Supplementary Material for "Learning Canonical Representations for Scene Graph to Image Generation"

In this supplementary file we provide additional implementation details, empirical results, and a proof of correctness for the SGC algorithm.

1 Scene-Graph-to-Layout

In Section 3.2, we introduced the WSGC-E and WSGC-S methods, two different procedures proposed for mapping an input scene graph into a weighted canonicalized relation graph. As mentioned in Section 3.2, although the WSGC-E is a natural extension of the SGC procedure, it is impractical for large complex graphs whereas the WSGC-S method adds fewer edges and is thus more practical for training. In what follows, we provide additional details about WSGC-E and WSGC-S, as well as comparison and analysis.

1.1 Exact Weighted Scene Graph Canonicalization (WSGC-E)

Next, we describe in detail the WSGC-E method for obtaining a weighted relation graph that is a natural extension of SGC. The WSGC-E begins with the user-specified graph E, with weights of one. Next two weighted completion steps are performed, corresponding to the SGC steps.

Converse Completion: In SGC, this step adds all converse edges. In the weighted case it makes sense to add the converse edge with its corresponding converse weight. For example, if the graph E contains the edge $(i, \text{above}, j, 1)$ and $p^{conv}(\text{below}|\text{above}) = 0.7$, we add the edge $(j, \text{below}, i, 0.7)$. See Figure 1b.

Transitive Completion: In SGC, all transitive edges are found and added. In the weighted case, a natural alternative is to set a weight of a path to be the product of weights along this path, and set the weight of a completed edge (i, r, j) to be the maximum weight of a path between i and j times the probability $p^{trans}(r)$ that the relation is transitive. See Figure 1c. The maximum path weight problem is equivalent to maximizing the sum of log probabilities, and since these are all negative, this can be solved in polynomial time via a shortest weight path algorithm (e.g., FW). However, when there are many nodes and relations, runtime can still be substantial, and thus we offer a faster approach next.

Training with WSGC-E. In the main text, we described the training loss and optimization for WSGC-S. Optimizing the loss for WSGC-E is similar, as we explain next. We describe the loss of the WSGC-E method for a single input scene graph E and its ground truth layout Y. The parameters of the model are as follows: θ^g are the parameters of the GCN in Section 4, θ^{trans} are the parameters of the transitive probability (Eq. 1), and θ^{conv} are those of the converse probability (Eq. 2). Let θ denote the set of all parameters. Denote the

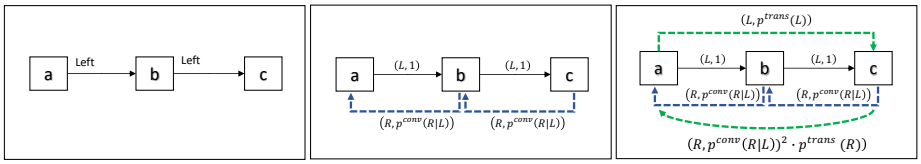

Fig. 1: An illustration of WSGC-E where relations are Left (L) and Right (R). (a) The input graph contains two relations with weight 1. (b) Converse edges (blue dashed arrows) are completed with the weights p^{conv}. (c) Transitive edges (green dashed arrows) are added and assigned the weight of the corresponding path times p^{trans}.

GCN applied to this graph by G_{θ^g}. We use L_1 as the loss between predicted and ground-truth bounding boxes Y. Namely, we wish to minimize the following objective (we write WSGC instead of WSGC-E below for brevity):

$$L(\theta) = \left\| Y - G_{\theta^g}(\text{WSGC}(E; \theta^{trans}, \theta^{conv})) \right\|_1 \tag{1}$$

When calculating $L(\theta)$, most of the operations are standard and are differentiated automatically by PyTorch. The only apparent complication is with the minimum weight path. However, we next explain why there is actually not a problem and one can simply take the gradient of the PyTorch computation graph for $L(\theta)$, which includes the minimum-weight-path computation.

Recall that in WSGC-E we first weight each edge by the corresponding converse weights p^{conv}_θ. Let $w(e;\theta)$ denote the weight of edge e after this weighting step. Next, we perform a transitive completion as follows. Given an edge e', its new weight will be (up to the multiplicative factor of p^{trans} which we leave out for brevity):

$$w_{trans}(e';\theta) = \max_{P \in \mathcal{P}} \prod_{e \in P} w(e;\theta) = e^{\max_{P \in \mathcal{P}} \sum_{e \in P} \log w(e;\theta)} \tag{2}$$

where $\mathcal{P}$ are all the paths between the two incident nodes of e'. To calculate the sub-gradient of this expression with respect to θ, we note that in the exponent we have a maximum over linear functions, and thus differentiating it wrt θ corresponds to finding the maximizing $P*$ and then differentiating $\sum_{e \in P*} \log w(e;\theta)$.

Technically, the above suggests a very simple PyTorch implementation. Implement the computation graph for Eq. 1, including the non-differentiable maximum weight path computation. And then let PyTorch take gradients for this graph. Since the maximum-weight-path cannot be differentiated through, the computation will fix the maximizing path and take the gradient there, and this is indeed the correct sub-gradient, as per our discussion above.

The downside of the WSGC-E method is that it assigns weights to all edges in the graph, and for all relations, and thus computations involve a dense graph, which makes training and inference slow. This motivates our use of WSGC-S which uses sparser graphs.

