Layout Generation and Completion with Self-attention

Kamal Gupta
University of Maryland College Park
kampta@cs.umd.edu

Alessandro Achille
Amazon AWS
aachille@amazon.com

Justin Lazarow
Amazon AWS
jlazarow@ucsd.edu

Larry Davis
University of Maryland College Park
lsd@umiacs.umd.edu

Vijay Mahadevan
Amazon AWS
vmahad@amazon.com

Abhinav Shrivastava
University of Maryland College Park
abhinav@cs.umd.edu

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 the understanding of 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 important 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 what 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).

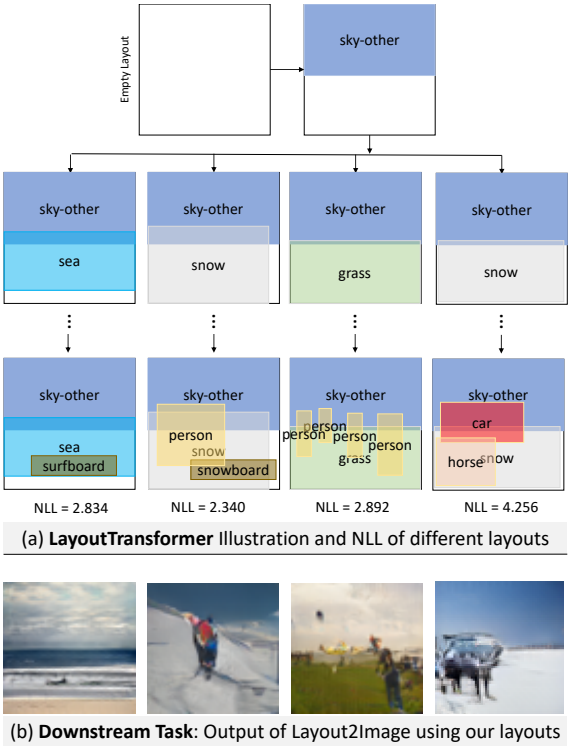

Figure 1: ((a) LayoutTransformer can generate multiple layouts consisting of variable number of elements starting from an empty canvas. (b) We can use tools such as Layout2Im [45] to generate image from layout (best viewed in color)

1. Introduction

In the real world, there exists a strong relationship between different objects that are found in the same environment [33, 31]. For example, a dining table usually has chairs around it; a surfboard is found near the sea; horses

do not ride cars, *etc*. Biederman [1] 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 [1], where this schema (or layout) can be characterized by contextual relationships between objects (*e.g*., support, occlusion, and relative likelihood, position, and size [1]). 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 [18, 17], animals [2, 43]). In order to generate complex scenes [5, 13, 15, 22, 29, 44, 45], most methods require proxy representations for layouts to be provided as inputs (*e.g*., scene segmentation [14, 36], textual descriptions [22, 44, 30], scene graphs [15]). 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 [1, 6]; 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). In this work, we propose an approach that learns such contextual relationships between objects and uses them to generate realistic layouts of scenes.

To generate plausible semantic layouts, we propose a sequential and iterative approach using self-attention. A layout is represented by a set of elements, with each element parameterized by semantic attributes/categories, location, and size. Our approach generates elements one by one in sequential order. To predict the next element in the layout, our approach computes contextual relationships between potential next elements and all elements in the current layout. Therefore, 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. This process is compactly modeled using a Transformer [35] based self-attention [24], which is order-invariant [42], *i.e*., regardless of the order in which the existing elements were originally placed in the layout, the model will generate an identical distribution for the next element, allowing us to compute likelihoods of the generated layouts.

Our approach, **LayoutTransformer**, has several advantages over current layout generation approaches without sacrificing their benefits. Similar to LayoutVAE [16], we can compute the likelihoods of our generated layouts; but unlike them, we do not require a list of categories present in the layout apriori. LayoutGAN [21] can generate a layout without an input, however it can only produce fixed-length layouts. Our approach can also do layout generation as well as completion of partial layouts. Finally, our self-attention model enables us to visualize what existing elements are important for generating the next category in the sequence.

Our approach can be readily plugged into many scene generation frameworks (e.g., Layout2Image [45], GauGAN [28]). 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 [7, 25], 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 [40, 3] or 3D scenes [4, 39, 38]. In this work, we will explore several such applications.

