Layout Generation and Completion with Self-attention

Kamal Gupta[1], Alessandro Achille[2], Justin Lazarow[2], Larry Davis[1,2], Vijay Mahadevan[2], and Abhinav Shrivastava[1]

[1]University of Maryland, College Park [2]Amazon AWS

Abstract. We address the problem of layout generation for diverse domains such as images, documents, and mobile applications. A layout is a set of graphical elements, belonging to one or more categories, placed together in a meaningful way. Generating a new layout or extending an existing layout requires understanding the relationships between these graphical elements. To do this, we propose a novel framework, LayoutTransformer, that leverages a self-attention based approach to learn contextual relationships between layout elements and generate layouts in a given domain. The proposed model improves upon the state-of-the-art approaches in layout generation in four ways. First, our model can generate a new layout either from an empty set or add more elements to a partial layout starting from an initial set of elements. Second, as the approach is attention-based, we can visualize which previous elements the model is attending to predict the next element, thereby providing an interpretable sequence of layout elements. Third, our model can easily scale to support both a large number of element categories and a large number of elements per layout. Finally, the model also produces an embedding for various element categories, which can be used to explore the relationships between the categories. We demonstrate with experiments that our model can produce meaningful layouts in diverse settings such as object bounding boxes in scenes (COCO bounding boxes), documents (PubLayNet), and mobile applications (RICO dataset).

Keywords: Generative modeling, Self-attention, Layout generation

1 Introduction

In the real world, there exists a strong relationship between different objects that are found in the same environment [37,35]. For example, a dining table usually has chairs around it; a surfboard is found near the sea; horses do not ride cars; *etc.* Biederman [2] provided strong evidence in cognitive neuroscience that perceiving and understanding a scene involves two related processes: *perception* and *comprehension*. Perception deals with processing the visual signal or the appearance of a scene. Comprehension deals with understanding the *schema* of a scene [2], where this schema (or layout) can be characterized by contextual relationships between objects (*e.g.*, support, occlusion, and relative likelihood,

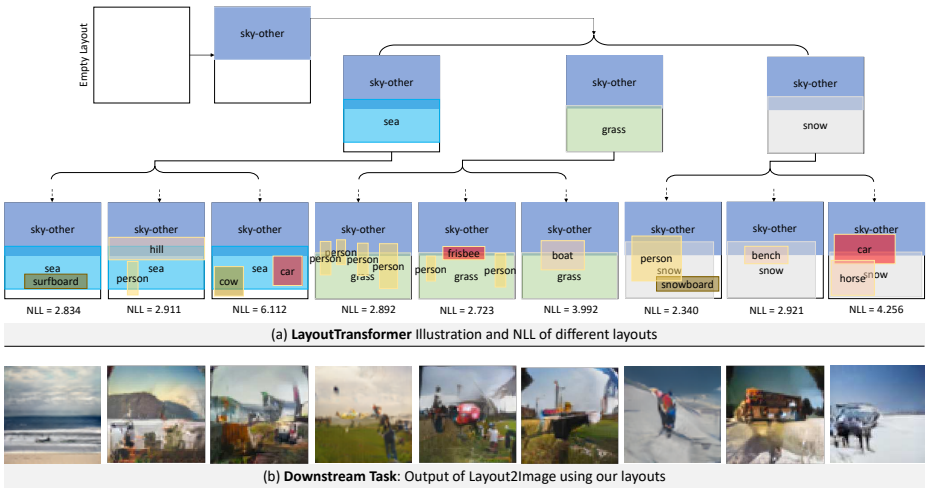


Fig. 1: (a) LayoutTransformer can generate multiple layouts consisting of variable number of elements starting from an empty canvas. (b) We can use tools such as Layout2Im [48] to generate image from layout (best viewed in color)

position, and size [2]). For generative models that synthesize scenes, this evidence underpins the importance of two factors that contribute to the *realism* or plausibility of a generated scene: layout, *i.e.*the arrangement of different objects, and their appearance (in terms of pixels). Therefore, generating a realistic scene necessitates both these factors to be plausible.

The advancements in the generative models for image synthesis have primarily targeted plausibility of the appearance signal by generating incredibly realistic images of objects (*e.g.*, faces [20,19], animals [3,46]). In order to generate complex scenes [6,15,17,26,33,47,48], most methods require proxy representations for layouts to be provided as inputs (*e.g.*scene segmentation [16,40], textual descriptions [26,47,34], scene graphs [17]). We argue that to plausibly generate large and complex scenes without such proxies, it is necessary to understand and generate the layout of the scene, in terms of contextual relationships between various objects present in the scene.

The layout of a scene, capturing what objects occupy what parts of the scene, is an incredibly rich representation. A plausible layout needs to follow prior knowledge of world regularities [2,7]; for example, the sky is above the sea, indoor scenes have certain furniture arrangements *etc.*. Learning to generate layouts is a challenging problem due to the variability of real-world layouts. Each scene contains a small fraction of possible objects, objects can be present in a wide range of locations, the number of objects varies for each scene and so do the contextual relationships between objects (*e.g.*, a person can carry a surfboard or ride a surfboard, a person can ride horse but not carry it). We parameterize each object or element of the layout by semantic attributes/categories, location, and size. In order to realize plausible semantic relationships, a generative model

for layouts should be able to look at all existing objects and propose placement of a new object.

We propose a sequential and iterative approach for modeling layouts that uses self-attention to look at existing layout elements. Our generative process can start from an empty set or an unordered set of elements already present in the scene, and can iteratively attend to existing elements to generate a new element. Moreover, by predicting either to stop or to generate the next element, our sequential approach can generate variable length layouts.

Our approach can be readily plugged into many scene generation frameworks (e.g., Layout2Image [48], GauGAN [32]). However, layout generation is not limited to scene layouts. Several stand-alone applications require generating layouts or templates. For instance, in the UI design of mobile apps and websites [8,28], an automated model for generating plausible layouts can significantly decrease the manual effort and cost of building such apps and websites. Finally, a model to create layouts can potentially help generate synthetic data for augmentation tasks [44,4] or 3D scenes [5,43,42].

We summarize the contributions of our work as follows:

- We develop a simple yet powerful auto-regressive model for generating layouts that can synthesize new layouts, complete partial layouts, and compute likelihood of existing layouts. Our self-attention approach allows us to visualize what existing elements are important for generating the next category in the sequence,
- We propose modeling position and size of layout elements with discrete multinomial distribution that enables our approach to generalize across very diverse data distributions,
- We present an exciting finding – encouraging a model to understand layouts results in feature representations that capture the semantic relationships between objects automatically (without explicitly using semantic embeddings, like word2vec [29]). This demonstrates the utility of the task of layout generation as a proxy-task for learning semantic representations,
- We show the performance and adaptability of our model on four layout datasets: MNIST Layout [25], Rico Mobile App Wireframes [8], PubLayNet Documents [45], and COCO Bounding Boxes [27]

2 Related Work

Generative models. Deep generative models in recent years have shown a great promise in terms of faithfully learning a given data distribution and sampling from it. Approaches such as variational auto-encoders [22] try to maximize approximate log-likelihood of data generated from a known distribution. Auto-regressive and flow-based approaches such as Pixel-RNN [31], RealNVP [9] can compute exact log-likelihood but are inefficient to sample from. GANs [11] are arguably the most popular generative models demonstrating state-of-the-art image generation results [3,20], but do not allow log-likelihood computation. Most

of these approaches and their variations work well when generating entire images, especially for datasets of images with a single entity or object. While these models can generate realistic objects, they often fail to capture global semantic and geometric relations between objects, which are needed to generate more complex realistic scenes [26].

Scene generation. Most works in 2D or 3D scene synthesis generate a scene conditioned on a sentence [26,47,34], a scene graph [17,24,1], a layout [10,13,16,41] or an existing image [23]. These involved pipelines are often trained and evaluated end-to-end, and surprisingly little work has been done to evaluate the layout generation component itself. Given the input, some works generate a fixed layout and diverse scenes [48], while other works generate diverse layouts and scenes [17,26]. Again the focus of *all* these works is on the quality of the final image and not the feasibility of the layout itself. In this work, we evaluate the layout modeling capabilities of two of the recent works [17,26] that have layout generation as an intermediate step.

