LayoutGAN: Synthesizing Graphic Layouts With Vector-Wireframe Adversarial Networks

Jianan Li, Jimei Yang, Aaron Hertzmann, *Fellow, IEEE*, Jianming Zhang, and Tingfa Xu

Abstract—Layout is important for graphic design and scene generation. We propose a novel Generative Adversarial Network, called LayoutGAN, that synthesizes layouts by modeling geometric relations of different types of 2D elements. The generator of LayoutGAN takes as input a set of randomly-placed 2D graphic elements, represented by vectors and uses self-attention modules to refine their labels and geometric parameters jointly to produce a realistic layout. Accurate alignment is critical for good layouts. We, thus, propose a novel differentiable wireframe rendering layer that maps the generated layout to a wireframe image, upon which a CNN-based discriminator is used to optimize the layouts in image space. We validate the effectiveness of LayoutGAN in various experiments including MNIST digit generation, document layout generation, clipart abstract scene generation, tangram graphic design, mobile app layout design, and webpage layout optimization from hand-drawn sketches.

Index Terms—Generative adversarial networks, graphic design, layout, wireframe

1 Introduction

Graphic design is an important visual communication tool in our modern world, encompassing everything from book covers to magazine layouts to web design. Whereas methods for generating realistic natural-looking images have made significant progress lately, particularly with Generative Adversarial Networks (GANs) [1], methods for creating designs are far more primitive. This is, in part, due to the difficulty of finding data representations suitable for learning. Graphic designs are normally composed of vector representations of primitive objects, such as polygons, curves and ellipses, instead of pixels laid on a regular lattice. The quality and content of a design depends on the presence of elements, their attributes, and their relations to other elements. The visual perception of design depends on the arrangement of these elements; misalignment of two elements of just a few millimeters can ruin the design. Conventional GANs that synthesize pixels directly from convolutional features will not respect such an element-wise data structure but instead mix up the vector-space layout with its pixel-space rendering, i.e., images, and are thus likely to generate blurrier and smasher design images without clear notations of elements, which are hard to interpret or edit. However, design elements are often explicit representations of human abstract knowledge about visual world and their layouts represent how these knowledge are organized to convey ideas visually [2], [3], [4], [5]. For example, a document heading often appears on top left of a paragraph (Fig. 5), or if a boy wears a hat then the hat is right on top of his head (Fig. 8). We are interested in building a neural network model that respects not only the intrinsic data structure of graphic design but also its visual perception by humans.

This paper introduces LayoutGAN, a novel GAN which directly arranges a set of graphical elements in a design. In a given design problem, a fixed set of element classes (e.g., "title," "figure") is specified in advance. In our network, each element is represented by its class probabilities and its geometric parameters, i.e., bounding-box keypoints. The generator takes as input random layouts, formatted as graphic elements with randomly-sampled class probabilities and geometric parameters, and arranges them in a design; the output is the refined class probabilities and geometric parameters of the design elements. Using random layouts as input allows our method to handle two different tasks in a single representation: it can generate pure layouts from scratch using random layouts, and it can take existing layouts as input and refine them. The generator is permutation-invariant: it will generate the same layout if we re-order the input elements.

We propose two kinds of discriminator networks for this structured data. The first is similar in structure to the generator: it operates directly on the class probabilities and geometric parameters of the elements. Though effective, it is not sensitive enough to misalignment and occlusion between elements. The second discriminator operates in the visual domain. Like a human viewer who judges a design by looking at its rasterized image, this discriminator evaluates the relations among different elements by mapping them to 2D bitmaps. Then Convolutional Neural Networks (CNNs) can be used for layout optimization as they are specialized in distinguishing visual patterns including but not limited to misalignment and occlusion. However, the key challenge is how to map the geometric parameters to pixel-level layouts differentiably. One approach would be to render the graphic elements

• *J. Li and T. Xu are with the Beijing Institute of Technology, 5 South Zhongguancun Street, Haidian District, Beijing 100081, China. E-mail: {20090964, xutingfa}@bit.edu.cn.*
• *J. Yang, A. Hertzmann, and J. Zhang are with the Adobe Research, 345 Park Ave, San Jose, CA 95110 USA. E-mail: {jimyang, hertzman, jianmzha}@adobe.com.*

Manuscript received 20 Feb. 2019; revised 4 Dec. 2019; accepted 29 Dec. 2019. Date of publication 1 Jan. 2020; date of current version 3 June 2021. (Corresponding author: Jianan Li.) Recommended for acceptance by D. Lowd. Digital Object Identifier no. 10.1109/TPAMI.2019.2963663

into bitmap masks using Spatial Transformer Networks [6]. But we find that filled pixels within the design elements cause occlusions and are ineffective for back-propagation, for example, when a small polygon hides behind a larger one. We experimented with bitmap mask rendering but it failed to converge. In this paper, we propose a novel differentiable wireframe rendering layer that rasterizes both synthesized and real structured data of graphic elements into wireframe images, upon which a standard CNN can be used to optimize the layout across the visual and the graphic domain. The wireframe rendering discriminator has several advantages. First, convolution layers are very good at extracting spatial patterns of images so that they are more sensitive to alignment. Second, the rendered wireframes make elements visible even when they overlap and thus the network is alleviated from inferring the occlusions that may occur in other renderings such as masks.

We evaluate the LayoutGAN on a variety of different tasks, including a sanity test on MNIST digits, generating page layouts from labeled bounding boxes, generating clipart abstract scenes, tangram graphic design, mobile app design layouts, and optimizing webpage layouts from hand-drawn sketches. In each case, our method successfully generates layouts respecting the types of elements and their relationships for the problem domain.

This paper is an extension of the work originally presented in Li *et al.* [7], upon which we further explore the ability of the LayoutGAN for layout optimization besides layout generation and provide more details and analysis about the experiments. In summary, the LayoutGAN comprises the following contributions: 1. A generator that directly synthesizes structured data, represented as a resolution-independent set of labeled graphic elements in a design. 2. A differentiable wireframe rendering layer which allows the discriminator to judge alignment from discrete element arrangements.

