackGAN [59], a state-of-the art method for generating images from sentence descriptions.

Despite the different input modalities between our method and StackGAN, we can compare the two on COCO, which in addition to object annotations also provides captions for each image. We use our method to generate images from synthetic scene graphs built from COCO object annotations, and StackGAN[3] to generate images from COCO captions for the same images. Though the methods receive different inputs, they should generate similar images due to the correspondence between COCO captions and objects.

For user studies we downsample StackGAN images to 64×64 to compensate for differing resolutions; we repeat all trials with three workers and randomize order in all trials.

Caption Matching. We measure semantic interpretability by showing users a COCO caption, an image generated by StackGAN from that caption, and an image generated by our method from a scene graph built from the COCO objects corresponding to the caption. We ask users to select the image that better matches the caption. An example image pair and results are shown in Figure 7.

[3]We use the pretrained COCO model provided by the authors at https://github.com/hanzhanggit/StackGAN-Pytorch

Which objects are present? motorcycle, person, clouds

Thing recall	470 / 1650 (28.5%)	**772** / 1650 **(46.8%)**
Stuff Recall	1285 / 3556 (36.1%)	**2071** / 3556 **(58.2%)**

Figure 8. We performed a user study to measure the number of recognizable objects in images from our method and from StackGAN [59]. **Top:** We use StackGAN to generate an image from a COCO caption, and use our method to generate an image from a scene graph built from the COCO objects corresponding to the caption. For each image, we ask users which COCO objects they can see in the image. **Bottom:** Across 1024 val image pairs, we measure the fraction of *things* and *stuff* that users can recognize in images from each method. Our method produces more objects.

This experiment is biased toward StackGAN, since the caption may contain information not captured by the scene graph. Even so, a majority of workers preferred the result from our method in 67.6% of image pairs, demonstrating that compared to StackGAN our method more frequently generates complex, semantically meaningful images.

Object Recall. This experiment measures the number of recognizable objects in each method's images. In each trial we show an image from one method and a list of COCO objects and ask users to identify which objects appear in the image. An example and results are snown in Figure 8.

We compute the fraction of objects that a majority of users believed were present, dividing the results into *things* and *stuff*. Both methods achieve higher recall for stuff than things, and our method achieves significantly higher object recall, with 65% and 61% relative improvements for thing and stuff recall respectively.

This experiment is biased toward our method since the scene graph may contain objects not mentioned in the caption, but it demonstrates that compared to StackGAN, our method produces images with more recognizable objects.

5. Conclusion

In this paper we have developed an end-to-end method for generating images from scene graphs. Compared to leading methods which generate images from text descriptions, generating images from structured scene graphs rather than unstructured text allows our method to reason explicitly about objects and relationships, and generate complex images with many recognizable objects.

Acknowledgments We thank Shyamal Buch, Christopher Choy, De-An Huang, and Ranjay Krishna for helpful comments and suggestions.

References

[1] P. Anderson, B. Fernando, M. Johnson, and S. Gould. Spice: Semantic propositional image caption evaluation. In *ECCV*, 2016. 1, 2

[2] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014. 2

[3] H. Caesar, J. Uijlings, and V. Ferrari. Coco-stuff: Thing and stuff classes in context. *arXiv preprint arXiv:1612.03716*, 2016. 2, 6

[4] A. Chang, W. Monroe, M. Savva, C. Potts, and C. D. Manning. Text to 3d scene generation with rich lexical grounding. In *ACL*, 2015. 2

[5] A. X. Chang, M. Savva, and C. D. Manning. Learning spatial knowledge for text to 3d scene generation. In *EMNLP*, 2014. 2

[6] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. In *ICCV*, 2017. 2, 3, 4, 7

[7] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015. 2

[8] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 1998. 2

[9] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 2

[10] J. Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014. 2

[11] C. Goller and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, 1996. 2

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 1, 2, 4

[13] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, 2005. 2

[14] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, 2016. 2

[15] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. 2

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 6

[17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 4

[18] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015. 4

[19] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *CVPR*, 2016. 2

[20] C. Jiang, Y. Zhu, S. Qi, S. Huang, J. Lin, X. Guo, L.-F. Yu, D. Terzopoulos, and S.-C. Zhu. Configurable, photorealistic image rendering and ground truth synthesis by sampling stochastic grammars representing indoor scenes. *arXiv preprint arXiv:1704.00112*, 2017. 2