Layout generation. Synthesising scene layouts is a relatively under-explored problem, mainly because generative models do not work well in practice when modeling sets of elements rather than images. LayoutGAN [25] attempts to solve the problem by starting with maximum number of possible elements in the scene and modifying their geometric and semantic properties. LayoutVAE [18] starts with a label set, *i.e.*, categories of all the elements present in the layout and then generates a feasible layout of the scene. Wang *et al.* [40,39] generate layout of an indoor room starting from top-down view of the room. However, their method is very specific to indoor rooms data and make assumptions about presence of walls, roof etc., and hence cannot be easily extended to other datasets. Zheng et al. [49] attempt to generate document layouts given the images, keywords and category of the document.

Our approach, **LayoutTransformer**, offers several advantages over current layout generation approaches without sacrificing their benefits. Unlike [25,49] autoregressive nature of model allows us to generate layouts of arbitrary lengths as well as start with partial layouts. We observe that modeling the position and size of layout elements as discrete values (as discussed in §3.1) helps us realize better performance on datasets, such as documents and app wireframes, where bounding boxes of layouts are typically axis aligned. Finally, assumptions of various kind of inputs limit the applicability of existing methods to diverse datasets. For example, to use scene graphs [17,24] as input, relationships need to be redefined for different datasets, text [26,47,34] need not exist for documents or wireframes, and to images can be used as input for documents but not for images. We get rid of lot of assumptions and simplify our layout generation pipeline to such an extent that it can be used to synthesize layouts for very diverse datasets. With thorough experimentation, we show the superiority of LayoutTransformer over three diverse real world datasets COCO Bounding Boxes [27], PubLayNet Documents [45], and Rico Mobile App Wireframes [8].

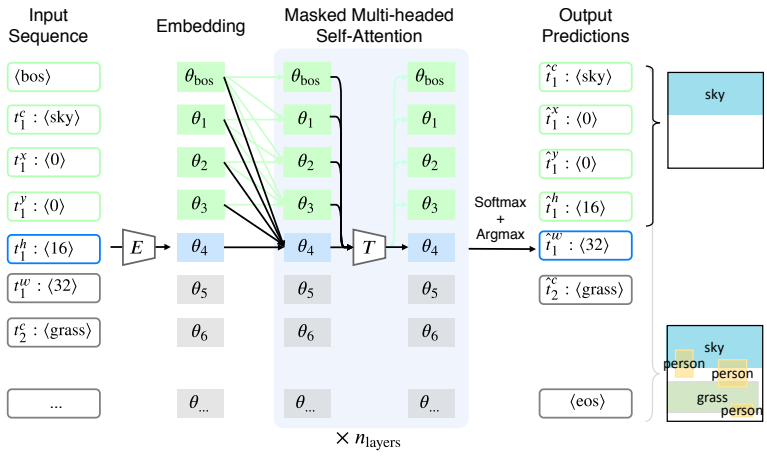


Fig. 2: The architecture for the Transformer model depicted for a toy example. It takes layout elements as input and predicts next layout elements as output. During training, we use teacher forcing, *i.e.*, use the groundtruth layout tokens as input to a multi-head decoder block. The first layer of this block is a masked self-attention layer, which allows the model to see only the previous elements in order to predict the current element. We pad each layout with a special ⟨bos⟩ token in the beginning and ⟨eos⟩ token in the end. To generate new layouts, we perform beam search starting with just the ⟨bos⟩ token or a partial sequence.

3 LayoutTransformer

Layouts are distinct from scenes or images in several ways. There is a strong non-local relationships between different elements in layouts. For example, presence of a small bird in corner of a scene changes distribution of objects present in rest of the scene, or a figure in document decides where the text can go. In case of images, on the other hand, local relationships play a more dominant role, *i.e.*, pixels close to each other are likely to be similar in values. While CNN based architectures such as VAE and GANs are excellent at generating pixels or images, an intuitive way to go about modeling distribution of complex scenes would be to use an auto-regressive model that can look at all existing elements, near or far, to generate semantic relationships followed by a convolutional architecture to generate the final image.

In this section, we propose LayoutTransformer, an auto-regressive self-attention network architecture to model the layouts. It allows us to learn non-local semantic relationships between layout elements and also give us flexibility to work with variable length layouts. We first describe the problem setup and follow it up with details of network architecture and training.

3.1 Problem Setup

We represent each layout as a sequence of graphical elements comprising the layout. For two-dimensional datasets such as documents, images, and wireframes,

each graphical element can be further represented by a bounding box of category $c \in C$, located at $(x, y) \in \mathbb{R}^2$, of size $(w, h) \in \mathbb{R}^2$. The goal of the layout model is now to learn the joint distribution of category, location, and size of layout elements represented by a tuple (c, x, y, w, h). We order all the elements in the raster scan order *i.e.*first by y coordinate, followed by x coordinate. After reordering, we concatenate all the graphical elements in a flat sequence. We also append two special symbols $t_{\langle \text{bos} \rangle}$ and $t_{\langle \text{eos} \rangle}$ to denote start and end of sequence. Hence, our layout of K graphical elements can be now represented as a sequence

$$t_{\text{seq}} : \{t_{\langle \text{bos} \rangle}, t_1^c, t_1^x, t_1^y, t_1^h, t_1^w, t_2^c \dots t_K^w, t_K^h, t_{\langle \text{eos} \rangle}\}$$

For simplicity, we use t_i to represent any element in the flattened sequence of the tuples, *i.e.*, $t_i \in t_{\text{seq}}$. We now pose the problem of modeling this joint distribution as product over series of conditional distributions using chain rule:

$$p(t_{1:K}) = \prod_{k=1}^{K} p(t_k | t_{1:k-1})$$

Representation of layout element. As introduced earlier, each layout element is represented by its category and the bounding box enclosing it. Instead of treating the location and size of bounding boxes as continuous variables, we model them as a discrete distribution where each $p(t)$ is obtained as the output of a softmax layer. Apart from allowing us to represent each of t^c, t^x, t^y, t^h, t^w in a simple and consistent manner, this strategy has the additional advantage that it does not assume a prior on position and size of bounding boxes and lets the network model them arbitrarily. If we divide our layout in $H \times W$ grid cells, each element of t_{seq} can be represented by a one-hot encoded vector of size $V = C + 2 \times (H + W)$, where we use V to denote the size of vocabulary

$$t_i \in \{\underbrace{1, \dots, C}_{\text{category}}, \underbrace{1, \dots, W}_{\text{x-coord.}}, \underbrace{1, \dots, H}_{\text{y-coord.}}, \underbrace{1, \dots, W}_{\text{width}}, \underbrace{1, \dots, H}_{\text{height}}\}$$

In Fig. 5, we show that discretizing position and shape in this way is particularly advantageous for datasets such as document layouts when bounding boxes are aligned to each other. We also discuss a variation of this strategy in ablation studies.

Ordering of elements. The sequence of elements is important in order to train our model. We use a simple raster scan order of layout elements in our case. We show the impact of removing this ordering strategy and using an arbitrary order in ablation studies.

3.2 Network Architecture and Training

We use a Transformer Decoder [38] to estimate the above joint probability distribution. Fig. 2 shows the network architecture for a toy example. Each layout element t_i gets mapped to a d-dimensional embedding such that $\theta_k = \theta(t_k) \in \mathbb{R}^d$.

These embeddings are then passed to a sequence of masked self-attention layers. The masking is done so that model predicts the probability of an element $p(t_i)$ using only the embeddings of previous layout elements. This means

$$p(t_k) = f_{\text{dec}}(\theta_{1:k-1})$$

where f_{dec} represent the masked multi-headed self-attention transformer decoder module. It includes a softmax layer in the end to normalize the output values between 0 and 1. Instead of using a standard cross-entropy loss, we follow the approach commonly used in Transformer-like architectures. For every groundtruth element in the sequence t, we create a pseudo groundtruth distribution $\hat{t}$ using Label Smoothing [30] with high confidence at the groundtruth index and rest of the mass distributed uniformly. We then minimize the KL-Divergence loss of predictions with this distribution using $l(t, \hat{t}) = \mathbb{KL}\left[p(t) \parallel p(\hat{t})\right]$. Label Smoothing impacts the perplexity of the model adversely but prevents the model from becoming overconfident.