2 Related Work

Structured Data Generation. Convolutional networks have been shown successful for generating data in regular lattice data types, such as images [8], [9], videos [10], [11] and 3D volumes [12], [13]. When generating highly-structured data, such as text [14], [15] and programs [16], recurrent networks are often the first choice [17], especially equipped with attention [18], [19] and memory modules [20]. Recently, researchers show that convolutional networks can be also used to synthesize sequences [21], [22] using auto-regressive models. However, in many cases, an object has no sequential order [23], but is a set of elements, e.g., point clouds. Fan *et al.* [24] propose a point set generation network for synthesizing 3D point clouds of the object shape from a single image. It is further paired with a point set classification network [25] for auto-encoding 3D point clouds [26]. Our work extends the set representation to more general primitive objects, i.e., labeled polygons. Meanwhile, researchers also model the structured data of connected elements using graph convolutions [27].

Data-Driven Graphic Design. Automating layout is a classic problem in graphic design [28], [29]. ODonovan *et al.* [30] formulate an energy function by assembling various heuristic visual cues and design principles to optimize single-page layouts, and extend this to an interactive tool [31]. The model requires design content features as input, and parameters are learned from a small number of example designs. Pang *et al.* [32] optimize layout for desired gaze direction. Deka *et al.* [4] collect a mobile app design database for harnessing data-driven applications and present preliminary results of learning similarities of pixel-level textural/non-textual masks for design search, but do not learn models from this data. Swearngin *et al.* [33] propose an interactive system that converts example design screenshots to vector graphics for designers to re-use and edit. Bylinskii *et al.* [34] analyze the visual importance of graphic designs and use saliency map as the driving force to assist retargeting and thumbnailing. Previous methods have learned models for other design elements, such as fonts [35] and colors [36]. These are orthogonal to the layout problem, and could be combined in future work. No previous method has learned to create design or layout from large datasets, and no previous work has applied GANs to layout.

3D Scene Synthesis. Interior scene synthesis [37] and furniture layout generation [38] draw great interest in graphics community. Early approaches focus on optimization of hand-crafted design principles [39] and learning statistical priors of pairwise object relationships [40] due to limited data. Wang *et al.* [41] proposed a sequential decision making approach to indoor scene synthesis. In each step, a CNN is trained to predict either location or category of one object by looking at the rendered top-down views. This resembles our wireframe rendering discriminator, in the sense of using convolutions to capture spatial patterns of layouts.

3 LayoutGAN

This section describes our data and model representations.

3.1 Design Representation

In our model, a graphic design is comprised of a set of N primitive design elements $\{(\boldsymbol{p}_1, \boldsymbol{\theta}_1), \ldots, (\boldsymbol{p}_N, \boldsymbol{\theta}_N)\}$. Each element has a set of geometric parameters $\boldsymbol{\theta}$, and a vector of class probabilities $\boldsymbol{p}$. The entries of these variables are problem-dependent. For example, document layouts include 6 classes, e.g., "title" and "picture" while clipart layouts include 6 classes, e.g., "boy" and "hat". For 2D point set generation (in MNIST digits), $\boldsymbol{\theta} \equiv [x, y]$, representing the coordinates of each point; for bounding-box generation in document layout, $\boldsymbol{\theta} \equiv [x^L, y^T, x^R, y^B]$, representing the top-left and bottom-right coordinates of each bounding box; for layouts with scale and flip (clipart abstract scenes), $\boldsymbol{\theta} \equiv [x, y, s, l]$, representing the center coordinates, scale and flip of each element.

3.2 Generator Architecture

In the LayoutGAN, the generator is a function $G(z)$ that takes a layout as input, where $z = \{(\boldsymbol{p}_1, \boldsymbol{\theta}_1), \ldots, (\boldsymbol{p}_N, \boldsymbol{\theta}_N)\}$ consisting of initial graphic elements with randomly-sampled geometric parameters $\boldsymbol{\theta}_i$, and one-hot encoding of a randomly-sampled class $\boldsymbol{p}_i$. The generator outputs a refined layout $G(z) = \{(\boldsymbol{p}_1{}', \boldsymbol{\theta}_1{}'), \ldots, (\boldsymbol{p}_N{}', \boldsymbol{\theta}_N{}')\}$ which is meant to resemble a real graphic design. Note that, unlike, conventional GANs where z represents a low-dimensional latent variable, our z represents initial random graphic layout that has the same structure as the real one. The