In summary, we propose LayoutTransformer, a layout generation and completion technique using self-attention. Our approach can generate arbitrary length layouts, compute their likelihoods, and improve the downstream tasks such as scene synthesis. We show the performance and adaptability of our model on four layout datasets: MNIST Layout [21], Rico Mobile App Wireframes [7], PubLayNet Documents [41], and COCO Bounding Boxes [23]. We devise quantitative evaluation strategies for the layout generation task for each of these datasets. We demonstrate that our approach can effectively capture key properties of plausible layouts, *e.g*., the spatial distribution of objects, their co-occurrences, *etc*. Finally, 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 [26]). This demonstrates the utility of the task of layout generation as a proxy-task for learning semantic representations.

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 [19] try to maximize approximate log-likelihood of data generated from a known distribution. Auto-regressive and flow-based approaches such as Pixel-RNN [34], RealNVP [8] 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 [2, 18], 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.

Scene generation. Most works in 2D or 3D scene synthesis generate a scene conditioned on a sentence [22, 44, 30], a scene graph [15], a layout [9, 12, 14, 37] or an existing image [20]. 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 [45], while other works generate diverse layouts and scenes [15, 22]. Again the focus of *all* these works is on the quality of the final image and not the feasibility of the layout itself.

Layout generation. Synthesising scene layouts is a relatively under-explored problem, mainly because generative models do not work well in practice for a set of elements. LayoutGAN [21] attempts to solve the problem by starting with maximum number of possible elements in the scene and modifying their geometric and semantic properties. LayoutVAE [16] 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.* [36] generate layout of an indoor room starting from top-down view of the room. However, their method cannot be easily extended to other datasets. In our approach, we do not require a starting label and can generate layouts of arbitrary lengths.

3. Our Approach

In this section, we introduce LayoutTransformer, our framework for generating layouts. The main challenge for current approaches is modelling global properties of a varied length unordered sequence of objects. The recently introduced Transformer architecture [35] addresses this problem with a non-local attention mechanism over sequences, and achieves state-of-the-art results on various language tasks. Therefore, we utilize the Transformer framework to model unordered layouts of varying lengths.

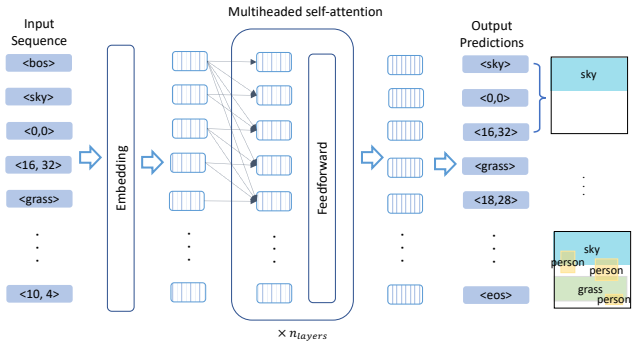

Figure 2: The architecture for the Transformer model depicted for a toy example. It takes layout elements as input and predicts next layout elements as output. During training, we use teacher forcing, *i.e.*, use the groundtruth layout tokens as input to a multi-head decoder block. The first layer of this block is a masked self-attention layer, which allows model to see only previous elements in order to predict 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.

Next, we provide a brief review of the relevant components of the Transformer architecture, and then present our approach in Section 3.2.

3.1. Preliminaries: Transformer

Transformers [35] are models that take a sequence of discrete values as input and generate another sequence of discrete values as output. The inputs are typically modeled with a lookup table where each input in the sequence is mapped to a low-dimensional embedding. Let $S = \{s_i\}_{i=1}^{N}$ be the vocabulary of discrete values in input sequences and $T = \{t_i\}_{i=1}^{M}$ be the vocabulary of discrete values in output sequences. We denote by $\phi_k = \phi(s_k)$ and $\theta_k = \theta(t_k)$ the lookup tables that map these discrete values to an embedding space. Given an input sequence of length n, $(s_1, s_2, ..., s_n)\,; s_i \in S$ and a target sequence of length m, $(t_1, t_2, ..., t_m)\,; t_i \in T$, the Transformer models the k^{th} value of the output sequence, as:

$$\begin{aligned} p(t_k) &= f_{\text{dec}}\left(f_{\text{enc}}(\phi_{1:n}), \theta_{1:k-1}\right) \\ p(t_1) &= f_{\text{dec}}\left(f_{\text{enc}}(\phi_{1:n}), \theta(t_{\text{bos}})\right) \\ p(t_{\text{eos}}) &= f_{\text{dec}}\left(f_{\text{enc}}(\phi_{1:n}), f_{\text{dec}}(\theta_{1:m})\right) \end{aligned} \tag{1}$$

where t_{bos} and t_{eos} are two special tokens to represent beginning and end of sequence. Each of encoder f_{enc} and decoder f_{dec} have stacked self-attention based pointwise fully connected architectures. Encoder f_{enc} consists of $n_{\text{layers}} = 6$ identical encoder blocks. Each encoder block has a multi-head self-attention mechanism with $n_{\text{heads}} = 8$ heads followed by a feed-forward network. Decoder f_{dec} follows

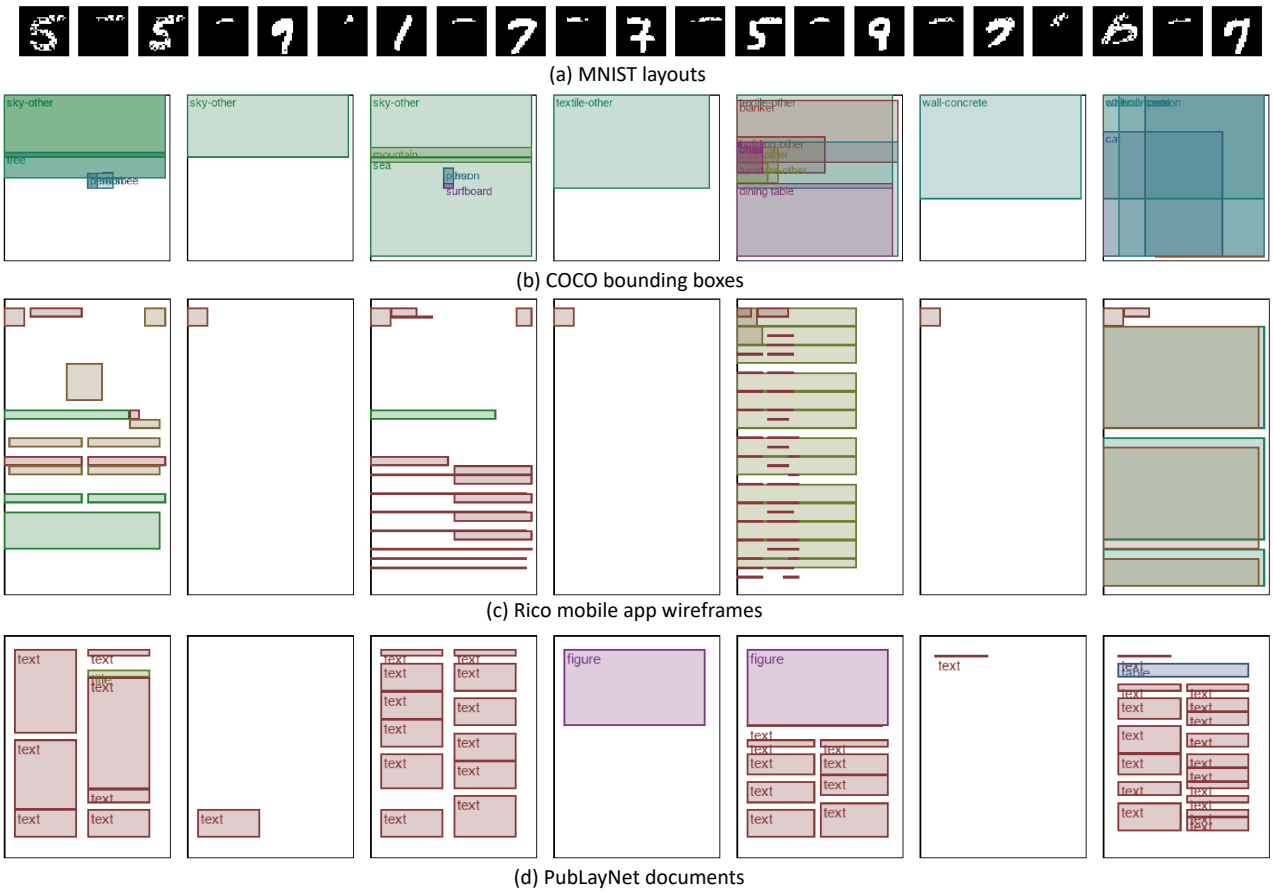

Figure 3: Generated layouts. First column (in each dataset) shows a random layout from validation data rendered on a blank image. We use part of the sequence in this layout (from validation data) to generate the most probable layout using our model. Second column onwards, we show initial layout given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