In our base model, we use $d = 512$, $n_{\text{layers}} = 6$, and $n_{\text{heads}} = 8$ in the decoder. Label smoothing uses an $\epsilon = 0.1$. We observe that our model is quite robust to these choices, as we show in the ablation studies. The rest of the details of network architecture and training are in the supplementary material.

3.3 Sampling layouts

At training and validation time, we use teacher forcing, *i.e.*, since we know all the sequences, we use groundtruth sequences of variable lengths to train our model efficiently. To sample a new layout, we can start off with either just a start of sequence token t_{bos} or a set of tokens $(t_{\text{bos}}, t_1, \ldots, t_k)$. A naïve decoding strategy would be greedy decoding, *i.e.*, we predict the next element with the highest probability, append it to the existing sequence, and repeat till we reach the end of the sequence t_{eos}. A better way would be to do a beam search [36], *i.e.*, keep track of b most likely sequences while decoding to generate multiple possible layouts starting from the same initial sequence.

4 Experiments

In this section, we first discuss datasets used for evaluation, followed by qualitative results of our approach on these datasets. We then analyze the performance of LayoutTransformer on general and dataset-specific quantitative metrics.

4.1 Datasets

We evaluate the performance of our approach on multiple diverse datasets. Specifically, we use a toy MNIST Layout dataset as proposed in LayoutGAN [25], Rico Mobile App Wireframes [8,28], COCO bounding boxes [27] and PubLayNet Documents [45]. Note that each of the datasets involves a pre-processing step,

Fig. 3: Generated layouts. First column (in each dataset) shows a random layout from validation data rendered on a blank image. We use part of the sequence in this layout (from validation data) to generate the most probable layout using our model. Second column onwards, we show initial layout given to LayoutTransformer for completion followed by layout as completed by LayoutTransformer. We skip the label names in case of RICO dataset for clarity.

and we tried to faithfully replicate these steps as provided in original publications [25,18]. We will release the code for pre-processing, our approach, and experiments for reproducibility and future reference.

MNIST Layout. Following LayoutGAN [25], in all 28×28 images, we consider pixels with values greater than a fixed threshold (fixed to 64 in all experiments) as foreground pixels. Each image is now represented by a set of randomly selected foreground pixel indices with a minimum of 32 and a maximum of 128 indices per image. Overall, we get 59993 training and 10000 validation layouts from the original train/val split of MNIST. The mean and median length of layouts is 110.1 and 121, respectively.

COCO bounding boxes. COCO bounding boxes dataset is obtained using bounding box annotations in COCO Panoptic 2017 dataset [27]. We ignore the images where the *isCrowd* flag is true following the LayoutVAE [18] approach. The bounding boxes come from all 80 thing and 91 stuff categories. Our final dataset has 118280 layouts from COCO train split with a median length of 42 elements and 5000 layouts from COCO valid split with a median length of 33.

Rico Mobile App Wireframes. Rico mobile app dataset [8,28] consists of layout information of more than 66000 unique UI screens from over 9300 android apps. Each layout consists of one or more of the 25 categories of graphical elements such as text, image, icon *etc.* A complete list of these elements and frequency of their appearances is provided in the supplementary material. Overall, we get 62951 layouts in Rico with a median length of 36.

PubLayNet. PubLayNet [45] is a large scale document dataset recently released by IBM. It consists of over 1.1 million documents collected from PubMed Central. The layouts are annotated with 5 element categories - text, title, list, label, and figure. We filter out the document layouts with over 128 elements. Our final dataset has 335703 layouts from PubLayNet train split with a median length of 33 elements and 11245 layouts from PubLayNet dev split with a median length of 36.

4.2 Baseline Models

We consider following baseline approaches:

LayoutVAE. LayoutVAE [18] uses a similar representation for layout and consists of two separate autoregressive VAE models. Starting from a label set, which consists of categories of elements that will be present in a generated layout, their CountVAE generates counts of each of the elements of the label set. After that BoundingBoxVAE, generates the location and size of each occurrence of the bounding box.

ObjGAN. ObjGAN [26] provides an object-attention based GAN framework for text to image synthesis. An intermediate step in their image synthesis approach is to generate a bounding box layout given a sentence using a BiLSTM (trained independently). We adopt this step of the ObjGAN framework to our problem setup. Instead of sentences we provide categories of all layout elements as input to the ObjGAN and synthesize all the elements' bounding boxes.

sg2im. Image generation from scene graph [17] attempts to generate complex scenes given scene graph of the image by first generating a layout of the scene using graph convolutions and then using the layout to generate complete scene using GANs. The system is trained in an end-to-end fashion. Since sg2im requires a scene graph input, following the approach of [17], we create a scene graph from the input and reproduce the input layout using the scene graph.

4.3 Qualitative Evaluation

Generated layout samples. Figure 3 shows some of the generated samples of our model from different datasets. Note that our model can generate samples from empty or partial layouts. We demonstrate this by taking partial layouts from validation data and generating a full layout with greedy decoding.

Nearest neighbors. To see if our model is memorizing the training dataset, we compute nearest neighbors of generated layouts using chamfer distance on

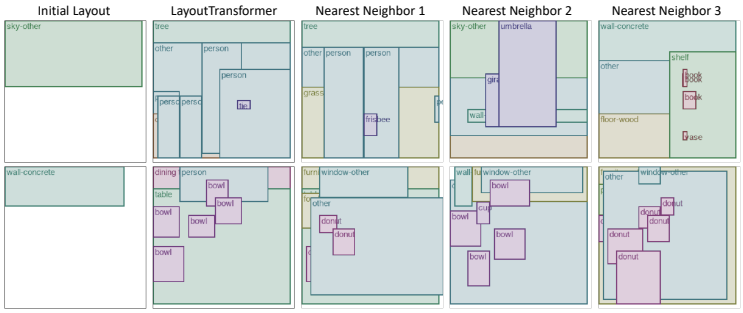


Fig. 4: Nearest neighbors from training data. Column 1 shows the initial layout provided (for completion) to LayoutTransformer. Column 2 shows the layout as generated by LayoutTransformer. Column 3, 4 and 5 show the 3 closest neighbors from training dataset. We use chamfer distance on bounding box coordinates to obtain the nearest neighbors from the dataset.

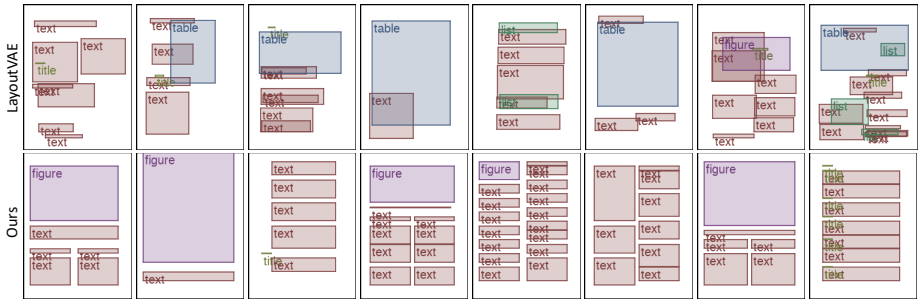


Fig. 5: Generated samples for PubLayNet dataset using LayoutVAE [18] and our method. Our method produces aligned bounding boxes for various synthesized layout elements such as figure, text, title and tables.

top-left and bottom-right bounding box coordinates of layout elements. Figure 4 shows the nearest neighbors of some of the generated layouts from the training dataset. Our model is able to generate novel layouts not present in the training data.

Visualizing attention. The self-attention based approach proposed enables us to visualize which existing elements are being attending to while the model is generating a new element. This is demonstrated in Figure 6.