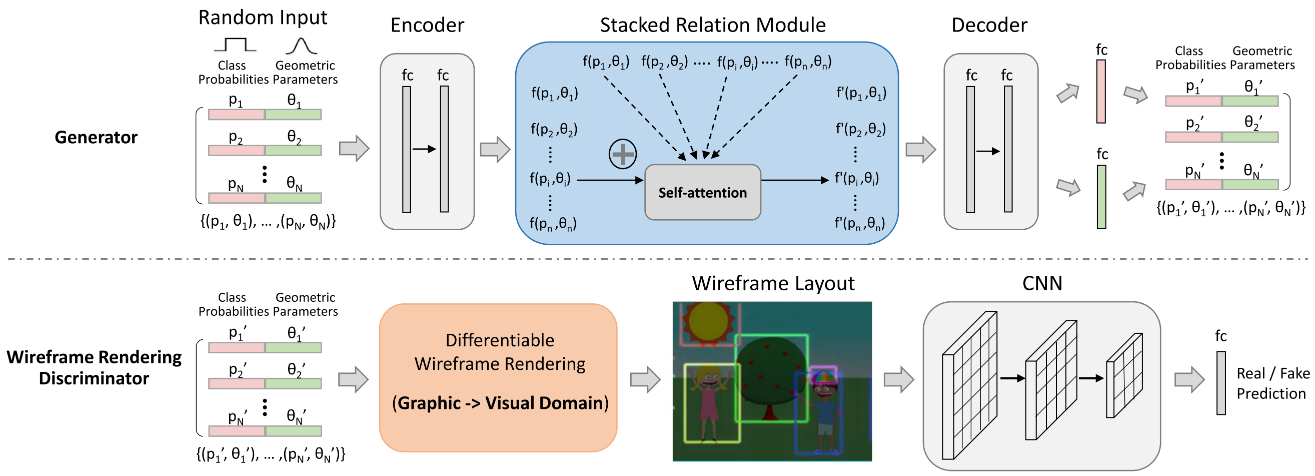


Fig. 1. Overall architecture of LayoutGAN. The generator takes as input graphic elements with randomly sampled class probabilities and geometric parameters from Uniform and Gaussian distribution respectively. An encoder embeds the input and feeds them into the stacked relation module, which refines the embedded features of each element in a coordinative manner by considering its semantic and spatial relations with all the other elements. Finally, a decoder decodes the refined features back to class probabilities and geometric parameters. The wireframe rendering discriminator feeds the generated results to a differentiable wireframe rendering layer which raterizes the input graphic elements into 2D wireframe images, upon which a CNN is applied for layout optimization.

discriminator learns to capture the geometric relations among different types of elements for layout optimization from both the graphic and the visual domain. Next, we go into details of the generator and discriminator design.

As illustrated in Fig. 1, the generator takes as input a set of graphic elements with random class probabilities and geometric parameters sampled from Uniform and Gaussian distributions, respectively. An encoder consisting of a multilayer perceptron network (implemented as multiple fully-connected layers) first embeds the class one-hot vectors and geometric parameters of each graphic element. The relation module, implemented as self-attention in Wang *et al.* [42], is then used to adjust the relations among all elements jointly by embedding the feature of each graphic element as a function of its spatial context, i.e., its relations with all the other elements in the design. Denote $f(\boldsymbol{p_i}, \boldsymbol{\theta_i})$ as the embedded feature of the graphic element i, its refined feature representation $f'(\boldsymbol{p_i}, \boldsymbol{\theta_i})$ can be obtained through a contextual residual learning process, which is defined as

$$f'(\boldsymbol{p_i}, \boldsymbol{\theta_i}) = \\ \boldsymbol{W_r} \frac{1}{N} \sum_{\forall j \neq i} H(f(\boldsymbol{p_i}, \boldsymbol{\theta_i}), f(\boldsymbol{p_j}, \boldsymbol{\theta_j})) U(f(\boldsymbol{p_j}, \boldsymbol{\theta_j})) + f(\boldsymbol{p_i}, \boldsymbol{\theta_i}), \tag{1}$$

where a unary function U performs a linear embedding (implemented as 1×1 convolution) on the feature $f(\boldsymbol{p_j}, \boldsymbol{\theta_j})$ of element j and a pairwise function H computes a scalar value representing the relation between elements i and j. Thus, all the other elements $j \neq i$ contribute to the feature refinement of element i by summing up their relations. The response is normalized by the total number of elements in the set, N. The weight matrix $\boldsymbol{W_r}$ computes a linear embedding, producing the contextual residual to be added to $f(\boldsymbol{p_i}, \boldsymbol{\theta_i})$ for feature refinement. In our experiments, we define H as a dot-product

$$H(f(\boldsymbol{p_i}, \boldsymbol{\theta_i}), f(\boldsymbol{p_j}, \boldsymbol{\theta_j})) = \boldsymbol{\psi}(f(\boldsymbol{p_i}, \boldsymbol{\theta_i}))^T \boldsymbol{\phi}(f(\boldsymbol{p_j}, \boldsymbol{\theta_j})), \tag{2}$$

where $\boldsymbol{\psi}(f(\boldsymbol{p_i}, \boldsymbol{\theta_i})) = \boldsymbol{W_\psi} f(\boldsymbol{p_i}, \boldsymbol{\theta_i})$ and $\boldsymbol{\phi}(f(\boldsymbol{p_j}, \boldsymbol{\theta_j})) = \boldsymbol{W_\phi} f(\boldsymbol{p_j}, \boldsymbol{\theta_j})$ are two linear embeddings. We stack 4 relation modules for feature refinement in our experiments. Finally, a decoder consisting of another multilayer perceptron network followed by two branches of fully connected layer with a sigmoid activation function is used to map the refined feature of each element back to class probabilities and geometric parameters respectively. Optionally, non-maximum suppression can be applied to remove duplicate elements [43] (not included in our experiments). During the whole generating process, each input random element is first processed individually by a shared encoder and then adjusted by its relations with other elements in relation modules and finally its output vector representation is decoded independently by a shared Multilayer Perception (MLP). All these steps handle input elements regardless of their order when fed into the network. Thus the generator is permutation-invariant. This is beneficial for training as the layout data often lacks sequential annotations of elements being placed on the page.

3.3 Discriminator Network Architectures

The discriminator aims to distinguish between synthesized and real layouts. We present two types of discriminators, one based on relation modules directly built upon layout parameters, and the other based on how the layouts look like via rendering.

3.3.1 *Relation-Based Discriminator*

The relation-based discriminator takes as input a set of graphic elements represented by class probabilities and geometric parameters, and feeds them to an encoder consisting of a multilayer perceptron network for feature embedding $f(\boldsymbol{p_i}, \boldsymbol{\theta_i})$. It then extracts their global graphical relations $g(\boldsymbol{r}(\boldsymbol{p_1}, \theta_1), \ldots, \boldsymbol{r}(\boldsymbol{p_N}, \theta_N))$ where $\boldsymbol{r}(\boldsymbol{p_i}, \boldsymbol{\theta_i})$ represents a simplified relation module as in Equation (1) by removing the shortcut connection (addition of $f(\boldsymbol{p_i}, \boldsymbol{\theta_i})$), and g is a max-pooling function used in Charles *et al.* [25]. Thus, the global