1.2 Empirical Comparison of WSGC-E and WSGC-S

Table 1 shows a comparison of WSGC-E and WSGC-S on the standard COCO and VG datasets, where WSGC-E runs in a reasonable time so that comparison is possible. The size of the graphs on the standard datasets is less than an average of 1000 triplets per image, while on the packed datasets it is $24,000$ triplets per image. Thus it is impossible to run the WSGC-E on packed datasets. It can be seen that the methods achieve comparable performance, suggesting that indeed WSGC-S is a scalable alternative to WSGC-E.

Method	COCO			Visual Genome		
	mIOU	R@0.3	R@0.5	mIOU	R@0.3	R@0.5
WSGC-S 5 *GCN*	41.9	**63.3**	38.2	18.0	25.9	**10.6**
WSGC-E 5 *GCN*	**42.2**	63.0	**38.7**	18.0	**26.5**	9.9

Table 1: Evaluation of WSGC-E and WSGC-S on Standard COCO and Visual Genome.

1.3 Analysis of Learned Weights

Our approach parameterizes the weights of converse and transitive relations and learns these parameters from data. It is interesting to see whether the learned weights recover known converse and transitive relations.

Inspecting the converse weights p^{conv} that were learned on the standard COCO dataset reveals that all weights have converged to values close to 0 and 1, and align well with the expected true converse relation. Specifically, weights corresponding to converse pairs such as ("below", "above") all converged to 1, while the rest of the pairs, such as ("left of", "inside") converged to 0. For transitive weights p^{trans}, 5/6 of the transitive relations correctly converged to 1 and a single relation to 0. Concretely, "above", "left of", "right of", "inside" and "below" converged to 1 while "surrounding" did not. The learned values are shown in Figure 2.

1.4 Weighted Graph Convolutional Network

In Section 4, we presented Graph Convolutional Network (GCN) [6] as a natural architecture for the SG to layout task. We use a similar approach to recent methods [1, 2] for this task, but modify the GCN to our weighted scene graph. This is done by revising the graph convolution layer such that the aggregation step of each node is set to be a weighted average, where the weights are those in the canonical SG. In what follows, we provide additional details about our Weighted GCN.

Each object category $c \in \mathcal{C}$ is assigned a learned embedding $\boldsymbol{\phi}_c \in \mathbb{R}^D$ and each relation $r \in \mathcal{R}$ is assigned a learned embedding $\boldsymbol{\psi}_r \in \mathbb{R}^D$. Given an SG with

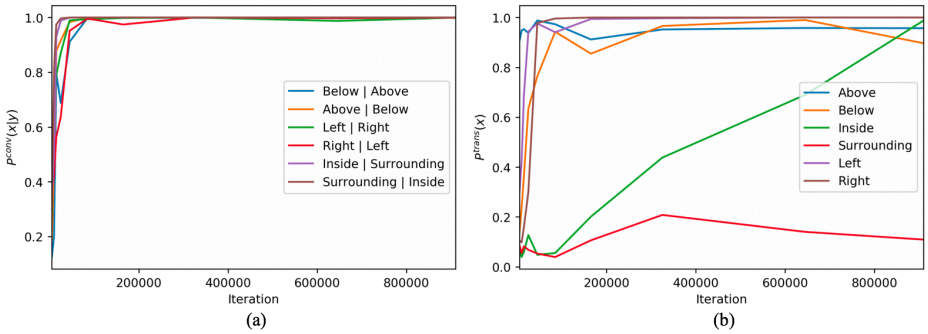

Fig. 2: Learned p^{conv} and p^{trans} weights for the WSGC-S model on the COCO dataset. The learned values of p^{conv} (see a) and of p^{trans} (see b) are presented as function of training iteration.

N objects, the GCN iteratively calculates a representation for each object and each relation in the graph. Let $\boldsymbol{v}_i^k \in \mathbb{R}^d$ be the representation of the i^{th} object in the k^{th} layer of the GCN. Similarly, for each edge $e = (i, r, j, w)$ in the graph let $\boldsymbol{u}_e^k \in \mathbb{R}^d$ be the representation of the relation in this edge. These representations are calculated as follows. Initially we set: $\boldsymbol{v}_i^0 = \boldsymbol{\phi}_{o(i)}, \boldsymbol{u}_e^0 = \boldsymbol{\psi}_{r(e)}$, where $r(e)$ is the relation for edge e. Next, we use three functions (MLPs) F_s, F_r, F_o, each from $\mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}^D$ to $\mathbb{R}^D$ to obtain an updated object representation (see Section 4.1 for implementation details). These can be thought of as processing three vectors on an edge (the subject, relation and object representations) and returning three new representations. Given these functions, the updated object representation is the weighted average of all edges incident on i:[1]

$$\boldsymbol{v}_i^{t+1} = \frac{1}{c}\left[\sum_{e=(i,r,j,w)} wF_s(\boldsymbol{v}_i^t, \boldsymbol{u}_e^t, \boldsymbol{v}_j^t) + \sum_{e=(j,r,i,w)} wF_o(\boldsymbol{v}_j^t, \boldsymbol{u}_e^t, \boldsymbol{v}_i^t)\right] \tag{3}$$

where c is a normalizing constant $c = \sum_{e=(i,r,j,w)} w + \sum_{e=(j,r,i,w)} w$. For the edge we set: $\boldsymbol{u}_e^{t+1} = F_r(\boldsymbol{v}_i^{t+1}, \boldsymbol{u}_e^t, \boldsymbol{v}_j^{t+1})$.