4.4 Semantics Emerge via Layout

We posited earlier that capturing layout should capture contextual relationships between various elements. We provide further evidence of our argument in three ways. We visualize the 2D-tsne plot of the learned embeddings for categories, as shown in Figure 7. We observe that super-categories from COCO are clustered together in the embedding space of the model. Certain categories such as window-blind and curtain (which belong to different super-categories) also appear close to

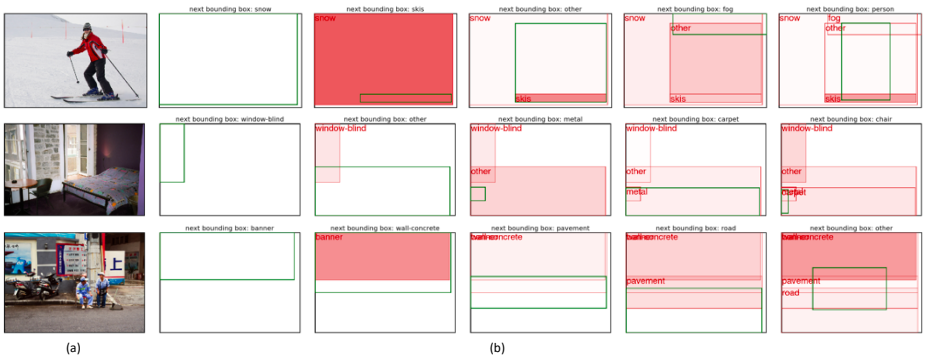


Fig. 6: Visualizing attention. (a) Image source for the layout (b) In each row, the model is predicting one element at a time (shown in a green bounding box). While predicting that element, the model pays the most attention to previously predicted bounding boxes (in red). For example, in the first row, "snow" gets the highest attention score while predicting "skis". Similarly in the last column, "skis" get the highest attention while predicting "person".

Table 1: **Bigrams and trigrams**. We consider the most frequent pairs and triplets of (distinct) categories in real *vs.*generated layouts.

Real	Ours	Real	Ours
other person	other person	person other person	other person clothes
person other	person clothes	other person clothes	person clothes tie
person clothes	clothes tie	person handbag person	tree grass other
clothes person	grass other	person clothes person	grass other person
chair person	other dining table	person chair person	wall-concrete other person
person chair	tree grass	chair person chair	grass other cow
sky-other tree	wall-concrete other	person other clothes	tree other person
car person	person other	person backpack person	person clothes person
person handbag	sky-other tree	person car person	other dining table table
handbag person	clothes person	person skis person	person other person

Table 2: **Analogies**. We demonstrate linguistic nuances being captured by our category embeddings by attempting to solve word2vec [29] style analogies.

Analogy	Nearest neighbors
snowboard:snow::surfboard:?	waterdrops, sea, sand
car:road::train:?	railroad, platform, gravel
sky-other:clouds::playingfield:?	net, cage, wall-panel
mouse:keyboard::spoon:?	knife, fork, oven
fruit:table::flower:?	potted plant, mirror-stuff

each other. Table 1 captures the most frequent bigrams and trigrams (categories that co-occur) in real and synthesized layouts. Table 2 shows word2vec [29] style analogies being captured by embeddings learned by our model. Note that the model was trained to generate layouts and we did not specify any additional objective function for analogical reasoning task. These observations are in line with observations of Gupta et al [12].

4.5 Quantitative Evaluation

Quantitative evaluation methods of the layout generation problem differ from dataset to dataset. In this section, we discuss some of these methods applicable to the datasets under consideration.

Downstream Task - Image generation. To evaluate the ability of layout generation approaches in generating plausible layouts for the COCO dataset we use Layout2Im [48] to generate an image from a layout. Table 3 shows images generated from layouts by our method and LayoutVAE. We also compute compute

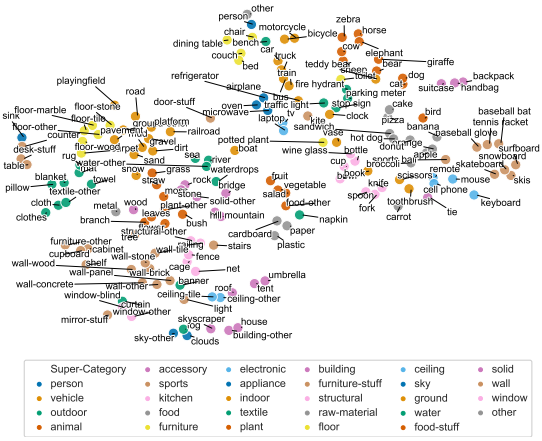


Fig. 7: Word embeddings as learned by LayoutTransformer on COCO bounding boxes. Words are colored by their super-categories provided in the COCO dataset. We see that semantically similar categories cluster together, *e.g.*, animals, fruits, furniture, etc. Cats and dogs are closer to each other compared to sheep, zebra, or cow. Also, semantically similar words from different super-categories (such as curtain, window-blind, and mirror) are close in LayoutTransformer's embedding.

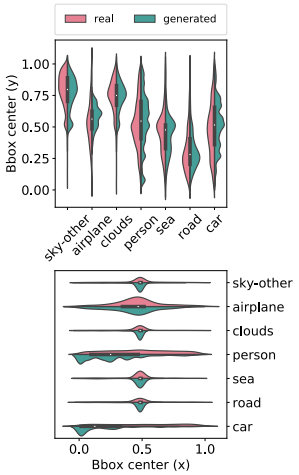
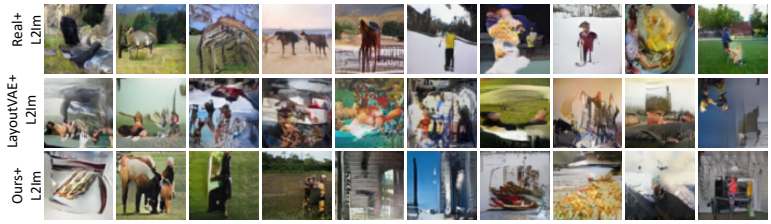


Fig. 8: We plot the distribution of x- and y-coordinates of the center of bounding boxes (normalized between 0 and 1). The y-coordinate is more informative (*e.g.*, sky is usually on the top of the image while road and sea are at the bottom). Distributions for generated layouts and real layouts tend to be similar.

Inception Score (IS) and Frchet Inception Distance (FID) to compare quality and diversity of generated images. Our method improves upon LayoutVAE in both the metrics.

Negative log-likelihood. For each of the datasets, Table 4 shows the negative log-likelihood of all the layouts in validation set. The results indicate that our approach generates more plausible layouts (more details on this are provided in the supplementary material).

Dataset statistics. Depending on the dataset and definition of graphical elements, we can define statistics that layouts should follow. For Rico wireframes and PubLayNet docs, we compare two important statistics of layouts in Table 5. **Overlap** represents the intersection over union (IoU) of various layout elements. Generally in these datasets, elements do not overap with each other and Overlap is small. **Coverage** indicates the percentage of canvas covered by the layout elements. The table shows that layouts generated by our method resemble real data statistics better than LayoutGAN and LayoutVAE.



Method	IS
Real	16.1
[18] + [48]	7.1
Ours + [48]	**7.5**

Method	FID
[18] + [48]	64.1
Ours + [48]	**57**

Table 3: Image generation from layouts - We use L2Im [48] to convert layouts to images. (Left) The first row shows images using layouts from validation data. The second row shows generated novel layouts converted to image using the same model. The third row shows layouts generated by LayoutVAE converted to image in a similar manner. (Right) We compute Inception Score (IS) and Frchet Inception Distance (FID) to compare quality and diversity of generated images from our layouts as compared to real layouts.

4.6 Ablation studies

We evaluate the importance of different model components with negative log-likelihood on COCO layouts. The ablation studies clearly show the following:

Small, medium and large elements: NLL of our model for COCO large, medium, and small boxes is 2.4, 2.5, and 1.8 respectively. We observe that even though discretizing box coordinates introduces approximation errors, it later allows our model to be agnostic to large vs small objects.

Varying n_{anchor}: n_{anchor} decides the resolution of the layout. Increasing it allows us to generate finer layouts but at the expense of a model with more parameters. Also, as we increase the n_{anchor}, NLL increases, suggesting that we might need to train the model with more data to get similar performance (Table 6).

Size of embedding: Increasing the size of the embedding d improves the NLL, but at the cost of increased number of parameters (Table 7).

Model depth: Increasing the depth of the model n_{layers}, does not significantly improve the results (Table 8). We fix the $n_{\text{layers}} = 6$ in all our experiments.