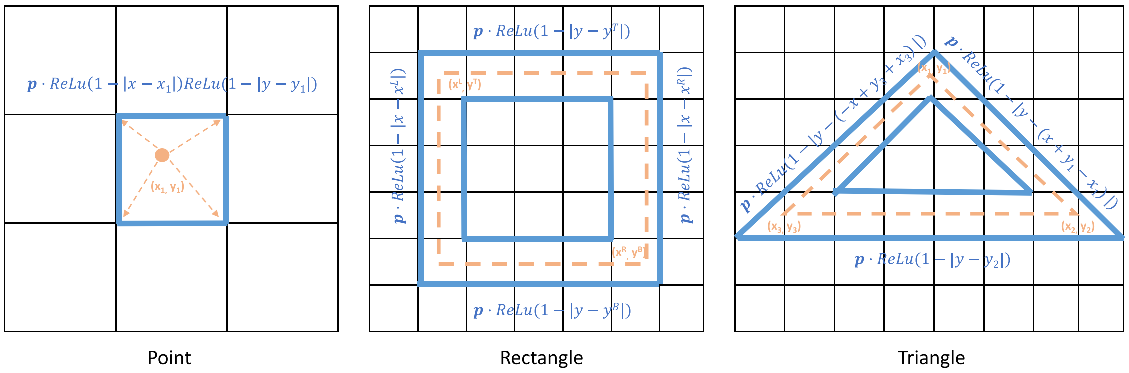

$p \cdot ReLu(1-|x-x_1|)ReLu(1-|y-y_1|)$
(x_1, y_1)
Point
$p \cdot ReLu(1-|y-y^T|)$
$p \cdot ReLu(1-|x-x^L|)$
$p \cdot ReLu(1-|x-x^R|)$
(x^L, y^T)
(x^R, y^B)
$p \cdot ReLu(1-|y-y^B|)$
Rectangle
$p \cdot ReLu(1-|y-(-x+y_3+x_3)|)$
$p \cdot ReLu(1-|y-(x+y_1-x_1)|)$
(x_1, y_1)
(x_3, y_3)
(x_2, y_2)
$p \cdot ReLu(1-|y-y_2|)$
Triangle

Fig. 2. Wireframe rendering of different polygons (point, rectangle and triangle). The black grids represent grids of target image. The orange dots/dotted lines represent the graphic element mapped onto the image grid. The blue solid lines represent the rasterized wireframes expressed as differentiable functions of graphic elements in terms of both class probilities and geometric parameters.

relations among all the graphic elements can be modeled, upon which a classifier composed by a multilayer perception network is applied for real/fake prediction.

3.3.2 *Wireframe Rendering Discriminator*

The wireframe rendering discriminator exploits CNNs to classify layouts, in order to learn to classify the visual properties of a layout. The discriminator consists of a wireframe rendering layer, which produces an output image $\boldsymbol{I}$, which is then fed into a CNN for classification.

The rasterization is performed as follows. Suppose there are N elements in the design, with parameters $\{(\boldsymbol{p_1}, \boldsymbol{\theta_1}), \ldots, (\boldsymbol{p_N}, \boldsymbol{\theta_N})\}$. Each element can be rendered into its own grayscale image $\boldsymbol{F}_{\theta}(x, y)$; rendering details for specific types of elements are given below. The image dimensions for each individual $\boldsymbol{F}$ are $W \times H$, where W and H are the width and height of the design in pixels.

The layer output is a multi-channel image $\boldsymbol{I}$ of dimensions $W \times H \times M$, where each channel corresponds to one of the M element types. In other words, pixel (x, y) of $\boldsymbol{I}$ is a class activation vector for that pixel, and is computed as

$$\boldsymbol{I}(x, y, c) = \max_{i \in [1 \ldots N]} p_{i,c} \boldsymbol{F}_{\theta_i}(x, y), \tag{3}$$

where $p_{i,c}$ is the probability that element i is of class c.

We next describe the rendering process for computing $\boldsymbol{F}_{\theta}$, in the cases where θ represents a point, a rectangle, or a triangle.

Point. We start with the simplest geometric form, a single keypoint $\boldsymbol{\theta_i} = (x_i, y_i)$ for element i. We implement an interpolation kernel k for its rasterization. Its spatial rendering response on (x, y) in the rendered image can be written as

$$\boldsymbol{F}_{\boldsymbol{\theta_i}}(x, y) = k(x - x_i) k(y - y_i). \tag{4}$$

We adopt bilinear interpretation [44], corresponding to the kernel $k(d) = \max(0, 1 - |d|)$ (implemented as ReLU activation), as shown in Fig. 2. As $\boldsymbol{I}$ is a differentiable function of the class probabilities and the coordinates, the subgradients of the rasterized image can be thus propagated backward to them. We validate such rendering design for MNIST digit generation, detailed experiments can be seen in Section 4.1.

Rectangle. We now consider more complex polygons. Assuming an element is a rectangle, or bounding box represented by its top-left and bottom-right coordinates $\boldsymbol{\theta} = (x^L, y^T, x^R, y^B)$, which is very common in various designs. Specifically, considering a rectangle $\boldsymbol{i}$ with coordinates $\boldsymbol{\theta_i} = (x_i^L, y_i^T, x_i^R, y_i^B)$, as shown in Fig. 2, the black grids represent the locations in the rendered image and the orange dotted box represents the rectangle being rasterized in the rendered image. For a wireframe representation, only the points near the boundary of the dotted box (lie in blue solid line) are related to the rectangle, so its spatial rendering response on (x, y) can be formulated as

$$\boldsymbol{F}_{\boldsymbol{\theta_i}}(x, y) = \max \begin{pmatrix} k(x - x_i^L) b(y - y_i^T) b(y_i^B - y), \\ k(x - x_i^R) b(y - y_i^T) b(y_i^B - y), \\ k(y - y_i^T) b(x - x_i^L) b(x_i^R - x), \\ k(y - y_i^B) b(x - x_i^L) b(x_i^R - x) \end{pmatrix}, \tag{5}$$

where $b(d) = \min(\max(0, d), 1)$ constraining the rendering to nearby pixels.