After iterating the GCN for L updates, the layout for node i is obtained by applying an MLP with four outputs to $\boldsymbol{v}_i^L$.[2] Note that F_s, F_r, F_o and w depend on learned parameters which are optimized using gradient descent.

1.5 Generalization on Packed Scenes

To further test the effect of model capacity from Table 1 in the paper, we even trained bigger Sg2Im models with $32, 64$ layers on Packed COCO, resulting in

[1] Note that a box can appear both as a "subject" and an "object" thus two different sums in the denominator and the normalization is needed because we want to obtain a new single object representation while the number of object occurrences is varied.

[2] The MLP has a sigmoid activation in the last layer so that the predicted normalized bounding box coordinates are in $[0, 1]$.

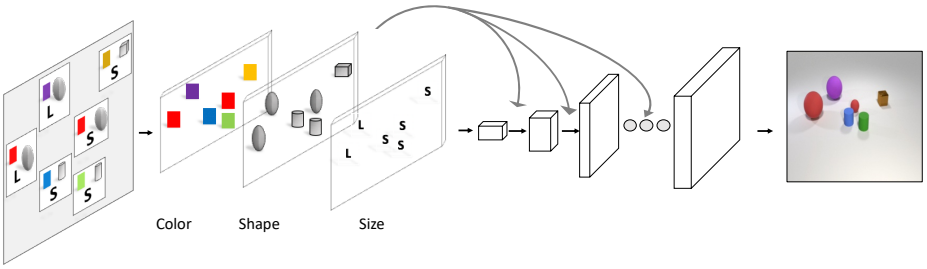

Fig. 3: Generating images with our AttSPADE model. Given a layout of boxes, our model generates an image using the layout into a series of residual blocks with upsampling layers. The layout is modeled by multiple semantic attributes per box rather than a single class descriptor.

IOU of 36.93, 11.65. We also trained a Sg2Im model with 1024 hidden units and 16 layers, and IOU deteriorated to 37.01. These results suggest that increasing the capacity of Sg2Im leads to overfitting and that WSGC improvement is indeed due to canonicalization.

2 Layout-to-Image with AttSPADE

For the CLEVR dataset [3], we use a novel generator, which we refer to as AttSPADE. This generator can be used for directly controlling attributes of the generated image, and this is not supported by other generators such as LostGAN. Although the generator is not the main focus of our contribution, we believe it is of independent interest, and thus describe it in some detail below, and show images that it generates.

2.1 The AttSPADE Model

The key idea in the AttSPADE model is to condition generation on the attribues, as opposed to only the object class as done in current models. In what follows we describe the model.

We consider the case where a bounding box has an associated set of attributes. For example, the object category is an attribute and the size is an attribute (with possible values "small", "medium" and "large"). Additionally, if a segmentation mask is provided as input, it can be added as a binary attribute. We encode this set of attributes via a multi-hot vector $\boldsymbol{z} \in \mathbb{R}^r$ that is set to one for the corresponding attributes, and apply a FC layer to it to obtain a vector $\boldsymbol{v} \in \mathbb{R}^d$. Next, we construct a tensor $M \in \mathbb{R}^{d \times H \times W}$ where H and W are the boxes height and width and $M[:, i, j] = \boldsymbol{v}$. This encodes the attributes for each pixel in the bounding box.[3] Finally, we use M as input to a SPADE [10] gen-

[3] We note that different pixels may have different attributes in principle although we don't use this here.

erator to obtain the generated image. Thus, our approach simply replaces the input of the SPADE model (which is just an object mask) with the tensor M.

Lastly, our model uses two discriminators: one for the image (to achieve a better quality of the entire image), and one for the boxes (in order to better capture each box). This is similar to [1, 2] but with a few modifications (see next section). A high level description of the architecture is shown in Figure 3.

2.2 The Loss Functions

Our AttSPADE model contains several modifications of the loss functions. First, the generator is trained with the same multi-scale discriminator and loss function used in pix2pixHD [12], except we replace the squared error loss [8] with the hinge loss [7, 9, 13]. Second, since our layout-to-image model generates the image from a given layout of bounding boxes, we add a box term loss to guarantee that the generated objects in these boxes look real. For this purpose, we crop the bounding boxes to create object images and train the discriminator to discriminate between real object images and generated object images. The image discriminator is implemented as in SPADE [10].

2.3 Baseline Models

We report generation results that vary both the layout being used and the layout-to-image component. For the layout we consider three options: (1) Ground truth layout. (2) Our WSGC predicted layout. (3) The layout used in Sg2Im [2]. For the image generation we use three options: (1) Our AttSpade generator. (2) The LostGAN generator [11] (the most recent state-of-the-art generation model). (3) The Grid2Im [1] generator, which uses the same graph model as [2]. The results reported in [1] use a coarse version of the GT layout (i.e., the layout rounded to a 5×5 grid). Since this variant comes close to actually using the GT layout, we also consider an additional version of [1] that does not use this information. We refer to this version as "Grid2Im No-Att" (code provided by the authors of [1]).

For a fair comparison, all models were tested with the same external code evaluation metrics.