identical structure except each decoder block has an attention layer for input embeddings as well as a masked attention layer for output embeddings. Masking ensures that a decoder block only sees the embeddings of current and previous tokens. At the end of final decoder block, f_{dec} has a softmax layer to compute the probabilities of output tokens and minimize the cross-entropy loss.

Next, we present our Transformer-based approach for generating layout elements.

3.2. LayoutTransformer

Problem setup. Let each layout be a sequence of $m+2$ graphical elements $\{t_{\text{bos}}, t_1, \ldots, t_m, t_{\text{eos}}\}$, where m varies for different layouts in the dataset. The task of predicting the next graphical element t_i can be thought of as predicting the category $c_i \in C$, location $l_i \in L$ and geometry $g_i \in G$ of the graphical element. Hence, our layout can equivalently be represented as $\{t_{\text{bos}}, c_1, l_1, g_1, \ldots, c_m, l_m, g_m, t_{\text{eos}}\}$. Now, given just a start token t_{bos} or a set of graphical elements $(t_i, \ldots, t_k)$ as input, the objective of our model is to keep predicting $p(t_{k+1})$ until we reach the end of sequence token t_{eos}, *i.e.*, $\arg\max_t (p(t_{k+1})) = t_{\text{eos}}$. This framework is generic, and can be used to generate layouts in 1D, 2D or 3D scenes. However, in this work, we restrict our attention to 2D scenes where the graphical elements can be represented by a rectangular bounding box (possible extensions to more complex geometries, and 3D scenes is discussed in future work). Next we discuss how we represent each layout element.

Representation of a layout element. The category of an element encodes its semantic attributes. We represent the element category as one of the C possible categories. Given an image or scene of any size, we divide it into $H_{\text{I}} \times W_{\text{I}}$ grid cells. Then, the location of an element, given by the top-left corner of the element bounding box, can be represented by one of the $H_{\text{I}} \times W_{\text{I}}$ grid locations. Therefore, the vocabulary L needed to represent the location is of size $H_{\text{I}} \times W_{\text{I}}$. Alternatively, we can further split the task of predicting the location as first predicting the row and then predicting the column, which reduces the vocabulary to size $H_{\text{I}} + W_{\text{I}}$. We analyze these two discretization strategies in ablation studies. Similarly, for the geometry of the element, the height and width of the bounding box can be

represented by one of the H_{B} and W_{B} values each, with two discretization strategies discussed above. So, a single layout element can be represented by 3 values, one for location, one for geometry, and one for category. A layout of length m can be represented by $3m + 2$ discrete values including start and end of sequence tokens. Our base model uses $H_{\text{I}} = W_{\text{I}} = H_{\text{B}} = W_{\text{B}} = 32$. We refer to this parameter as n_{anchor} and we discuss other values in ablation studies.

Network structure. For simplicity, we eliminate the encoder from the Transformer architecture, and use only the decoder block, *i.e.*, each element gets mapped to a d-dimensional embedding such that $\theta_k = \theta(t_k) \in \mathbb{R}^d$. Therefore, the model of 1 simplifies to:

$$\begin{aligned} p(t_1) &= f_{\text{dec}}(\theta(t_{\text{bos}})) \\ p(t_k) &= f_{\text{dec}}(\theta_{1:k-1}) \\ p(t_{\text{eos}}) &= f_{\text{dec}}(f_{\text{dec}}(\theta_{1:m})) \end{aligned} \tag{2}$$

We also remove the sinusoidal position encoding proposed in the original Transformer model as we want our decoder block to be invariant to the order of input elements. Figure 2 shows the network architecture for a sample sequence in the training data.

Order of elements. We use a raster scan order of layout elements to train our model. During training, the decoder is invariant to the order of input elements; but it predicts the next element according to the raster scan order. This is beneficial to simplify the decoder training. In ablation studies, we discuss an alternate strategy.