Triangle. We further describe the wireframe rendering process of another geometric form, triangle. For triangle i represented by its three vertices' coordinates $\boldsymbol{\theta_i} = (x_i^1, y_i^1, x_i^2, y_i^2, x_i^3, y_i^3)$, when $x_i^1 \neq x_i^2 \neq x_i^3$, its spatial rendering response on (x, y) in the rendered image can be calculated as

$$\boldsymbol{F}_{\boldsymbol{\theta_i}}(x, y) = \max \begin{pmatrix} k\left(y - \frac{(y_i^2 - y_i^1) \cdot (x - x_i^1)}{x_i^2 - x_i^1} - y_i^1\right) b(x - x_i^1) b(x_i^2 - x), \\ k\left(y - \frac{(y_i^3 - y_i^1) \cdot (x - x_i^1)}{x_i^3 - x_i^1} - y_i^1\right) b(x - x_i^3) b(x_i^1 - x), \\ k\left(y - \frac{(y_i^3 - y_i^2) \cdot (x - x_i^2)}{x_i^3 - x_i^2} - y_i^2\right) b(x - x_i^3) b(x_i^2 - x) \end{pmatrix}. \tag{6}$$

Through this wireframe rendering process, gradients can be propagated backward to both the class probabilities and geometric parameters of the graphic elements for joint optimization.

A CNN consisting of 3 convolutional layers, followed by a fully connected layer with sigmoid activation, is applied to the rasterization layer $\boldsymbol{I}$ to predict real/fake graphic layouts.

4 Experiments

The implementation is based on TensorFlow [45]. The network parameters are initialized from zero-mean Gaussian with standard deviation of 0.02 (the decoder branch for

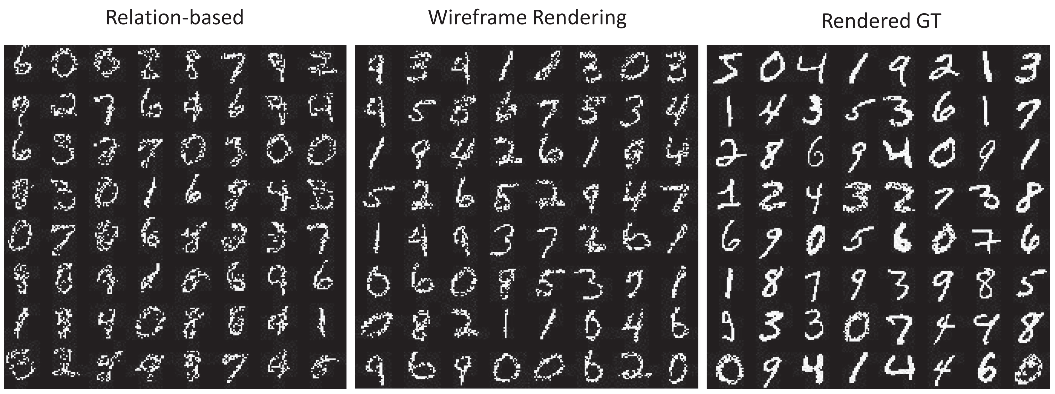


Fig. 3. Comparisons of MNIST digits from LayoutGAN with different discriminators and the real data.

predicting geometric parameters are initialized using standard deviation of 0.001). All the networks are optimized using Adam [46] with a fixed learning rate of 0.00002.

4.1 MNIST Digit Generation

As a toy problem, we generate MNIST layouts. MNIST is a handwritten digit database consisting of 60,000 training and 10,000 testing images. For each image, we extract the locations of 128 randomly-selected foreground pixels as the graphic representation so that digit generation can be formulated as generating of 2D point layouts. In Fig. 3, each image shows 8×8 digits rendered from point layouts. The left and middle images are the generated samples from the LayoutGAN with a relation-based discriminator and a wireframe rendering discriminator, respectively. The right image shows the digits rendered from ground-truth point layouts. One can see that the LayoutGAN with either discriminator captures various patterns. The wireframe rendering discriminator generates more compact, better-aligned point layouts.

For quantitative evaluation, we first train a multilayer perceptron network for digit classification, which achieves an accuracy of 98.91 percent on MNIST test set, and then use the classifier to compute the Inception Score (IS) [47] of 10,000 generated samples. In addition, we also report the Frchet Inception Distance (FID) [48]. As shown in Table 1, the LayoutGAN with wireframe rendering discriminator achieves better values in terms of both metrics than with the relation-based discriminator.

To shed light on the relational refinement process of LayoutGAN, we mark each input random point with a location-specific color, and track their refined locations in the generated layouts, as shown in Fig. 4. One can see that colors change gradually along the strokes, showing the network learns some contextually consistent local displacements of the points. Animations of these movements may be found in the supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2019.2963663.

TABLE 1
Quantitative Comparisons of Generated Digits

Methods	IS	FID
Relation	6.53	80.92
Wireframe	7.36	66.76
Real data	9.81	-

4.2 Document Layout Generation

One document page consists of a number of regions with different element types, such as heading, paragraph, table, figure, caption and list. Each region is represented by a bounding box. Modeling layouts of regions is critical for document analysis, retargeting, and synthesis [28]. In real document data, these bounding boxes are often carefully aligned along the canonical axes, and their placement follows some particular patterns, such as heading always appearing above paragraph or table. We first show some layout samples and their corresponding real document pages in Fig. 5.