2.4 Results

The results in Table 2 suggest that the AttSPADE model improves over previous approaches [1, 11] when generating an image from a GT layout, in both resolutions. In addition, our end-to-end model, which includes the WSGC and AttSPADE model, outperforms most of the baselines on the COCO and Visual Genome datasets.

Figure 5 shows a direct comparison between different generators using GT layout for COCO. It can be seen that AttSPADE provides higher quality images than the other generators.

Resolution	Methods		Inception Score		FID		Diversity Score	
	SG-to-Layout	Layout-to-Image	COCO	VG	COCO	VG	COCO	VG
128x128	Real Images	Real Images	23.0 ± 0.4	22.8 ± 1.7	-	-	-	-
	GT Layout	Grid2Im [1]	12.5 ± 0.3	-	59.5	-	-	-
	GT Layout	LostGAN [11]	11.8 ± 0.3	8.9 ± 0.3	64.0	66.7	**0.57 ± 0.06**	**0.59 ± 0.06**
	GT Layout	AttSPADE (Ours)	**15.6 ± 0.5**	**11.7 ± 0.8**	**54.7**	**36.4**	0.44 ± 0.09	0.51 ± 0.08
	WSGC	LostGAN [11]	**11.1 ± 0.6**	8.1 ± 0.3	**65.9**	73.4	0.57 ± 0.06	0.58 ± 0.06
	Sg2Im [2]	Grid2Im [1]	10.4 ± 0.4	-	75.4	-	-	-
	WSGC	AttSPADE (Ours)	10.8 ± 0.5	**10.0 ± 0.7**	73.8	**46.4**	**0.57 ± 0.06**	**0.58 ± 0.06**
256x256	Real Images	Real Images	30.3 ± 1.4	31.7 ± 2.0	-	-	-	-
	GT Layout	Grid2Im [1]	16.4 ± 0.7	-	65.2	-	0.48 ± 0.09	-
	GT Layout	AttSPADE (Ours)	**19.5 ± 0.9**	**16.9 ± 1.2**	**64.65**	**42.9**	**0.55 ± 0.11**	**0.62 ± 0.08**
	Sg2Im [2]	Grid2Im [1] No-Att	6.6 ± 0.3	-	127.0	-	0.65 ± 0.05	-
	WSGC	AttSPADE (Ours)	**13.9 ± 0.3**	**16.5 ± 0.7**	**119.1**	**45.7**	**0.70 ± 0.07**	**0.68 ± 0.07**

Table 2: Quantitative comparisons for SG-to-image methods using Inception Score (higher is better), FID (lower is better) and Diversity Score (higher is better). Evaluation is done on the COCO-Stuff and VG datasets.

Figure 6 shows different generators that use both GT and generated layouts for COCO. Additional qualitative results on Visual Genome can be seen in Figure 7 and Figure 8. In the generation results it can be seen that AttSPADE produces more realistic images, when compared to other generators. Furthermore when using WSGC layout the images are qualitatively similar to using GT layout, which suggests that WSGC produces high quality layouts.

3 Datasets

3.1 Synthetic dataset

In Section 6, a synthetic dataset which was used to explore properties of the suggested WSGC model was presented. Example cases from this dataset are included in Figure 4. More specifically, this dataset was utilized to evaluate the contribution of the transitivity closure on the scene-graph-to-layout task.

Every object in this data is a square with one of two possible sizes, *small* or *large*. The set of relations includes:

- *Above* - The center of the subject is above the object. This relation is transitive.
- *OppositeHorizontally* - The subject and the object are on opposite sides of the image with respect to the middle vertical line. This relation is not transitive.
- $XNear$ - The subject and object are within distance equal to 10% of the image with respect to the x coordinate of each center. This relation is not transitive.

To generate SG-layout pairs for training and evaluation, we uniformly sample coordinates of object centers and object sizes and automatically compute relations among object pairs based on their spatial locations.

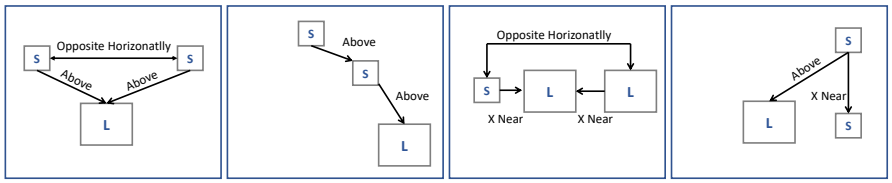

Fig. 4: Example of synthetic dataset samples. In these samples, the scene graph relations are overlaid on top of the ground truth layout. Every edge is described with a corresponding relation type and every square object is annotated with an object type: "S" for small and "L" for large.

3.2 Packed Datasets

Here we describe the specific characteristics of the packed datasets presented in the paper. For every packed dataset, only samples with at least 16 objects per image were included. The method for constructing relations for COCO and CLEVR is as described next. For VG, since Standard VG contains a limited number of relations we supplement the dataset with relations as follows. For every two graph nodes, edges representing geometric relations such as:"left", "right", "above", "below", "inside" and "surrounding" are constructed based on relative (x,y) coordinates. Redundant edges are removed such that the graph is minimal. This procedure differs from the one used in [2] in two ways: first, in [2], the decision to construct such edges is based on angles between two objects and second, in [2], there can be up to a single constructed edge for every pair of objects and the decision whether to construct or not is random. Hence, the procedure proposed here results in graphs that are more complex w.r.t number of edges and are more informative.