Ordering of the elements: The self-attention layer in our model is invariant to the ordering of elements. Therefore, while predicting the next element of the layout, we do not consider the order in which the elements were added. However, in our experiments, we observed that predicting the elements in a simple raster scan order of their position improves the model performance both visually and in terms of negative log-likelihood. This is intuitive as filling the elements in a pre-defined order is an easier problem. We leave the task of optimal ordering of layout elements to generate layouts for future research (Table 9).

Discretization strategy: Instead of the next bounding box, we tried predicting the x-coordinate and y-coordinate of the bounding box together (refer to the Split-xy column of Table 9). This increases the vocabulary size of the model (since we use $H \times W$ possible locations instead of $H + W$) and in turn the

Table 4: Negative log-likelihood of all the layouts in validation set (lower the better). For each of the approach we compute log-likelihood using importance sampling as described in [18]. For LayoutVAE, we use our own implementation (since official implementation is not provided

Model	COCO	Rico	PubLayNet
sg2im [17]	6.24	7.43	7.12
ObjGAN [26]	5.24	4.21	4.20
LayoutVAE [18]	3.29	2.54	2.45
Ours	**2.28**	**1.07**	**1.10**

Table 5: Spatial distribution analysis for the RICO dataset. As we limit the resolution for location and size of the bounding boxes to 32×32, our model is to be compared to the lower resolution version of the real data. Closer the values to real data, better is the performance. Clearly, the statistics of our layouts are the more similar to the real data statistics than sg2im, ObjGAN and LayoutVAE. All values in the table are percentages (std in parenthesis)

	Rico		PubLayNet	
Methods	Coverage	Overlap	Coverage	Overlap.
sg2im [17]	25.2 (46)	16.5 (31)	30.2 (26)	3.4 (12)
ObjGAN [26]	39.2 (33)	36.4 (29)	38.9 (12)	8.2 (7)
LayoutVAE [18]	41.5 (29)	34.1 (27)	40.1 (11)	14.5 (11)
Ours	33.6 (27)	23.7 (33)	47.0 (12)	0.13 (1.5)
Real Data (32×32)	30.2 (25)	20.5 (30)	47.8 (9)	0.02 (0.5)
Real Data	36.6 (27)	22.4 (32)	57.1 (10)	0.1 (0.6)

Table 6: Effect of n_{anchors} on NLL

n_{anchors}	# params	COCO	Rico	PubLayNet
32×32	19.2	2.28	1.07	1.10
8×8	19.1	1.69	0.98	0.88
16×16	19.2	1.97	1.03	0.95
64×64	19.3	2.67	1.26	1.28
128×128	19.6	3.12	1.44	1.46

Table 7: Effect of d on NLL

d	# params	COCO	Rico	PubLayNet
512	19.2	2.28	1.07	1.10
32	0.8	2.51	1.56	1.26
64	1.7	2.43	1.40	1.19
128	3.6	2.37	1.29	1.57
256	8.1	2.32	1.20	1.56

Table 8: Effect of n_{layers} on NLL

n_{layers}	# params	COCO	Rico	PubLayNet
6	19.2	2.28	1.07	1.10
2	6.6	2.31	1.18	1.13
4	12.9	2.30	1.12	1.07
8	25.5	2.28	1.11	1.07

Table 9: Effect of other hyperparameters on NLL

Order	Split-XY	Loss	# params	COCO	Rico	PubLayNet
raster	Yes	NLL	19.2	2.28	1.07	1.10
random			19.2	2.68	1.76	1.46
	No		21.2	3.74	2.12	1.87
		LS	19.2	1.96	0.88	0.88

number of hyper-parameters with decline in model performance. An upside of this approach is that generating new layouts takes less time as we have to make half as many predictions for each element of the layout (Table 9).

Loss: We tried two different losses, label smoothing [30] and NLL. Although optimizing using NLL gives better validation performance in terms of NLL (as is expected), we do not find much difference in the qualitative performance when using either loss function. (Table 9)

5 Conclusion

We propose LayoutTransformer, a novel approach to generate layouts of graphical elements. Our model uses self-attention model to capture contextual rela-

tionship between different layout elements and generate novel layouts. We show that our model is better than previously proposed models for layout generation due to its ability to synthesize layouts from an empty set or complete a partial layout. The model can also produce layouts with a variable number of elements and categories. We show that our model can produce qualitatively better layouts than the state-of-the-art approaches for diverse datasets such as Rico Mobile App Wireframes, COCO bounding boxes, and PubLayNet documents. While we demonstrated results for our approach by generating layouts in two dimensions, this framework is applicable to three dimensional scenes as well. One limitation of the model is that while it is capable of predicting size and location of objects of the scene, it cannot be easily extended to predict object masks. We will explore these directions in future work.

Acknowledgements. We thank Yuting Zhang, Luis Goncalves, Stefano Soatto and Guha Balakrishnan for many helpful discussions. This work was partially supported by DARPA via ARO contract number W911NF2020009.

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. Biederman, I.: On the semantics of a glance at a scene. In: Perceptual organization, pp. 213–253. Routledge (2017)
3. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096 (2018)
4. Capobianco, S., Marinai, S.: Docemul: a toolkit to generate structured historical documents. CoRR **abs/1710.03474** (2017), `http://arxiv.org/abs/1710.03474`
5. Chang, A.X., Monroe, W., Savva, M., Potts, C., Manning, C.D.: Text to 3d scene generation with rich lexical grounding. CoRR **abs/1505.06289** (2015), `http://arxiv.org/abs/1505.06289`
6. Chen, Q., Koltun, V.: Photographic image synthesis with cascaded refinement networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1511–1520 (2017)
7. Chen, X., Shrivastava, A., Gupta, A.: Neil: Extracting visual knowledge from web data. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1409–1416 (2013)
8. Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: Proceedings of the 30th Annual Symposium on User Interface Software and Technology. UIST '17 (2017)
9. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using real nvp. arXiv preprint arXiv:1605.08803 (2016)
10. Dong, H., Yu, S., Wu, C., Guo, Y.: Semantic image synthesis via adversarial learning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 5706–5714 (2017)
11. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)

12. Gupta, T., Schwing, A., Hoiem, D.: Vico: Word embeddings from visual co-occurrences. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 7425–7434 (2019)
13. Hinz, T., Heinrich, S., Wermter, S.: Generating multiple objects at spatially distinct locations. CoRR **abs/1901.00686** (2019), http://arxiv.org/abs/1901.00686
14. Holtzman, A., Buys, J., Forbes, M., Choi, Y.: The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751 (2019)
15. Hong, S., Yang, D., Choi, J., Lee, H.: Inferring semantic layout for hierarchical text-to-image synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7986–7994 (2018)
16. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. arxiv (2016)
17. Johnson, J., Gupta, A., Fei-Fei, L.: Image generation from scene graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1219–1228 (2018)
18. Jyothi, A.A., Durand, T., He, J., Sigal, L., Mori, G.: Layoutvae: Stochastic scene layout generation from a label set. arXiv preprint arXiv:1907.10719 (2019)
19. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
20. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4401–4410 (2019)
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
22. Kingma, D.P., Welling, M.: Auto-encoding variational bayes (2013)
23. Lee, D., Liu, S., Gu, J., Liu, M., Yang, M., Kautz, J.: Context-aware synthesis and placement of object instances. CoRR **abs/1812.02350** (2018), http://arxiv.org/abs/1812.02350
24. Li, B., Zhuang, B., Li, M., Gu, J.: Seq-sg2sl: Inferring semantic layout from scene graph through sequence to sequence learning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 7435–7443 (2019)
25. Li, J., Yang, J., Hertzmann, A., Zhang, J., Xu, T.: Layoutgan: Generating graphic layouts with wireframe discriminators. arXiv preprint arXiv:1901.06767 (2019)
26. Li, W., Zhang, P., Zhang, L., Huang, Q., He, X., Lyu, S., Gao, J.: Object-driven text-to-image synthesis via adversarial training. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 12174–12182 (2019)
27. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
28. Liu, T.F., Craft, M., Situ, J., Yumer, E., Mech, R., Kumar, R.: Learning design semantics for mobile apps. In: The 31st Annual ACM Symposium on User Interface Software and Technology. pp. 569–579. UIST '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3242587.3242650, http://doi.acm.org/10.1145/3242587.3242650
29. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
30. Müller, R., Kornblith, S., Hinton, G.E.: When does label smoothing help? CoRR **abs/1906.02629** (2019), http://arxiv.org/abs/1906.02629
31. van den Oord, A., Kalchbrenner, N.: Pixel rnn (2016)