In this experiment, we focus on one-column layouts with no more than nine bounding boxes that may belong to six possible classes as mentioned above. We are interested in how these layout patterns can be captured by our network. For training data, we collect totally around 25,000 layouts from real documents, in which elements are either left- or center-aligned with no overlaps with each other (except for captions with figures or tables). We also implemented a baseline method that represents the layout by semantic masks as used in Yang *et al.* [5] and Deka *et al.* [4]. Specifically, we render all elements into masks and train a DCGAN [8] to generate mask layouts in pixels. Then we extract the connected mask regions of each class and output enclosing bounding boxes of all regions. Fig. 6 shows some representative generation results (each row consists of 6 samples; high-resolution results can

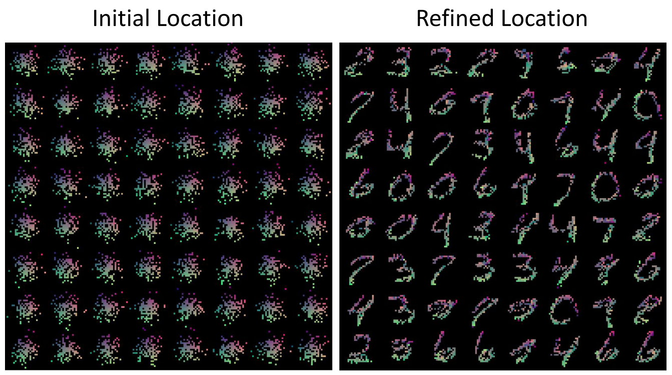


Fig. 4. Point tracing of location refinement process (best viewed in color).

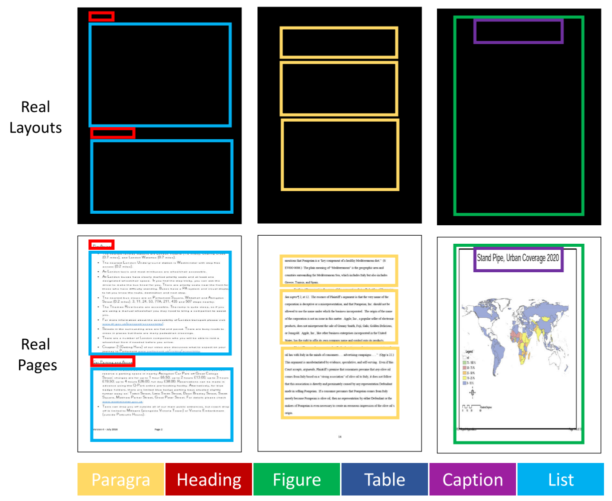


Fig. 5. Visualization of document layout samples and their corresponding real document pages.

be found in the Appendix, available online. The first row shows the results from DCGAN. DCGAN mixes a layout and its rendering in the generation process. When convolutional layers in DCGAN fails to perfectly replicate the rendering process (which is usually the case), it generates fuzzy and noisy label maps, making the extracted bounding boxes less consistent and less accurate. The third row shows the results from LayoutGAN with wireframe rendering discriminator. We retrieve the most similar real layouts from the training set in the last row as references. It can be seen that LayoutGAN can well capture different document layout patterns with clear definitions of instance-level semantic bounding boxes.

To validate the advantage of wireframe rendering discriminator, we compare the generated results with those from LayoutGAN with relation-based discriminator. As shown in the second row of Fig. 6, the latter can also capture different layout patterns but sometimes suffers from overlapping and misalignment issues. As the bounding boxes in real document layouts are either left- or center-aligned without overlapping, we propose two metrics to quantitatively measure the quality of generated layouts. The first one is overlap index, which is the percentage of total overlapping area among any two bounding boxes inside the whole page. Assuming s_i represents the normalized region area computed by the product of the bounding box width and height of element i, we compute the overlap index as $\sum_{i=1}^{N}\sum_{\forall j\neq i}\frac{s_i\cap s_j}{2}$, where $s_i \cap s_j$ denotes the overlapping area between element i and j in one layout. The second one is alignment index calculated by finding the minimum standard deviation of either left or center coordinates of all the bounding boxes. Denote x_i^L/x_i^C as the left/center coordinate of bounding box i, and $\bar{x}^L/\bar{x}^C$ represents the average left/center coordinate of all bounding boxes in one layout. The alignment index can be computed as $\min_{*\in[L,C]}\sqrt{\frac{\sum_{i=1}^{N}(x_i^*-\bar{x}^*)^2}{N}}$. Table 2 shows quantitative comparisons of real layouts and synthesized layouts from DCGAN and LayoutGAN with two different discriminators. One can see that LayoutGAN with wireframe rendering discriminator achieves lower value in both overlapping index and

TABLE 2
Spatial Analysis of Generated Document Layouts

Methods-(*Number of Self-attention*)	Overlap (%)	Alignment (%)
DCGAN	1.54	9.8
Relation-4 (default)	1.52	6.4
Wireframe-2	1.29	4.1
Wireframe-4 (default)	1.17	3.4
Wireframe-6	1.12	3.3
Real data	0.05	0.5