[21] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 2

[22] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei. Image retrieval using scene graphs. In *CVPR*, 2015. 1, 2, 3

[23] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[24] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014. 2

[25] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2

[26] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *IJCV*, 2017. 2, 6

[27] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. In *CVPR*, 2015. 2

[28] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015. 2

[29] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2015. 2

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 6

[31] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy. Improved image captioning via policy gradient optimization of spider. In *ICCV*, 2017. 1

[32] C. Lu, R. Krishna, M. Bernstein, and L. Fei-Fei. Visual relationship detection with language priors. In *ECCV*, 2016. 1, 2

[33] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013. 6

[34] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013. 2

[35] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 2

[36] A. Newell and J. Deng. Pixels to graphs by associative embedding. In *NIPS*, 2017. 1, 2

[37] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. In *ICML*, 2017. 2, 4

[38] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016. 2

[39] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, 2014. 2

[40] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2

[41] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In *NIPS*, 2016. 1, 2

[42] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text-to-image synthesis. In *ICML*, 2016. 1, 2

[43] S. E. Reed, A. van den Oord, N. Kalchbrenner, S. Gómez, Z. Wang, D. Belov, and N. de Freitas. Parallel multiscale autoregressive density estimation. In *ICML*, 2017. 1, 2

[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015. 7

[45] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016. 7

[46] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009. 2

[47] S. Schuster, R. Krishna, A. Chang, L. Fei-Fei, and C. D. Manning. Generating semantically precise scene graphs from textual descriptions for improved image retrieval. In *EMNLP Vision and Language Workshop*, 2015. 1, 2

[48] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011. 2

[49] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 1997. 2

[50] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 7

[51] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, 2015. 2

[52] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. In *ICLR*, 2016. 2

[53] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with PixelCNN decoders. In *NIPS*, 2016. 2

[54] M. Wang, Y. Tang, J. Wang, and J. Deng. Premise selection for theorem proving by deep graph embedding. In *NIPS*, 2017. 2

[55] J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene derendering. In *CVPR*, 2017. 2

[56] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu. A theory of generative convnet. In *ICML*, 2016. 2

[57] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei. Scene graph generation by iterative message passing. In *CVPR*, 2017. 1, 2

[58] M. Y. Yang, W. Liao, H. Ackermann, and B. Rosenhahn. On support relations and semantic scene graphs. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2017. 1, 2

[59] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017. 1, 2, 7, 8

Index	Inputs	Operation	Output shape
(1)	-	Subject vector v_s	D_{in}
(2)	-	Relationship vector v_r	D_{in}
(3)	-	Object vector v_o	D_{in}
(4)	(1), (2), (3)	Concatenate	$3D_{in}$
(5)	(4)	Linear($3D_{in} \to H$)	H
(6)	(5)	ReLU	H
(7)	(6)	Linear($H \to 2H + D_{out}$)	$2H + D_{out}$
(8)	(7)	ReLU	$2H + D_{out}$
(9)	(8)	Split into 3 chunks	H, D_{out}, H
(10)	(9)	1st chunk $\tilde{v}_s$	H
(11)	(9)	2nd chunk v'_r	D_{out}
(12)	(9)	3rd chunk $\tilde{v}_o$	H

Table 3. Network architecture for the first network g used in graph convolution; this single network implements the three functions g_s, g_p, and g_o from the main text.

Supplementary Material

A. Network Architecture

Here we describe the exact network architectures for all components of our model.

A.1. Graph Convolution Layer

As described in Section 3 of the main paper, we process the input scene graph with a *graph convolution network* composed of several *graph convolution layers*.

A graph convolution layer accepts as input a vector of dimension D_{in} for each node and edge in the graph, and computes new vectors of dimension D_{out} for each node and edge. A single graph convolution layer can be applied to graphs of any size of shape due to *weight sharing*. A single graph convolution layer proceeds in two stages.