Decoding. At training time, we know all the sequences, and our model training can be done efficiently in parallel for different sequence lengths. During inference, we are given with just begin 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 [32], *i.e.*, keep track of b most likely sequences while decoding to generate multiple possible layouts starting from the same initial sequence.

Training details. In our base model, we use $d = 512$, $n_{\text{layers}} = 6$ and $n_{\text{heads}} = 8$ in the decoder. 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.

4. Experiments

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

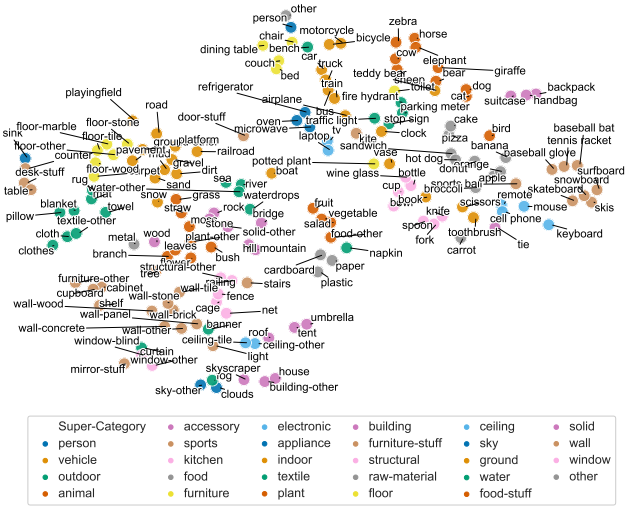

Figure 4: Word embeddings as learned by LayoutTransformer model on COCO Bounding Boxes. Words are colored by their super-categories provided in the COCO dataset. We can see that semantically similar categories cluster together - e.g. animals, fruits, furniture etc. Cats and dogs are closer to each other as compared to sheep, zebra or cow. Also, certain words which belong to different super-categories in the dataset such as curtain, window-blind, and mirror appear close in the embedding space.

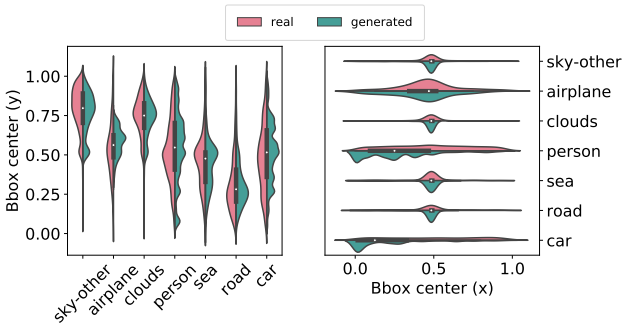

Figure 5: We plot the distribution of x-coordinates and y-coordinates of the center of bounding boxes (normalized between 0 and 1). The y-coordinate is more informative about different categories, for example, sky and clouds are usually on the top of the image while road and sea are usually at the bottom. Distributions for generated layouts and real layouts tend to be similar.

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 [21], Rico Mobile App Wireframes [7, 25], COCO bounding boxes [23] and PubLayNet Documents [41]. Note that each of the