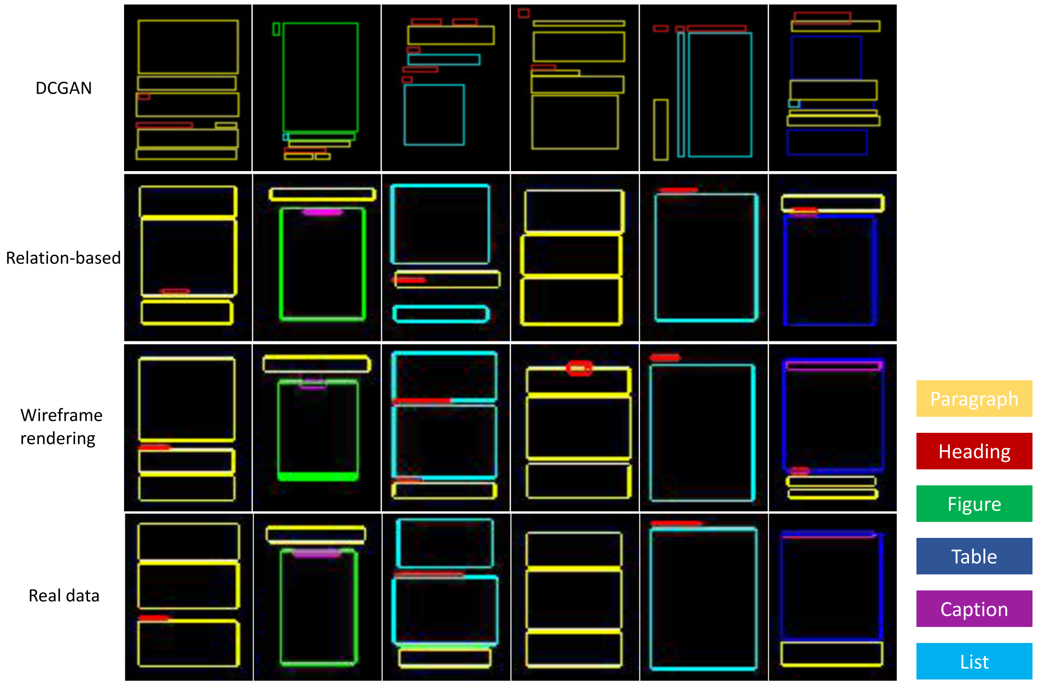


Fig. 6. Comparisons of document layouts from DCGAN, LayoutGAN with different discriminators and the real data.

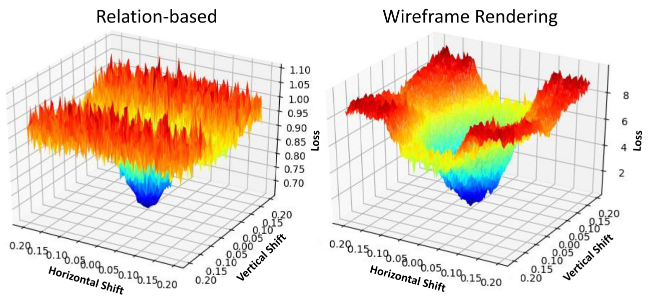


Fig. 7. Comparisons of discriminator loss landscapes.

alignment index than the one with relation-based discriminator and DCGAN, validating the superiority of the proposed wireframe rendering method for layout generation. In our experiments, we construct the stacked relation module in the generator with four cascaded self-attention blocks [42]. To validate our design choice, we further report the results of model variants comprised by different numbers (i.e., 2, 4 and 6) of self-attention blocks. As shown in Table 2, better synthesized results can be obtained from more steps of contextual feature refinement. Since no noticeable improvement can be observed by adding more blocks, we use four cascaded self-attention blocks throughout our experiments.

A similar conclusion about the superiority of the proposed wireframe rendering method can also be drawn by analyzing the loss functions. We add shift perturbations to the bounding boxes of real layouts and feed them to both discriminators to examine their loss behaviors. Fig. 7 visualizes the loss landscape of both discriminators corresponding to different absolute extent of shift perturbation (in normalized coordinate) in the horizontal and vertical direction for all elements in one layout. One can see that the loss surface w.r.t. shift perturbation of the wireframe rendering discriminator is much smoother than the one of relation-based discriminator. In addition, a perturbation recovery test is further implemented, in which we add some random perturbations (zero-mean Gaussian with the standard deviation of 0.05) to the

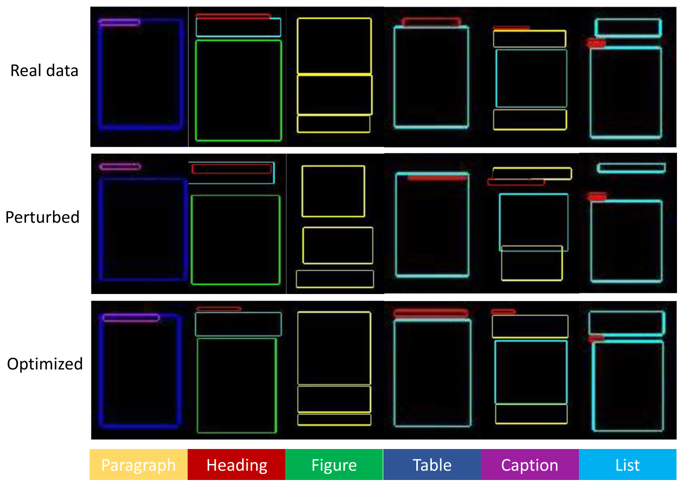


Fig. 9. Optimizing perturbed document layouts.

geometric parameters of bounding boxes of real layouts and train a LayoutGAN ($\sim 20k$ iterations) with the perturbed layouts as generator input to see if it can recover the real ones. Fig. 9 shows some results on the test set, our network can pull the displaced regions such as captions and headings back to the top of tables and paragraphs respectively, which confirms that the network successfully figures out the relations between different graphic elements, validating its effectiveness.

4.3 Clipart Abstract Scene Generation

We now consider generating scenes composed of a set of specific clipart elements. We use the abstract scene dataset [49] including boy, girl, glasses, hat, sun, and tree elements. To synthesize a reasonable abstract scene, we first use LayoutGAN to generate the layout of inner scene elements by representing each of them as a labeled bounding box with normalized center coordinates, width and height, and flip indicator. Then, we render the corresponding clipart image of each scene element onto a background image