First, along relationship of the scene graph we apply three functions g_s, g_p, and g_o; these functions take as input the vectors v_s, v_r, and v_o for the starting node, edge, and ending node of the relationship and produce new vectors for the two nodes and the edge. The new vector for the edge $v'_r = g_p(v_s, v_r, v_o)$ has dimension D_{out}, and is used as the output vector from the graph convolution layer for the edge. The new vectors for the starting and ending nodes $\tilde{v}_s = g_s(v_s, v_r, v_o)$ and $\tilde{v}_o = g_o(v_s, v_r, v_o)$ are *candidate vectors* of dimension H. In practice the three functions g_s, g_p, and g_o and implemented with a single multilayer perceptron (MLP) whose architecture is shown in Table 3.

As a second stage of processing, for each object in the scene graph we collect all of its candidate vectors and process them with a symmeitric pooling function h which converts the set of candidate vectors into a a single vector of dimension D_{out}. Concretely, for object o_i in the scene graph G, let $V_i^s = \{g_s(v_i, v_r, v_j) : (o_i, r, o_j) \in G\}$ be the set of candidate vectors for o_i from relationships where o_i appears as the subject, and let $V_i^o = \{g_o(v_j, v_r, v_i) : (o_j, r, o_i) \in G\}$ be the set of candidate vectors for o_i from relationships where o_i appears as the object of the relationship. The pooling function h takes as input the two sets of vectors V_i^s and V_i^o, averages them, and feeds the result to an MLP to compute the output vector v'_i for object o_i from the graph convolution layer. The exact architecture of the network we use for h is shown in Table 4.

Index	Inputs	Operation	Output Shape
(1)	-	Subject candidate set V_i^s	$\lvert V_i^s\rvert \times H$
(2)	-	Object candidate set V_i^o	$\lvert V_i^o\rvert \times H$
(3)	(1), (2)	Set union	$(\lvert V_i^s\rvert + \lvert V_i^o\rvert) \times H$
(4)	(3)	Mean over axis 0	H
(5)	(4)	Linear($H \to H$)	H
(6)	(5)	ReLU	H
(7)	(6)	Linear($H \to D_{out}$)	D_{out}
(8)	(7)	ReLU	D_{out}

Table 4. Network architecture for the second network h used in graph convolution; this network implements a symmetric pooling function to convert the set of all candidate vectors for an object into a single output vector.

Index	Inputs	Operation	Output Shape
(1)	-	Graph objects	O
(2)	-	Graph relationships	R
(3)	(1)	Object Embedding	$O \times 128$
(4)	(2)	Relationship embedding	$R \times 128$
(5)	(1), (2)	gconv($128 \to 512 \to 128$)	$O \times 128, R \times 128$
(6)	(5)	gconv($128 \to 512 \to 128$)	$O \times 128, R \times 128$
(7)	(6)	gconv($128 \to 512 \to 128$)	$O \times 128, R \times 128$
(8)	(7)	gconv($128 \to 512 \to 128$)	$O \times 128, R \times 128$
(9)	(8)	gconv($128 \to 512 \to 128$)	$O \times 128, R \times 128$

Table 5. Architecture of the graph convolution network used to process input scene graphs. The input scene graph has O objects and R relationships. Due to weight sharing in graph convolutions, the same network can process graphs of any size or topology. The notation gconv($D_{in} \to H \to D_{out}$) is graph convolution with input dimension D_{in}, hidden dimension H, and output dimension D_{out}.

Overall a graph convolution layer has three hyperparameters defining its size: the *input dimension* D_{in}, the *hidden dimension* H, and the *output dimension* D_{out}. We can therefore specify a graph convolution layer with the notation gconv($D_{in} \to H \to D_{out}$).

A.2. Graph Convolution Network

The input scene graph is processed by a *graph convolution network*, the exact architecture of which is shown in Table 5. Our network first embeds the objects and relationships of the graph with embedding layers to produce vectors of dimension $D_{in} = 128$; we then use five layers of graph convolution with $D_{in} = D_{out} = 128$ and $H = 512$.

Index	Inputs	Operation	Output Shape
(1)	-	Object embedding vector	128
(2)	(1)	Linear($128 \to 512$)	512
(3)	(2)	ReLU	512
(4)	(3)	Linear($512 \to 4$)	4

Table 6. Architecture of the box regression network.

A.3. Box Regression Network

We predict bounding boxes for images using a *box regression network*. The input to the box regression network are the final embedding vectors for objects produced by the graph convolution network. The output from the box regression network is a predicted bounding box for the object, parameterized as (x_0, y_0, x_1, y_1) where x_0, x_1 are the left and right coordinates of the box and y_0, y_1 are the top and bottom coordinates of the box; all box coordinates are normalized to be in the range $[0, 1]$. The architecture of the box regression network is shown in Table 6.