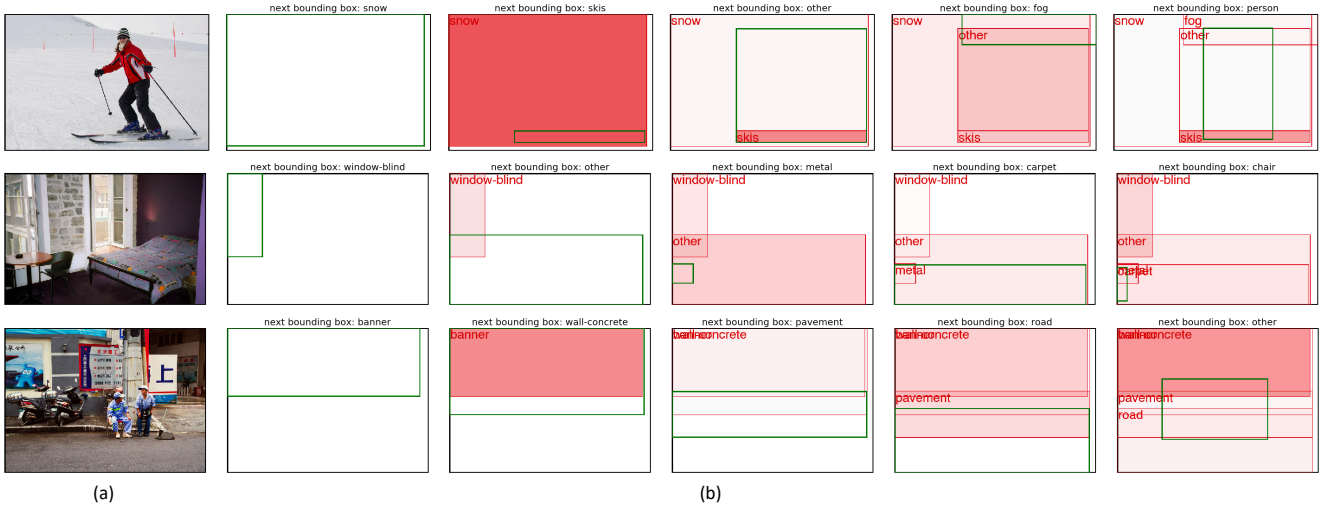

Figure 6: Visualizing attention. (a) Image source for the layout (b) In each row, the model is predicting one element at a time (shown in a green bounding box). While predicting that element, the model pays the most attention to previously predicted bounding boxes (in red). For example, in the first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

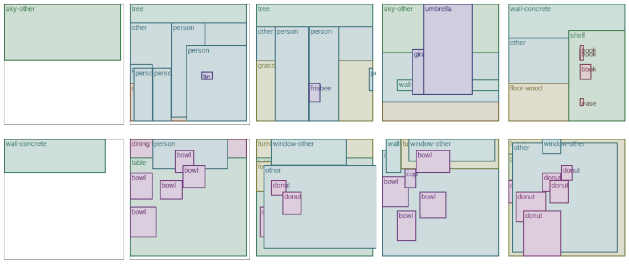

Figure 7: Nearest neighbors from training data. Column 1 shows the initial layout provided (for completion) to LayoutTransformer. Column 2 shows the layout as generated by LayoutTransformer. Column 3, 4 and 5 shows the 3 closest neighbors from training dataset. We use chamfer distance on bounding box coordinates to obtain the nearest neighbors from the dataset.

datasets involves a pre-processing step, and we tried to faithfully replicate these steps as provided in original publications [21, 16]. We will release the code for pre-processing, our approach, and experiments for reproducibility and future reference.

MNIST Layout. Following LayoutGAN [21], 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 [23]. We ignore the images where the *isCrowd* flag is true following the LayoutVAE [16] 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 [7, 25] 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 [41] 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[1]:

LayoutGAN. LayoutGAN [21] represents each layout with a fixed number of bounding boxes. Starting with bounding box coordinates sampled from a Gaussian distribution, its

[1]Note that code for LayoutGAN and LayoutVAE is not available and we use our own implementation to make the comparisons

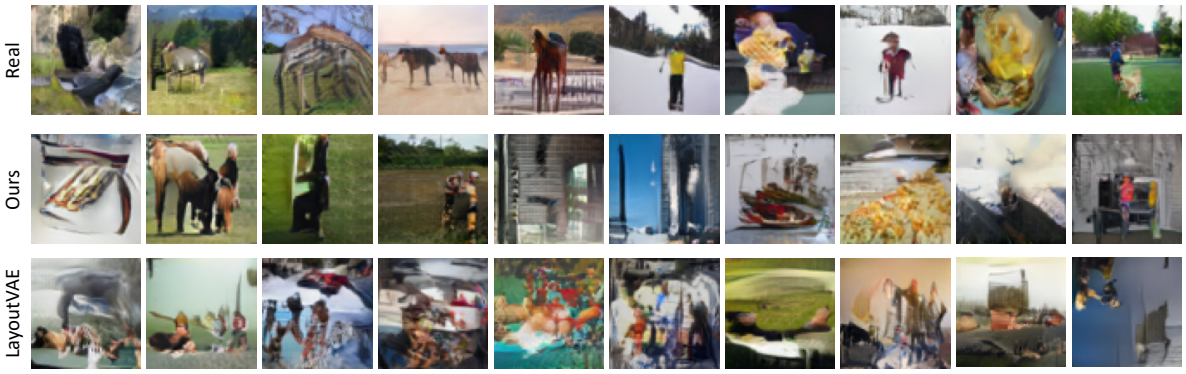

Figure 8: Image generation from layouts. We use the work of Zhao et al. [45] to convert layouts to images. The first row shows images using layouts from validation data. The second row shows generated novel layouts converted to image using the same model. The third row shows layouts generated by LayoutVAE converted to image in a similar manner. Images generated with our layouts have less clutter and more recognizable objects.