32. Park, T., Liu, M.Y., Wang, T.C., Zhu, J.Y.: Gaugan: semantic image synthesis with spatially adaptive normalization. In: ACM SIGGRAPH 2019 Real-Time Live! p. 2. ACM (2019)
33. 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)
34. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396 (2016)
35. Shrivastava, A., Gupta, A.: Contextual priming and feedback for faster r-cnn. In: European Conference on Computer Vision. pp. 330–348. Springer (2016)
36. Steinbiss, V., Tran, B.H., Ney, H.: Improvements in beam search. In: Third International Conference on Spoken Language Processing (1994)
37. Torralba, A., Sinha, P.: Statistical context priming for object detection. In: Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001. vol. 1, pp. 763–770. IEEE (2001)
38. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
39. Wang, K., Lin, Y.A., Weissmann, B., Savva, M., Chang, A.X., Ritchie, D.: Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. ACM Transactions on Graphics (TOG) **38**(4), 1–15 (2019)
40. Wang, K., Savva, M., Chang, A.X., Ritchie, D.: Deep convolutional priors for indoor scene synthesis. ACM Transactions on Graphics (TOG) **37**(4), 70 (2018)
41. 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: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018)
42. Wu, J., Lu, E., Kohli, P., Freeman, W.T., Tenenbaum, J.B.: Learning to see physics via visual de-animation. In: Advances in Neural Information Processing Systems (2017)
43. Wu, J., Tenenbaum, J.B., Kohli, P.: Neural scene de-rendering. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
44. Yang, X., Yümer, M.E., Asente, P., Kraley, M., Kifer, D., Giles, C.L.: Learning to extract semantic structure from documents using multimodal fully convolutional neural network. CoRR **abs/1706.02337** (2017), `http://arxiv.org/abs/1706.02337`
45. Yepes, A.J., Tang, J., Zhong, X.: Publaynet: largest dataset ever for document layout analysis
46. Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-attention generative adversarial networks. arXiv preprint arXiv:1805.08318 (2018)
47. Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., Metaxas, D.N.: Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 5907–5915 (2017)
48. Zhao, B., Meng, L., Yin, W., Sigal, L.: Image generation from layout. In: CVPR (2019)
49. Zheng, X., Qiao, X., Cao, Y., Lau, R.W.: Content-aware generative modeling of graphic design layouts. ACM Transactions on Graphics (TOG) **38**(4), 1–15 (2019)

Appendix

A Architecture and training details

In all our experiments, our base model consists of $n_{\text{anchors}} = 32 \times 32$, $d = 512$, $n_{\text{layers}} = 6$, $n_{\text{heads}} = 8$ and $d_{\text{ff}} = 2048$ where n_{anchors} is the number of grid cells (corresponding to locations and size of bounding boxes), d is the dimension of embedding in each of the layers, n_{layers} is the number of self-attention layers, n_{heads} is the number of heads in each of the self-attention layer, and d_{ff} is the number of units in feedforward layer that follows the self-attention part in a single decoder block. We also use a dropout of 0.1 at the end of each feedforward layer for regularization. Starting with start token t_{bos}, our model predicts category, location and shape of next bounding box in a raster scan order. We fix the the maximum number of elements in each of the datasets to 128 which covers over 99.9% of the layouts in each of the COCO, Rico and PubLayNet datasets.

We also used Adam optimizer [21] with Noam learning rate scheduling. We train our model for 20 epochs for each dataset with early stopping based on maximum log likelihood on validation layouts (overall we trained our model for 8 epochs on COCO, 12 epochs on Rico, and 16 epochs on PubLayNet). Our COCO Bounding Boxes model takes about 2 hours to train on a single NVIDIA GTX1080 GPU. Batching matters a lot to improve the training speed. We want to have evenly divided batches, with absolutely minimal padding. We sort the layouts by the number of elements and search over this sorted list to use find tight batches for training.

B Evaluation Details

In this section, we provide more details on various qualitative and qualitative evaluation done in Section 4. We also provide some additional results (that didn't fit in the paper).

B.1 Generated samples

We show random samples generated for each of the dataset using different methods listed in Section 4.2. In case of LayoutVAE [18], we use label set of layouts in validation dataset as input to generate samples. In case of LayoutTransformer, we take one layout element from validation dataset and complete the layout using our model.

B.2 Computing nearest neighbors

While there is no standard method for comparing two layouts, in Section 4.3 of the paper, we use modified Chamfer distance[1] in order to compare two layouts

[1] https://github.com/ThibaultGROUEIX/AtlasNet

(with different number of elements). This metric doesn't take into account categories of various layout elements and just compute euclidean distance between bounding box top left x, y coordinates and height and width.

B.3 Computing log-likelihood

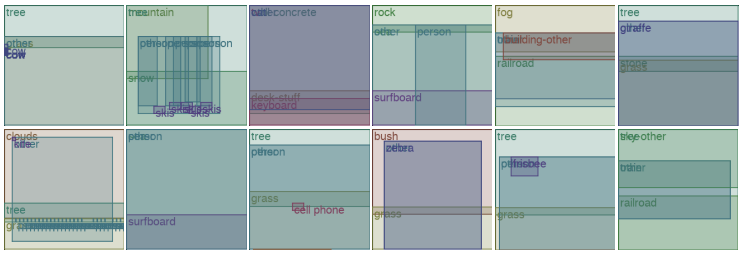
In order to compute NLL for layouts generated by LayoutVAE, we follow the approach as provided by the authors in their paper. Using teacher forcing, for a layout in validation set we compute Monte Carlo estimate of NLL by drawing 1000 samples from conditional prior. We add the NLL for CountVAE and BBoxVAE.

B.4 Layout Verification

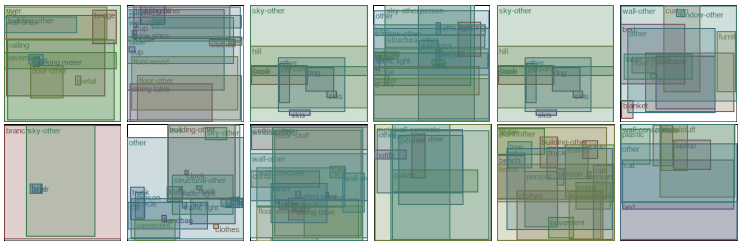
Since in our method it is straightforward to compute likelihood of a layout, we can use our method to test if a given layout is likely or not. Figure 12 shows the NLL given by our model by doing left-right and top-down inversion of layouts in COCO (following [25]). In case of COCO, if we flip a layout left-right, we observe that layout remains likely, however flipping the layout upside decreases the likelihood (or increases the NLL of the layout). This is intuitive since it is unlikely to see fog in the bottom of an image, while skis on top of a person.

B.5 LayoutGAN

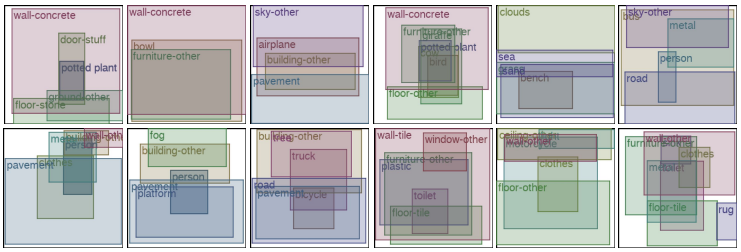
LayoutGAN [25] represents each layout with a fixed number of bounding boxes. Starting with bounding box coordinates sampled from a Gaussian distribution, its GAN based framework assigns new coordinates to each bounding box to resemble the layouts from given data. Optionally, it uses non-maximum suppression (NMS) to remove duplicates. The problem setup in LayoutGAN is similar to the proposed approach and they do not condition the generated layout on anything. Like many GAN setups, LayoutGAN is non-trivial to train across multiple datasets. In our implementation of LayoutGAN, we were unable to prevent mode collapse for all datasets except for MNIST.