A.4. Mask Regression Network

We predict segmentation masks for images using a *mask regression network*. The input to the mask regression network are the final embedding vectors for objects from the graph convolution network, and the output from the mask regresion network is a $M \times M$ segmentation mask with all elements in the range $(0, 1)$. The mask regression network is composed of a sequence of upsampling and convolution layers, terminating in a sigmoid nonlinearity; its exact architecture is shown in Table 7.

The main text of the paper states that the mask regression network uses transpose convolution, but in fact it uses upsampling and stride-1 convolutions as shown in Table 7. This error will be corrected in the camera-ready version of the paper.

A.5. Scene Layout

The final embedding vectors for objects from the graph convolution network are combined with the predicted bounding boxes and segmentation masks for objects to give a *scene layout*. The conversion from vectors, masks, and boxes to scene layouts does not have any learnable parameters.

The scene layout has shape $D \times H \times W$ where $D = 128$ is the dimension of embededing vectors for objects from the graph convolution network and $H \times W = 64 \times 64$ is the output resolution at which images will be generated.

A.6. Cascaded Refinement Network

The scene layout is converted to an image using a *Cascaded Refinement Network* (CRN) consisting of a number of *Cascaded Refinement Modules* (CRMs).

Index	Inputs	Operation	Output Shape
(1)	-	Object embedding vector	128
(2)	(1)	Reshape	$128 \times 1 \times 1$
(3)	(2)	Upsample	$128 \times 2 \times 2$
(4)	(3)	Batch Normalization	$128 \times 2 \times 2$
(5)	(4)	Conv($3 \times 3, 128 \to 128$)	$128 \times 2 \times 2$
(6)	(5)	ReLU	$128 \times 2 \times 2$
(7)	(6)	Upsample	$128 \times 4 \times 4$
(8)	(7)	Batch Normalization	$128 \times 4 \times 4$
(9)	(8)	Conv($3 \times 3, 128 \to 128$)	$128 \times 4 \times 4$
(10)	(9)	ReLU	$128 \times 4 \times 4$
(11)	(10)	Upsample	$128 \times 8 \times 8$
(12)	(11)	Batch Normalization	$128 \times 8 \times 8$
(13)	(12)	Conv($3 \times 3, 128 \to 128$)	$128 \times 8 \times 8$
(14)	(13)	ReLU	$128 \times 8 \times 8$
(15)	(14)	Upsample	$128 \times 16 \times 16$
(16)	(15)	Batch Normalization	$128 \times 16 \times 16$
(17)	(16)	Conv($3 \times 3, 128 \to 128$)	$128 \times 16 \times 16$
(18)	(17)	ReLU	$128 \times 16 \times 16$
(19)	(18)	conv($1 \times 1, 128 \to 1$)	$1 \times 16 \times 16$
(20)	(19)	sigmoid	$1 \times 16 \times 16$

Table 7. Architecture of the mask regression network. For 3D tensors we use $C \times H \times W$ layout, where C is the number of channels in the feature map and H and W are the height and width of the feature map. The notation Conv($K \times K, C_{in} \to C_{out}$) is a convolution with $K \times K$ kernels, C_{in} input channels and C_{out} output channels; all convolutions are stride 1 with zero padding so that their input and output have the same spatial size. Upsample is a 2×2 nearest-neighbor upsampling.

Each CRM recieves as input the scene layout of shape $D \times H \times W = 128 \times 64 \times 64$ and the previous feature map, and outputs a new feature map twice the spatial size of the input feature map. Internally each CRM upsamples the input feature map by a factor of 2, and downsamples the layout using average pooling the match the size of the upsampled feature map; the two are concatenated and processed with two convolution layers. A CRM taking input of shape $C_{in} \times H_{in} \times W_{out}$ and producing an output of shape $C_{out} \times H_{out} \times W_{out}$ (with $H_{out} = 2H_{in}$ and $W_{out} = 2W_{in}$ is denoted as CRM($H_{in} \times W_{in},\ C_{in} \to C_{out}$). The exact architecture of our CRMs is shown in Table 8.