4 Implementation Details

4.1 Scene-Graph-to-layout

In the WSGC GCN model, we follow the implementation details proposed in [2]. We use 5 hidden layers and an embedding layer of 128 units for each object and relation. The functions F_s, F_r, F_o which were presented in Section 4, are all implemented as a single 3 layers MLP with 512 units per layer. For optimization we use Adam [4], where for $\theta^{conv}, \theta^{trans}$ we use LR of $1e^{-2}$ and otherwise we use $1e^{-4}$.

4.2 AttSPADE

We apply Spectral Norm [9] to all the layers in both generator and discriminator. We use the ADAM solver [5] with $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and a learning rate of 0.0001 for both the generator and the discriminator. All the experiments are

conducted on NVIDIA V100 GPUs. We use PyTorch synchronized BatchNorm with the following batch sizes: 32 for 128×128 and 16 for 256×256 resolutions (statistics are collected from all the GPUs). The FC layer that calculates $\boldsymbol{v} \in \mathbb{R}^d$ (used to construct tensor M. See Section 2), is set d to 128.

5 Proof that SGC outputs the closure $C(E)$ (Section 3.1)

Lemma 1. *The SGC procedure described in Section 3.1 of the main paper outputs the closure $C(E)$.*

Proof. Let $G = (O, E)$. Denote $\hat{C}$ be the canonicalization procedure proposed. To show $\hat{C}(E) = C(E)$, it suffices to prove that (1) $C(E) \subseteq \hat{C}(E)$ and (2) $\hat{C}(E) \subseteq C(E)$.

Proof that $\hat{C}(E) \subseteq C(E)$:. Let there be $e \in \hat{C}(E)$ s.t $e = (i, r, j)$. We split into cases by e construction:

- **Original graph edge**. if $e \in E$ then by C definition $e \in C(E)$.
- **Converse constructed edge**. Therefore there exists $r' \in \mathcal{R}$ such that $(r, r') \in \mathcal{R}_{conv}$ and $(j, r', i) \in E$. Then $(j, r', i) \in C(E)$ and therefore $(i, r, j) = e \in C(E)$ by definition.
- **Transitive constructed edge**. Since e was constructed in the $Transitivity$ step, it must hold that $r \in \mathcal{R}_{trans}$ and e was contained in the transitive closure of r. Therefore, after the $ConverseRelations$ step, there existed a directed path $p = (o_{v_1}, ..., o_{v_k})$ with respect to r where $v_1 = i$ and $v_k = j$. To prove $e \in C(E)$, it is enough to show that for every edge in p it is also in $C(E)$. From here, since C respects transitivity, this will follow. Namely, let there be $e' = (i', r, j') \in \{(o_{v_m}, o_{v_{m+1}}) | m \in \{1, .., k\}\}$. If $e' \in E$, then $e' \in C(E)$ and we are done. Otherwise, by the $ConverseRelations$ construction step, there exists r' such that $(r, r') \in \mathcal{R}_{conv}$ and $(j', r', i') \in E$. Therefore, it follows that $(j', r', i') \in C(E)$ and $e' \in C(E)$ and we are done.

Proof that $C(E) \subseteq \hat{C}(E)$: For every $e = (i, r, j) \in C(E)$ we need to show that $e \in \hat{C}(E)$. Since $e \in C(E)$, e is a relation implied by E. If $e \in E$, since $\hat{C}$ does not drop edges, it holds that $e \in \hat{C}(E)$ and we're done. Otherwise, we assume by contradiction that $e \notin \hat{C}(E)$. let $p = (o_{v_1}, ..., o_{v_k})$ be a directed path from o_i to o_j in $C(E)$. Then, there exists $e' = (i', r, j') \in \{(o_{v_i}, o_{v_{i+1}}) | i \leq k\}$ where $e' \notin \hat{C}(E)$. Otherwise, if there is no such e', we get that there is a directed path between o_i to o_j and by $Transitivity$ step construction $e \in \hat{C}(E)$. Therefore, there must be $e_{conv} \in E$, such that $e_{conv} = (j, r', i)$ and $(r, r') \in \mathcal{R}_{conv}$. However, from the $ConverseRelations$ step construction, if there exists such edge we get that $e \in \hat{C}(E)$, in contrary to the assumption that $e \notin \hat{C}(E)$.

6 Generalization on Semantically Equivalent Graphs

Results in Table 2 of the main paper demonstrate that the learned WSGC model is more robust to changes in the scene graph input. In this experiment, we randomly transform each test sample scene graph into a semantically equivalent one, and test models on the resulting sample. To generate such samples from a given scene graph, we start by calculating all the possible location-based relations for any pair of objects. Then, for each pair of objects we use prior knowledge to identify pairs of converse relations, and drop one of the edges in such pairs with probability $p = 0.5$. After this step, we compute the transitive closure with respect to each relation and randomly drop ($p = 0.5$) each edge that does not change the semantics of the scene graph.