Table 1: 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. Clearly, the statistics of our layouts are the more similar to the real data statistics than LayoutVAE or LayoutGAN. All values in the table are percentages (std in parenthesis)

	Rico		PubLayNet	
Methods	Coverage	Overlap	Coverage	Overlap.
LayoutGAN [21]	97.2 (3)	4.8 (6)	1.7 (10)	71.8 (39)
LayoutVAE [16]	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)

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. In our implementation of LayoutGAN, we were unable to prevent mode collapse. We show the layouts generated in the supplementary material.

LayoutVAE. LayoutVAE [16] 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.

4.3. Qualitative Evaluation

Next we present qualitative analysis of our model.

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 top-left and bottom-right bounding box coordinates of layout elements. Figure 7 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

Category embeddings. We can visualize the 2D-tsne plot of learned embeddings for categories, as shown in Figure 4. 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 each other.

Image generation. To evaluate the ability of layout generation approaches in generating plausible layouts for the COCO dataset we use Layout2Im [45] to generate an image from a layout. Figure 8 shows images generated from layouts by our method and LayoutVAE. As seen in the Figure 8, images generated by our layouts have more recognizable objects and animals as compared to LayoutVAE.

4.4. 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.

Negative log-likelihood. For each of the datasets, Table 3 shows the negative log-likelihood of all the layouts in vali-

Table 2: **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 3: Negative log-likelihood of all the layouts in validation set (lower the better). LayoutGAN (our implementation) numbers are computed using [10], LayoutVAE (our implementation) using importance sampling as described in their paper.

Model	COCO	Rico	PubLayNet
sg2im [15]	214.03	-	-
LayoutVAE [16]	4.28	3.72	2.56
Ours	3.76	2.13	1.85

Table 4: Effect of n_{anchors} on NLL

n_{anchors}	# params	COCO	Rico	PubLayNet
32×32	21.2	3.329	1.491	1.455
8×8	19.2	2.527	1.462	1.285
16×16	19.6	3.043	1.787	1.522
64×64	27.5	4.594	2.584	2.237
128×128	52.7	5.588	3.237	2.654

Table 5: Effect of d on NLL

d	# params	COCO	Rico	PubLayNet
512	21.2	3.329	1.491	1.455
32	0.97	4.042	2.736	2.012
64	2.0	3.905	2.465	1.953
128	4.1	3.818	2.302	1.902
256	9.0	3.809	2.197	1.872

Table 6: Effect of n_{layers} on NLL

n_{layers}	# params	COCO	Rico	PubLayNet
6	21.2	3.329	1.491	1.455
2	8.6	3.884	2.317	1.920
4	14.9	3.776	2.198	1.874
8	27.5	3.784	2.089	1.844

Table 7: Effect of other model hyperparameters on NLL

Order	Split-XY	Loss	# params	COCO	Rico	PubLayNet
raster	Yes	NLL	21.2	3.329	1.491	1.455
random			21.2	4.440	3.169	2.419
	No		19.2	2.289	1.085	1.095
		LS	21.2	3.762	2.128	1.851

dation set. The results indicate that our approach generates more plausible layouts (more details on 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 2. **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.

4.5. Ablation studies

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

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 4).

Size of embedding: Increasing the size of the embedding d improves the NLL, but at the cost of increased number of parameters (Table 5).

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

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 7).

Discretization strategy: Instead of the next bounding box, we tried predicting the x-coordinate and y-coordinate of the bounding box alternately (refer to the Split-xy column of Table 7). This reduces the vocabulary size of the model and without any decline in model performance. The only downside of this approach is that generating new layouts takes longer as we have to make twice as many predictions for each element of the layout. (Table 7)

Loss: We tried two different losses, label smoothing [27] 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 7)

5. Conclusion

We propose LayoutTransformers, a novel approach to generate layouts of graphical elements. Our model uses self-attention model to capture contextual relationship 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 while it is capable of predicting size and location of objects of the scene, it cannot be trivially extended to predict object masks. We will explore these directions in the future work.