Our Cascaded Refinement Network consists of five Cascaded Refinement Modules. The input to the first module is Gaussian noise of shape $32 \times 2 \times 2$ and the output from the final module is processed with two final convolution layers to produce the output image. The architecture of the CRN is shown in Table 9.

A.7. Batch Normalization in the Generator

Most implementations of batch normalization operate in two modes. In *train* mode, minibatches are normalized using the empirical mean and variance of features; in *eval* mode a running mean of feature means and variances are used to normalize minibatches instead. We found that training models in *train* mode and running them in *eval* mode at

Index	Inputs	Operation	Output Shape
(1)	-	Scene Layout	$D \times H \times W$
(2)	-	Input features	$C_{in} \times H_{in} \times W_{in}$
(3)	(1)	Average Pooling	$D \times H_{out} \times W_{out}$
(4)	(2)	Upsample	$C_{in} \times H_{out} \times W_{out}$
(5)	(3), (4)	Concatenation	$(D + C_{in}) \times H_{out} \times W_{out}$
(6)	(5)	Conv$(3 \times 3, D + C_{in} \to C_{out})$	$C_{out} \times H_{out} \times W_{out}$
(7)	(6)	Batch Normalization	$C_{out} \times H_{out} \times W_{out}$
(8)	(7)	LeakyReLU	$C_{out} \times H_{out} \times W_{out}$
(9)	(8)	Conv$(3 \times 3, C_{out} \to C_{out})$	$C_{out} \times H_{out} \times W_{out}$
(10)	(9)	Batch Normalization	$C_{out} \times H_{out} \times W_{out}$
(11)	(10)	LeakyReLU	$C_{out} \times H_{out} \times W_{out}$

Table 8. Architecture of a Cascaded Refinement Module CRM($H_{in} \times W_{in}$, $C_{in} \to C_{out}$). The module accepts as input the scene layout, and an input feature map of shape $C_{in} \times H_{in} \times W_{in}$ and produces as output a feature map of shape $C_{out} \times H_{out} \times W_{out}$ where $H_{out} = 2H_{in}$ and $W_{out} = 2W_{in}$. For LeakyReLU nonlinearites we use negative slope 0.2.

Index	Inputs	Operation	Output Shape
(1)	-	Scene Layout	$128 \times 64 \times 64$
(2)	-	Gaussian Noise	$32 \times 2 \times 2$
(3)	(1), (2)	CRN$(2 \times 2,\ 32 \to 1024)$	$1024 \times 4 \times 4$
(4)	(1), (3)	CRN$(4 \times 4,\ 1024 \to 512)$	$512 \times 8 \times 8$
(5)	(1), (4)	CRN$(8 \times 4,\ 512 \to 256)$	$256 \times 16 \times 16$
(6)	(1), (5)	CRN$(16 \times 16,\ 256 \to 128)$	$128 \times 32 \times 32$
(7)	(1), (6)	CRN$(32 \times 32,\ 128 \to 64)$	$64 \times 64 \times 64$
(8)	(7)	Conv$(3 \times 3, 64 \to 64)$	$64 \times 64 \times 64$
(9)	(8)	LeakyReLU	$64 \times 64 \times 64$
(10)	(9)	Conv$(1 \times 1, 64 \to 3)$	$3 \times 64 \times 64$

Table 9. Architecture of our Cascaded Refinement Network. CRM is a Cascaded Refinement Module, shown in Table 8. LeakyReLU uses a negative slope of 0.2.

test-time led to significant image artifacts. To overcome this limitation while still benefitting from the optimization benefits that batch normalization provides, we train our models for 100K iterations using batch normalization in *train* mode, then continue training for an additional 900K iterations with batch normalization in *eval* mode.

Since discriminators are not used at test-time, batch normalization in the discriminators is always used in *train* mode.

A.8. Object Discriminator

Our object discriminator D_{obj} inputs image pixels corresponding to objects in real or generated images; objects are cropped using their bounding boxes to a spatial size of 32×32 using differentiable bilinear interpolation. The object discriminator serves two roles: it classifies objects as real or fake, and also uses an *auxiliary classifier* which attempts to classify each object. The exact architecture of our object discriminator is shown in Table 10.