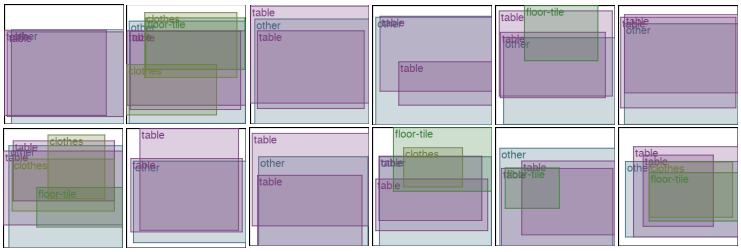
(a) Ours



(b) LayoutVAE [18]

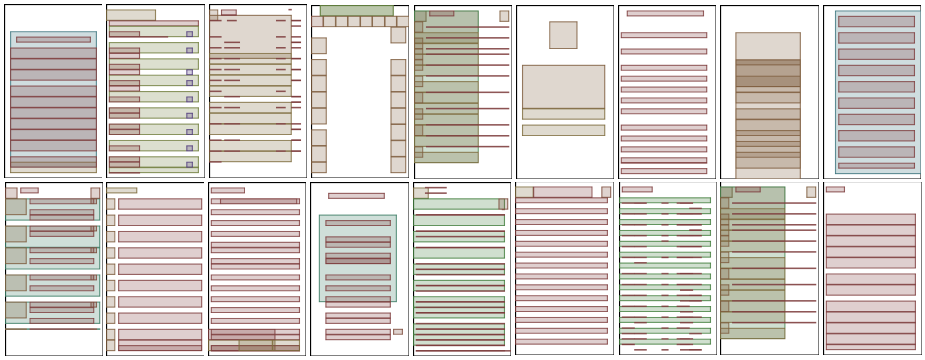


(c) SG2IM [17]

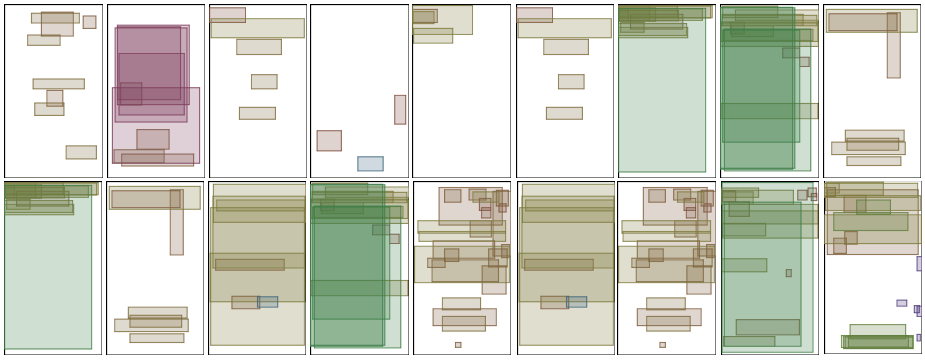


(d) Obj-GAN [26]

Fig. 9: Layout samples of COCO bounding boxes. LayoutVAE is constrained to start with a fixed set of categories. For the set of categories, which rarely occur together, LayoutVAE might have hard time finding appropriate placement for them. For example, in second layout of bottom row (of LayoutVAE), clock and traffic lights are put together which are a bit unlikely. Our method doesn't face this problem. One artifact of our method is since it generates most likely layouts first (using greedy decoding), it might not end up using some less common objects (although one work around of this problem by using penalized beam search or nucleus sampling [14]). Another point to be noted is both LayoutVAE and SG2IM needs as input a set of categories to be placed in the layout. For above samples, we use validation data to get the input sets.

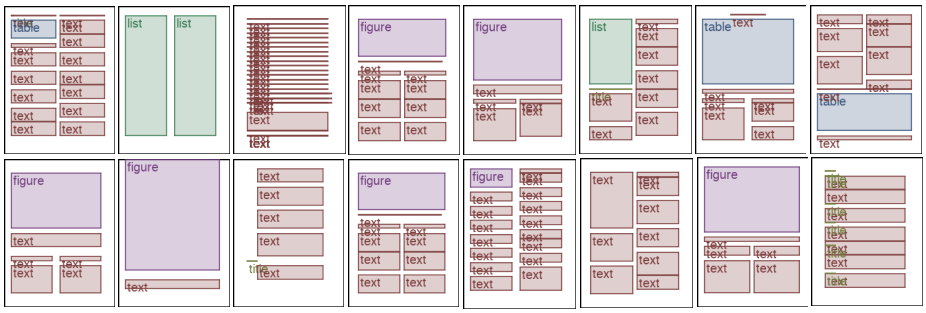


(a) Ours

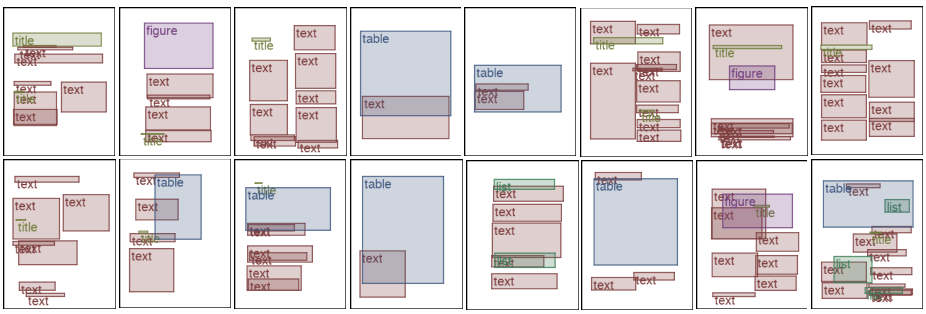


(b) LayoutVAE [18]

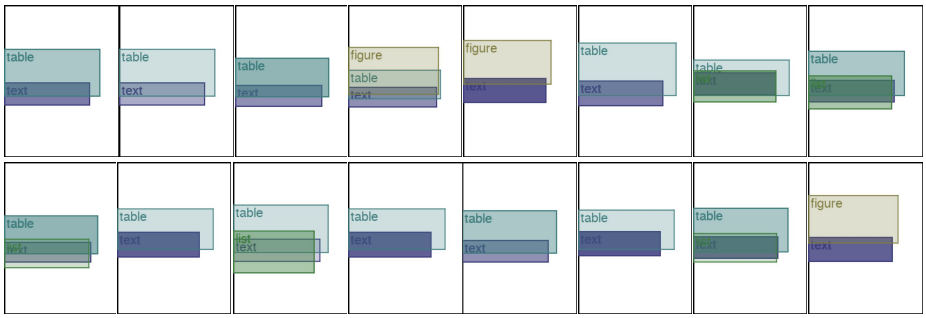
Fig. 10: Layout samples of Rico. LayoutVAE's bounding boxes are not aligned with each other as is the case with real samples (and our samples)



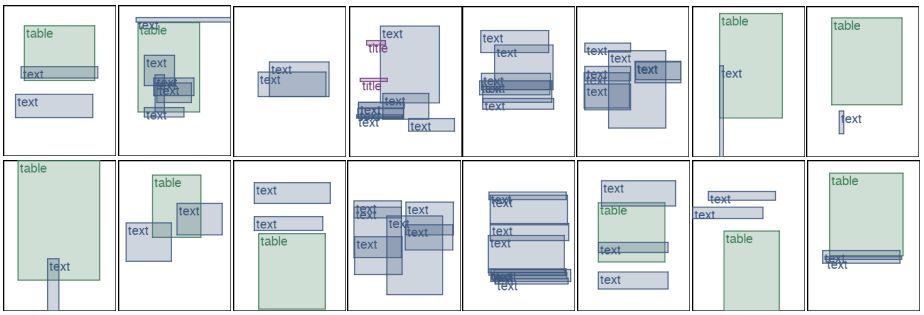
(a) Ours



(b) LayoutVAE [18]



(c) SG2IM [17]



(d) Obj-GAN [26]

Fig. 11: Layout samples of PubLayNet. ObjGAN, SG2IM and LayoutVAE's bounding boxes are not aligned with each other as is the case with real samples (and our samples)