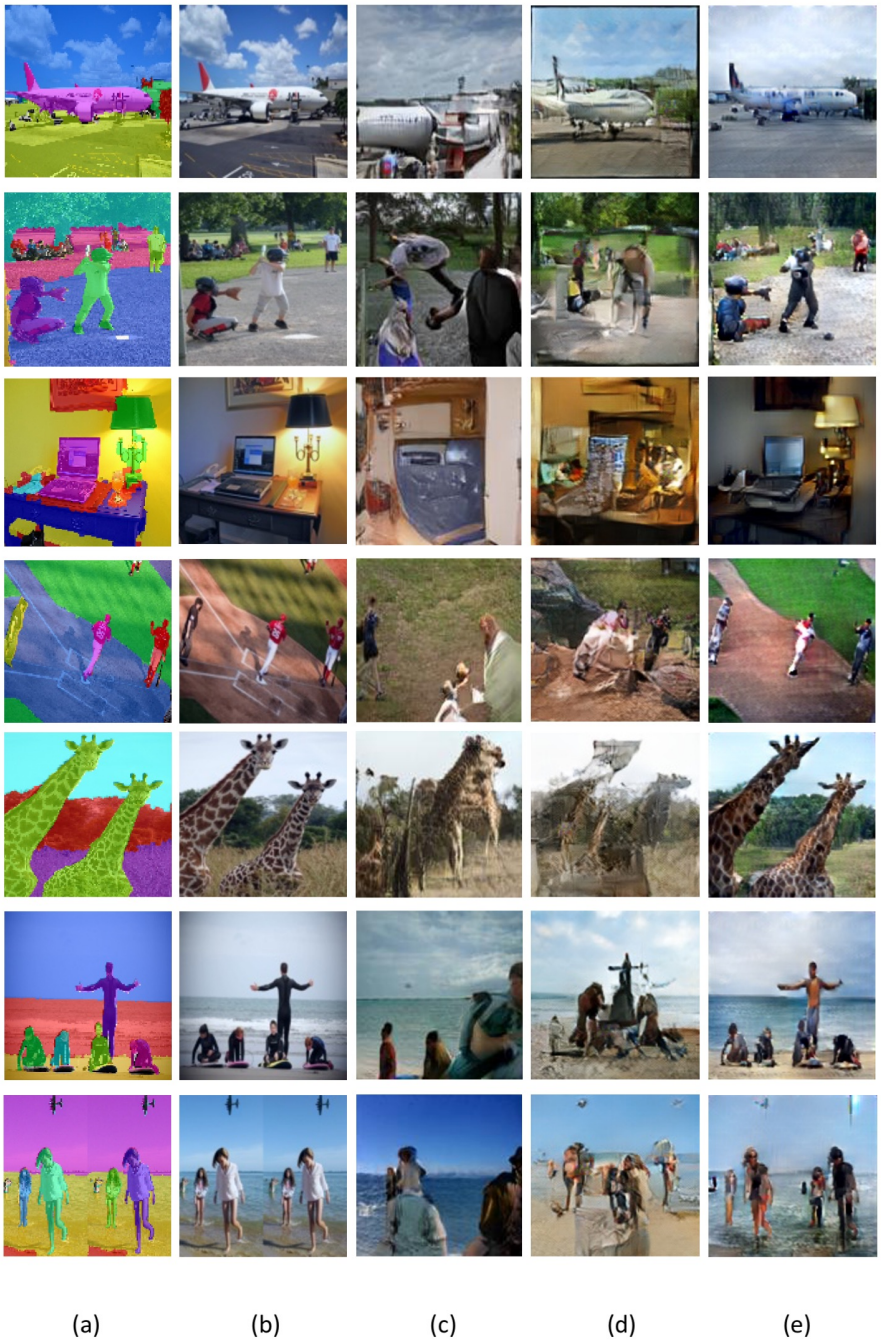

Fig. 5: Selected GT layout-to-image generation results on COCO-Stuff dataset on 128 × 128 resultion. Here, we compare our AttSPADE model, Grid2Im [1] and LostGAN [11] on generation from GT layout of masks. (a) GT layout (only masks). (b) GT image. (c) Generation with LostGAN [11] model. (d) Generation with Grid2Im [1]. (e) Generation with AttSPADE model (ours).