Index	Inputs	Operation	Output Shape
(1)	-	Object crop	$3 \times 32 \times 32$
(2)	(1)	Conv$(4 \times 4, 3 \to 64, s2)$	$64 \times 16 \times 16$
(3)	(2)	Batch Normalization	$64 \times 16 \times 16$
(4)	(3)	LeakyReLU	$64 \times 16 \times 16$
(5)	(4)	Conv$(4 \times 4, 64 \to 128, s2)$	$128 \times 8 \times 8$
(6)	(5)	Batch Normalization	$128 \times 32 \times 32$
(7)	(6)	LeakyReLU	$128 \times 32 \times 32$
(8)	(7)	Conv$(4 \times 4, 128 \to 256, s2)$	$256 \times 4 \times 4$
(9)	(8)	Global Average Pooling	256
(10)	(9)	Linear$(256 \to 1024)$	1024
(11)	(10)	Linear$(1024 \to 1)$	1
(12)	(10)	Linear$(1024 \to \lvert\mathcal{C}\rvert)$	$\lvert\mathcal{C}\rvert$

Table 10. Architecture of our object discriminator D_{obj}. The input to the object discriminator is a 32×32 crop of an object in either a generated or real image. The object discriminator outputs both a score for real / fake (11) and a classification score over the object categories $\mathcal{C}$ (12). In this model all convolution layers have stride 2 and no zero padding. LeakyReLU uses a negative slope of 0.2.

A.9. Image Discriminator

Our image discriminator D_{img} inputs a real or fake image, and classifies an overlapping grid of 8×8 image patches from its input image as real or fake. The exact architecture of our image discriminator is shown in Table 11.

A.10. Higher Image Resolutions

We performed preliminary experiments with a version of our model that produces 128×128 images rather than 64×64 images. For these models we compute the scene layout at 128×128 rather than at 64×64; we also add an extra Cascaded Refinement Module to our Cascaded Refinement Network; we add one additional convolutional layer to both D_{obj} and D_{img}, and for these models D_{obj} receives a 64×64 crop of objects rather than a 32×32 crop. During trainging we reduce the batch size from 32 to 24.

The images in Figure 6 from the main paper were generated from a version of our model trained to produce 128×128 images from Visual Genome.

B. Image Loss Functions

In Figure 9 we show additional qualitative results from our model trained on COCO, comparing the results from different ablated versions of our model.

Omitting the discriminators from the model (L1 only) tends to produce images that are overly smoothed. Without the object discriminator (No D_{obj}) objects tend to be less recognizable, and without the image discriminator (No D_{img}) the generated images tend to appear less realistic overall, with low-level artifacts. Our model trained to use ground-truth layouts rather than predicting its own layouts (GT Layout) tends to produce higher-quality images, but

Index	Inputs	Operation	Output Shape
(1)	-	Image	$3 \times 64 \times 64$
(2)	(1)	Conv($4 \times 4, 3 \rightarrow 64, s2$)	$64 \times 32 \times 32$
(3)	(2)	Batch Normalization	$64 \times 32 \times 32$
(4)	(3)	LeakyReLU	$64 \times 32 \times 32$
(5)	(4)	Conv($4 \times 4, 64 \rightarrow 128, s2$)	$128 \times 16 \times 16$
(6)	(5)	Batch Normalization	$128 \times 16 \times 16$
(7)	(6)	LeakyReLU	$128 \times 16 \times 16$
(8)	(7)	Conv($4 \times 4, 128 \rightarrow 256, s2$)	$256 \times 8 \times 8$
(9)	(8)	Conv($1 \times 1, 256 \rightarrow 1$)	$1 \times 8 \times 8$

Table 11. Architecture of our image discriminator D_{img}. The input to the image discriminator is either a real or fake image, and it classifies an overlapping 8×8 grid of patches in the input image as either real or fake. All but the final convolution have a stride of 2, and all convolutions use no padding. LeakyReLU uses a negative slope of 0.2.

requires both bounding-box and segmentation mask annotations at test-time.

The bottom row of Figure 9 also shows a typical failure case, where all models struggle to synthesize a realistic image from a complex scene graph for an indoor scene.

C. User Study

As discussed in Section 4.5 of the main paper, we perform two user studies on Amazon Mechanical Turk to compare the perceptual quality of images generated from our method with those generated using StackGAN.

In the first user study, we show users an image generated from a COCO caption using StackGAN, and an image generated using our method from a scene graph built from the COCO object annotations corresponding to the caption. We ask users to select the image that better matches the caption. In each trial of this user study the order of our image and the image from StackGAN are randomized.