References

[1] Irving Biederman. On the semantics of a glance at a scene. In *Perceptual organization*, pages 213–253. Routledge, 2017. 2

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 2, 3

[3] Samuele Capobianco and Simone Marinai. Docemul: a toolkit to generate structured historical documents. *CoRR*, abs/1710.03474, 2017. 2

[4] Angel X. Chang, Will Monroe, Manolis Savva, Christopher Potts, and Christopher D. Manning. Text to 3d scene generation with rich lexical grounding. *CoRR*, abs/1505.06289, 2015. 2

[5] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1511–1520, 2017. 2

[6] Xinlei Chen, Abhinav Shrivastava, and Abhinav Gupta. Neil: Extracting visual knowledge from web data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1409–1416, 2013. 2

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 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. 2, 5, 6

[8] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 3

[9] Hao Dong, Simiao Yu, Chao Wu, and Yike Guo. Semantic image synthesis via adversarial learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5706–5714, 2017. 3

[10] Hamid Eghbal-zadeh and Gerhard Widmer. Likelihood estimation for generative adversarial networks. *CoRR*, abs/1707.07530, 2017. 8

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 3

[12] Tobias Hinz, Stefan Heinrich, and Stefan Wermter. Generating multiple objects at spatially distinct locations. *CoRR*, abs/1901.00686, 2019. 3

[13] Seunghoon Hong, Dingdong Yang, Jongwook Choi, and Honglak Lee. Inferring semantic layout for hierarchical text-to-image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7986–7994, 2018. 2

[14] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016. 2, 3

[15] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1219–1228, 2018. 2, 3, 8

[16] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. Layoutvae: Stochastic scene layout generation from a label set. *arXiv preprint arXiv:1907.10719*, 2019. 2, 3, 6, 7, 8

[17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 2

[18] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019. 2, 3

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. 3

[20] Donghoon Lee, Sifei Liu, Jinwei Gu, Ming-Yu Liu, Ming-Hsuan Yang, and Jan Kautz. Context-aware synthesis and placement of object instances. *CoRR*, abs/1812.02350, 2018. 3

[21] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*, 2019. 2, 3, 5, 6, 7

[22] Wenbo Li, Pengchuan Zhang, Lei Zhang, Qiuyuan Huang, Xiaodong He, Siwei Lyu, and Jianfeng Gao. Object-driven text-to-image synthesis via adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12174–12182, 2019. 2, 3

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 5, 6

[24] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017. 2

[25] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. Learning design semantics for mobile apps. In *The 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, pages 569–579, New York, NY, USA, 2018. ACM. 2, 5, 6

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 2

[27] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? *CoRR*, abs/1906.02629, 2019. 8

[28] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Gaugan: semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!*, page 2. ACM, 2019. 2

[29] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019. 2

[30] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016. 2, 3

[31] Abhinav Shrivastava and Abhinav Gupta. Contextual priming and feedback for faster r-cnn. In *European Conference on Computer Vision*, pages 330–348. Springer, 2016. 1

[32] Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. Improvements in beam search. In *Third International Conference on Spoken Language Processing*, 1994. 5

[33] Antonio Torralba and Pawan Sinha. Statistical context priming for object detection. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 763–770. IEEE, 2001. 1

[34] Aäron van den Oord and Nal Kalchbrenner. Pixel rnn. 2016. 3

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 2, 3

[36] Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)*, 37(4):70, 2018. 2, 3

[37] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 3

[38] Jiajun Wu, Erika Lu, Pushmeet Kohli, William T Freeman, and Joshua B Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, 2017. 2

[39] Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[40] Xiao Yang, Mehmet Ersin Yümer, Paul Asente, Mike Kraley, Daniel Kifer, and C. Lee Giles. Learning to extract semantic structure from documents using multimodal fully convolutional neural network. *CoRR*, abs/1706.02337, 2017. 2

[41] Antonio Jimeno Yepes, Jianbin Tang, and Xu Zhong. Publaynet: largest dataset ever for document layout analysis. 2, 5, 6

[42] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017. 2

[43] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018. 2

[44] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017. 2, 3

[45] Bo Zhao, Lili Meng, Weidong Yin, and Leonid Sigal. Image generation from layout. In *CVPR*, 2019. 1, 2, 3, 7