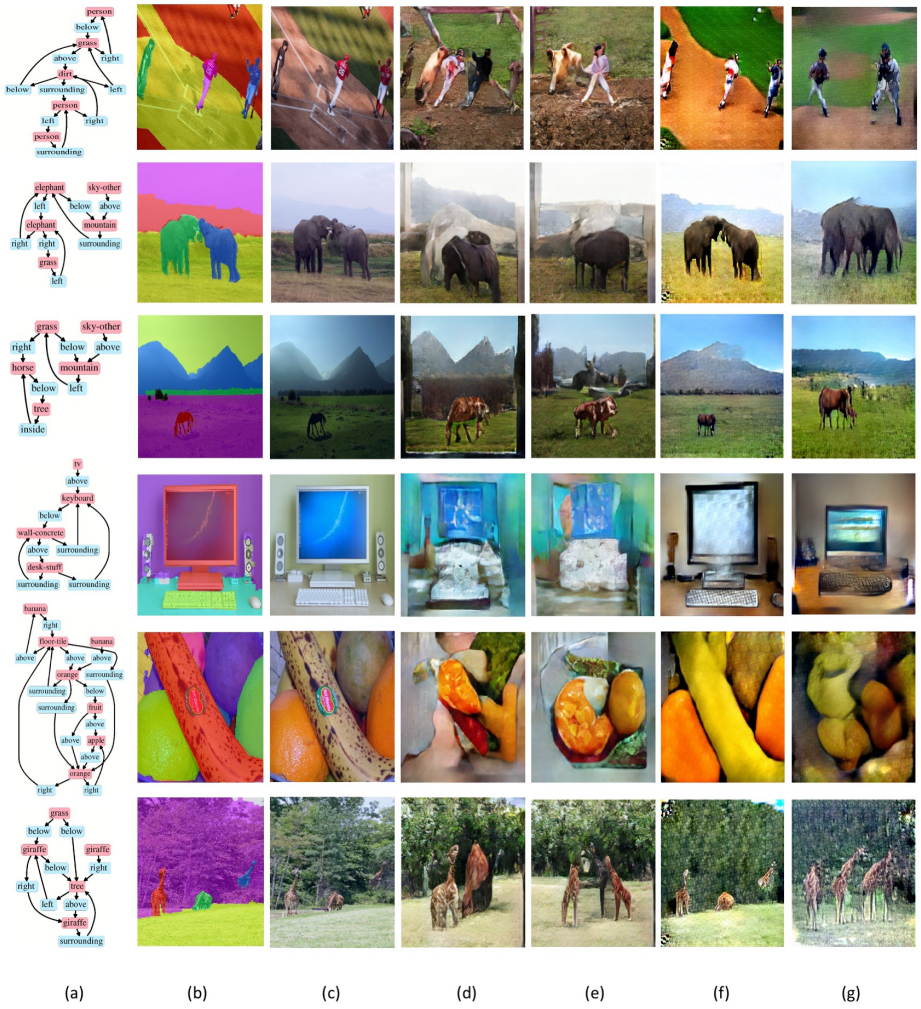

Fig. 6: Selected generation results on the COCO-Stuff dataset at 256×256 resolution. Here, we compare our AttSPADE model and Grid2Im [1] in two different settings: generation from GT layout of masks and generation from scene graphs. (a) GT scene graph. (b) GT layout (only masks). (c) GT image. (d) Generation with Grid2Im [1] using the GT layout. (e) Generation with Grid2Im No-att [1] from the scene graph (GT layout not used). (f) Generation with AttSPADE model (ours) using the GT layout. (g) Generation with WSGC + AttSPADE model (ours) from the scene graph (GT layout not used).