In the second user study, we again show users images generated using both methods, and we ask users to select the COCO objects that are visible in the image. In this experiment, if a single image contains multiple instances of the same object category then we only ask about its presence once. In each Mechanical Turk HIT users see an equal number of results from StackGAN and our method, and the order in which they are presented is randomized.

For both studies we use 1024 images from each method generated from COCO val annotations. All images are seen by three workers, and we report all results using majority opinions.

StackGAN produces 256×256 images, but our method produces 64×64 images. To prevent the differing image resolution from affecting worker opinion, we downsample StackGAN results to 64×64 using bicubic interpolation before presenting them to users.

Figure 9. Example images generated from our model and ablations on COCO. We show the original image, the synthetic scene graph generated from the COCO annotations for the image, and results from several versions of our model. Our model with no discriminators (L1 only) tends to be overly smooth; omitting the object discriminator (No D_{obj}) causes objects to be less recognizable; omitting the image discriminator (No D_{img}) leads to low-level image artifacts. Using the ground-truth rather than predicted layout (GT Layout) tends to result in higher quality images. The bottom row shows a typical failure case, where all versions of our model struggle with complex scene graphs for indoor scenes. Graphs best viewed with magnification.

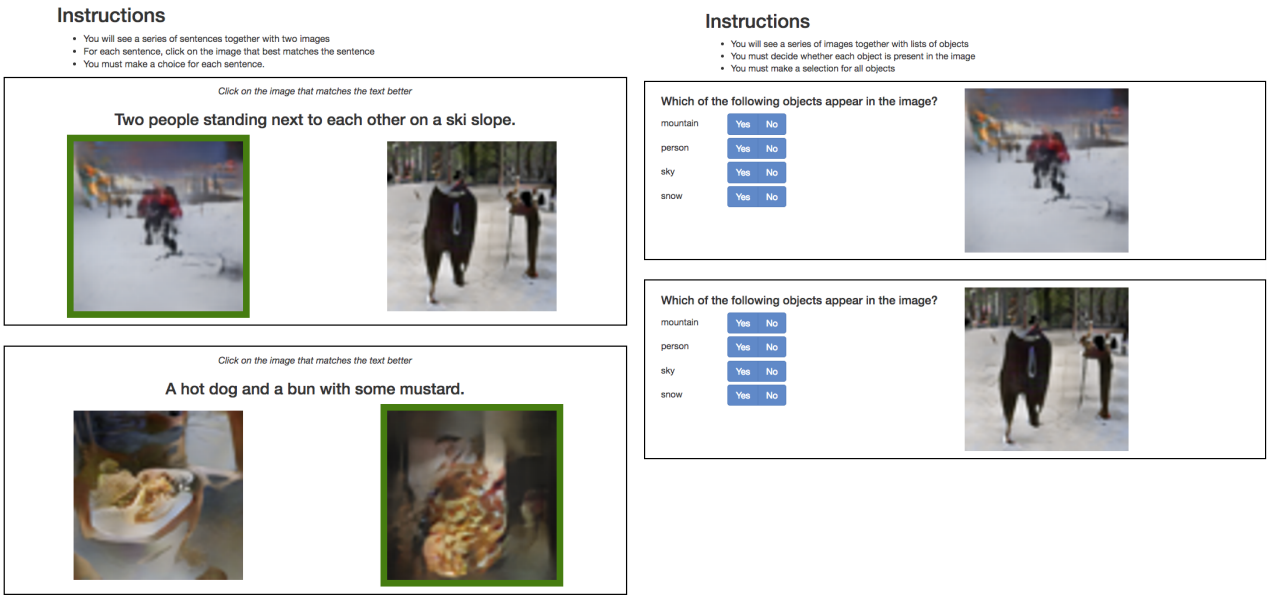

Figure 10. Screenshots of the user interfaces for our user studies on Amazon Mechanical Turk. **Left:** User interface for the user study from Figure 7 of the main paper. We show users an image generated by StackGAN from a COCO caption, and an image generated with our method from a scene graph built from the COCO object annotations corresponding to the caption. We ask users to select the image that best matches the caption. **Right:** User interface for the user study from Figure 8 of the main paper. We show again show users images generated using StackGAN and our method, and we ask users which COCO objects are present in each image.