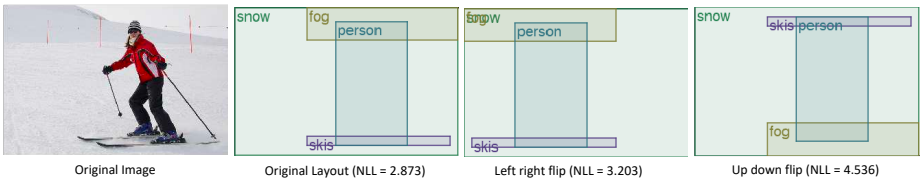
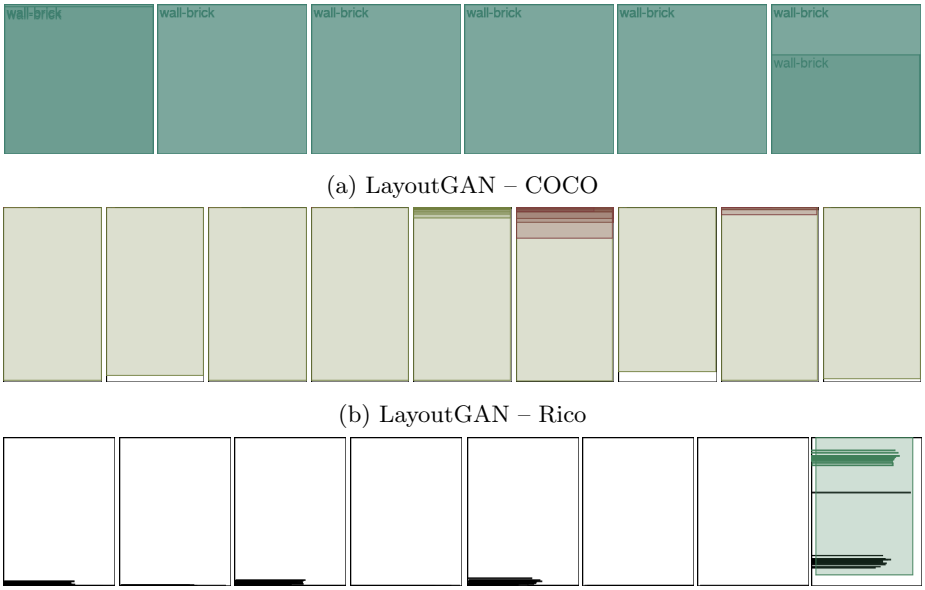


Fig. 12: We observe the impact of operations such as left right flip, and up down flip on log likelihood of the layout. We observe that unlikely layouts (such as fog at the bottom of image have higher NLL than the layouts from data.



(a) LayoutGAN – COCO

(b) LayoutGAN – Rico

(c) LayoutGAN – PubLayNet

Fig. 13: Layout samples using LayoutGAN. We were unable to prevent mode collapse for all datasets except for MNIST.

C Dataset Statistics

In this section, we share statistics of different elements and their categories in our dataset. In particular, we share the total number of occurrences of an element in the training dataset (in descending order) and the total number of distinct layouts an element was present in throughout the training data. Table 12 show these statistics for COCO bounding boxes, Tables 10, 10 show the statistics for Rico wireframes, and table 11 show the statistics for PubLayNet documents.

Table 10: Category statistics for Rico

Category	# occurrences	# layouts	Category	# occurrences	# layouts
Text	387457	50322	Modal	3248	3248
Image	179956	38958	Pager Indicator	2041	1528
Icon	160817	43380	Slider	1619	954
Text Button	118480	33908	On/Off Switch	1260	683
List Item	72255	9620	Button Bar	577	577
Input	18514	8532	Toolbar	444	395
Card	12873	3775	Number Stepper	369	147
Web View	10782	5808	Multi-Tab	284	275
Radio Button	4890	1258	Date Picker	230	217
Drawer	4138	4136	Map View	186	94
Checkbox	3734	1126	Video	168	144
Advertisement	3695	3365	Bottom Navigation	75	27

Table 11: Category statistics for PubLayNet

Category	# occurrences	# layouts
text	2343356	334548
title	627125	255731
figure	109292	91968
table	102514	86460
list	80759	53049

Table 12: Category statistics for COCO

Category	# occurrences	# layouts	Category	# occurrences	# layouts	Category	# occurrences	# layouts
person	257253	64115	floor-tile	6618	6618	flower	3259	3259
other	117266	117266	sea	6598	6598	leaves	3169	3169
car	43533	12251	vase	6577	3593	cloth	3129	3129
chair	38073	12774	horse	6567	2941	structural-other	3016	3016
tree	36466	36466	house	6549	6549	cage	2911	2911
sky-other	31808	31808	tie	6448	3810	desk-stuff	2909	2909
wall-concrete	31481	31481	cell phone	6422	4803	hot dog	2884	1222
clothes	27657	27657	floor-wood	6324	6324	keyboard	2854	2115
book	24077	5332	clock	6320	4659	branch	2813	2813
bottle	24070	8501	orange	6302	1699	railroad	2720	2720
building-other	23021	23021	sports ball	6299	4262	rug	2703	2703
grass	22575	22575	cake	6296	2925	frisbee	2681	2184
metal	22526	22526	ground-other	6252	6252	snowboard	2681	1654
cup	20574	9189	spoon	6159	3529	stairs	2667	2667
wall-other	19095	19095	suitcase	6112	2402	fog	2659	2659
pavement	18311	18311	surfboard	6095	3486	refrigerator	2634	2360
furniture-other	17882	17882	bus	6061	3952	gravel	2613	2613
table	16282	16282	pizza	5807	3166	blanket	2598	2598
dining table	15695	11837	tv	5803	4561	towel	2558	2558
road	15402	15402	couch	5779	4423	hill	2498	2498
bowl	14323	7111	apple	5776	1586	water-other	2453	2453
window-other	14209	14209	remote	5700	3076	wall-panel	2357	2357
textile-other	13052	13052	sink	5609	4678	river	2313	2313
traffic light	12842	4139	skateboard	5536	3476	window-blind	2297	2297
handbag	12342	6841	dog	5500	4385	mouse	2261	1876
light	11772	11772	elephant	5484	2143	fruit	2112	2112
fence	11303	11303	fork	5474	3555	railing	2068	2068
umbrella	11265	3968	wall-tile	5290	5290	wall-stone	2020	2020
plastic	11137	11137	zebra	5269	1916	vegetable	2016	2016
boat	10576	3025	playingfield	5251	5251	platform	2009	2009
ceiling-other	10546	10546	wall-brick	5246	5246	skyscraper	1998	1998
bird	10542	3237	airplane	5129	2986	pillow	1986	1986
dirt	10163	10163	giraffe	5128	2546	stop sign	1983	1734
truck	9970	6127	snow	5114	5114	toothbrush	1945	1007
clouds	9886	9886	curtain	5101	5101	fire hydrant	1865	1711
bush	9849	9849	wood	5053	5053	stone	1828	1828
bench	9820	5570	laptop	4960	3524	bridge	1676	1676
plant-other	9522	9522	mountain	4887	4887	microwave	1672	1547
paper	9521	9521	carpet	4858	4858	tent	1486	1486
door-stuff	9475	9475	tennis racket	4807	3394	scissors	1464	947
sheep	9223	1529	cat	4766	4114	napkin	1405	1405
banana	9195	2243	teddy bear	4729	2140	straw	1385	1385
floor-other	8893	8893	sand	4688	4688	net	1362	1362
kite	8802	2261	counter	4589	4589	bear	1294	960
backpack	8714	5528	shelf	4589	4589	parking meter	1283	705
motorcycle	8654	3502	train	4570	3588	floor-stone	1259	1259
potted plant	8631	4452	roof	4490	4490	cupboard	1004	1004
cow	8014	1968	sandwich	4356	2365	floor-marble	1002	1002
wine glass	7839	2533	bed	4192	3682	solid-other	749	749
knife	7760	4326	toilet	4149	3353	mud	659	659
carrot	7758	1683	banner	4135	4135	mat	559	559
broccoli	7261	1939	cardboard	3787	3787	salad	477	477
cabinet	7176	7176	baseball glove	3747	2629	ceiling-tile	351	351
bicycle	7056	3252	mirror-stuff	3622	3622	moss	256	256
donut	7005	1523	rock	3397	3397	toaster	225	217
food-other	6672	6672	oven	3334	2877	hair drier	198	189
wall-wood	6642	6642	baseball bat	3273	2506	waterdrops	121	121
skis	6623	3082						