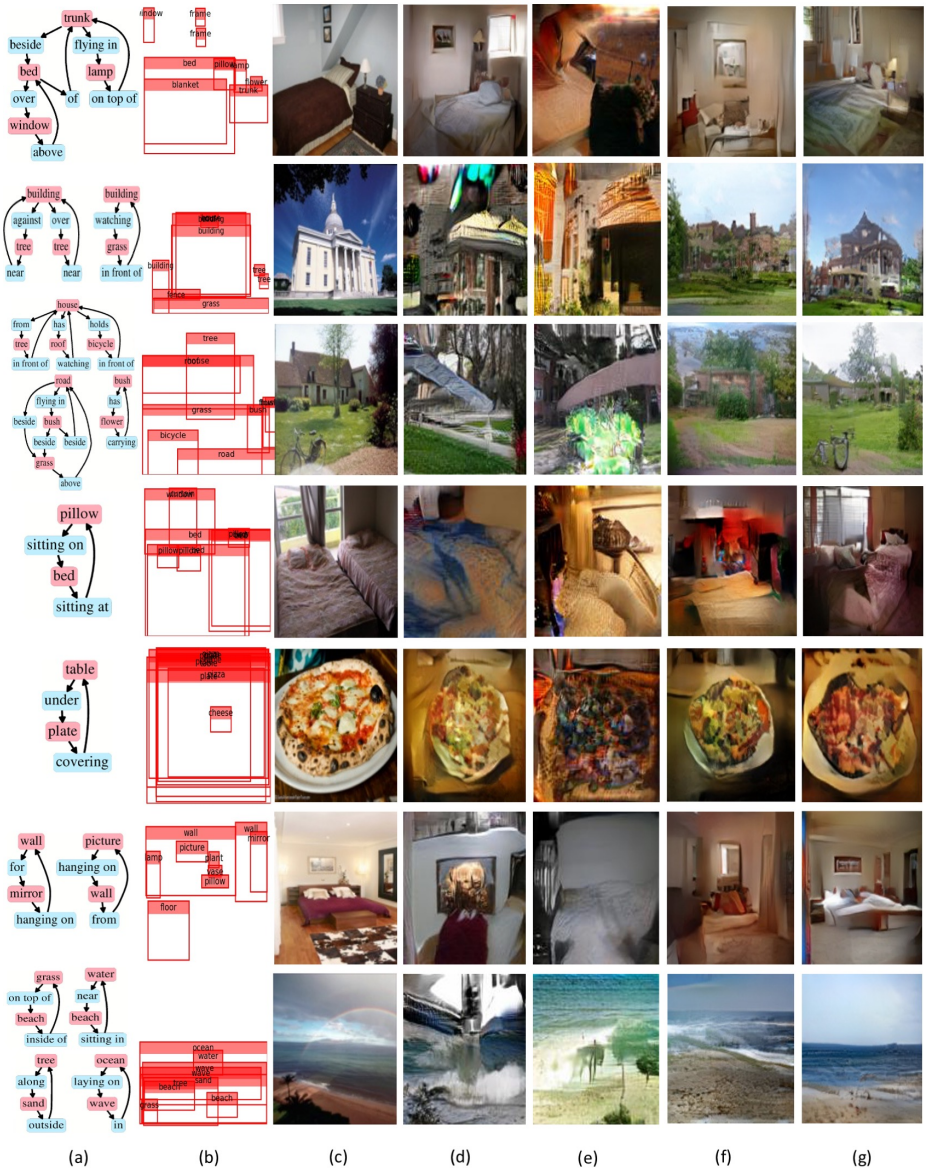

Fig. 7: Selected scene-graph-to-image results on Visual Genome dataset on 128 × 128 resolution. Here, we compare our AttSPADE model and LostGAN [11] in two different settings: generation from GT layout of boxes and generation from scene graphs. (a) GT scene graph. (b) GT layout (only boxes). (c) GT image. (d) Generation using LostGAN [11] from the GT layout. (e) Generation with the WSGC + LostGAN [11] from the scene graph (GT layout not used). (f) Generation with the AttSPADE model (ours) from the GT Layout. (g) Generation with the WSGC + AttSPADE model (ours) from the scene graph (GT layout not used).

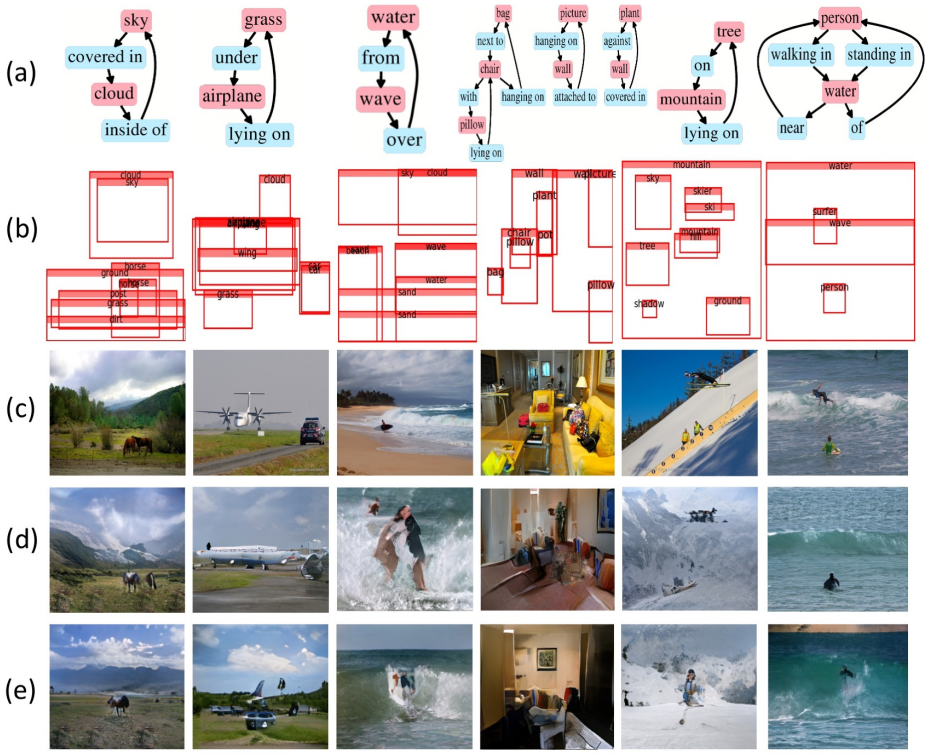

Fig. 8: Selected scene-graph-to-image results on the Visual Genome dataset at 256×256 resolution. Here, we test our AttSPADE model in two different settings: generation from GT layout of boxes and generation from scene graphs. (a) GT scene graph. (b) GT layout (only boxes). (c) GT image. (d) Generation with the AttSPADE model (ours) from the GT Layout. (e) Generation with the WSGC + AttSPADE model (ours) from the scene graph (GT layout not used).

References

1. Ashual, O., Wolf, L.: Specifying object attributes and relations in interactive scene generation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 4561–4569 (2019)
2. Johnson, J., Gupta, A., Fei-Fei, L.: Image generation from scene graphs. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
3. Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C.L., Girshick, R.: Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In: CVPR (2017)
4. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Int. Conf. on Learning Representations (2015)
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
7. Lim, J.H., Ye, J.C.: Geometric gan. arXiv preprint arXiv:1705.02894 (2017)
8. Mao, X., Li, Q., Xie, H., Lau, Y.R., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. In: Proc. Int. Conf. Comput. Vision (2017)
9. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: Int. Conf. on Learning Representations (2018)
10. Park, T., Liu, M.Y., Wang, T.C., Zhu, J.Y.: Semantic image synthesis with spatially-adaptive normalization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2337–2346 (2019)
11. Sun, W., Wu, T.: Image synthesis from reconfigurable layout and style. In: The IEEE International Conference on Computer Vision (ICCV) (October 2019)
12. Wang, T.C., Liu, M.Y., Zhu, J.Y., Tao, A., Kautz, J., Catanzaro, B.: High-resolution image synthesis and semantic manipulation with conditional gans. In: Proc. Conf. Comput. Vision Pattern Recognition (2018)
13. Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-attention generative adversarial networks. In: Int. Conf. Mach. Learning (2019)