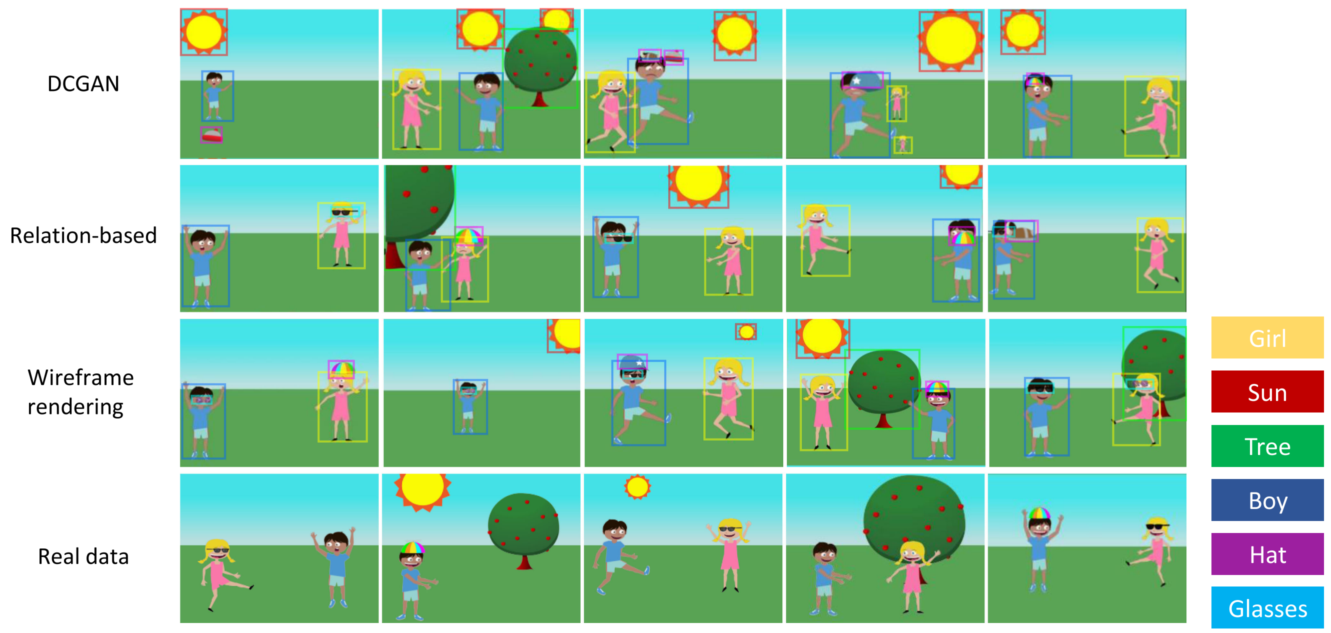


Fig. 8. Comparisons of Clipart abstract scenes from DCGAN, LayoutGAN with different discriminators, and the real data.

TABLE 3
User Study Results for Clipart Abstract Scenes

Methods	Excellent (%)	Fair (%)	Poor (%)
DCGAN	6.7	38.8	54.5
Relation	17.2	50.3	32.5
Wireframe	37.3	48.0	14.7

according to the predicted position (center coordinates), scale (width and height) and flip, forming a synthesized abstract scene. It is a challenging task as accurate pairwise or higher-order relations of objects are required for synthesizing reasonable scenes, for example, the glasses/hat should be exactly on the eyes/head of a boy/girl with proper scale and orientation.

In Fig. 8, the first three rows show the generated layouts and corresponding rendered scenes by DCGAN [8], LayoutGAN with relation-based discriminator and LayoutGAN with wireframe rendering discriminator respectively. The last row shows some samples rendered from ground-truth scene layouts. It can be seen that, compared to DCGAN and the LayoutGAN with relation-based discriminator, the one with wireframe rendering discriminator can capture pairwise relations of objects precisely, forming more meaningful scene layouts (for example, the glasses/hat accurately lies on the eyes/head of the person with varying scales and flips), validating the superiority of the proposed wireframe rendering strategy.

In addition, we conduct a user study involving 20 participants from our research organization, who were recruited by internal mail list and pass a pretest in which real abstract scenes should be appreciated among the synthesized ones, for a subjective evaluation of the results. Specifically, we randomly sample 30 abstract scenes synthesized by each of the above three models. A subject is asked to rate the synthetic scenes in terms of three scores (Excellent, Fair, Poor) by following three criteria: 1) whether they have coherent overall structures, 2) whether different objects are placed in reasonable relative positions, scales and flips, and 3) whether duplicate objects are avoided. We compute the rating percentage of all sampled scenes for each model. As shown in Table 3, the results of LayoutGAN with the wireframe rendering discriminator receive better ratings, which suggests that they are preferred by participants.

4.4 Tangram Graphic Design

A tangram is a geometric puzzle aiming to form specific shapes using seven pieces of 2D shapes without overlaps. The seven pieces include two large right triangles, one medium right triangle, two small right triangles, one square and one parallelogram, which can be assembled to form a square of side one unit and having area one square unit when choosing a unit of measurement. We collect 149 tangram graphic designs including animals, people and objects. In our experiments, we consider eight rotation/reflection poses for each piece. Given the seven pieces with each initialized by random location and ground-truth pose and class, the LayoutGAN with wireframe rendering discriminator is trained to refine their configurations jointly to form reasonable layouts.

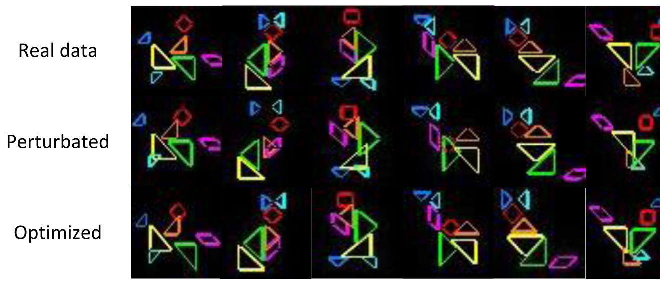


Fig. 10. Optimizing perturbed tangram graphic design.

Note that it is a challenging task due to the complex configurations and limited data.

We perform two experiments. The first one is a perturbation recovery test, in which we add some random spatial perturbations to seven pieces of real layouts and train a LayoutGAN with the perturbed layouts as generator input to see if it can recover the real ones. Fig. 10 shows that our network is able to pull back the displaced shapes to the right locations, confirming that the network successfully figures out the relations between different graphic elements.

We further train another LayoutGAN to generate tangram designs from purely random initialization. In Fig. 12, each row represents the sampled progressive generated results and the retrieved similar real tangrams. For comparison, we also present some results generated by DCGAN [8]. In addtion, we further implement a sequential model similar to Wang *et al.* [41], which generates a tangram layout iteratively, adding pieces one-by-one. In particular, the model generates a tangram layout by iteratively placing one piece at a time, where the location of the piece being placed is conditioned on the state of the existing pieces thus far. One can see from Fig. 11 that DCGAN cannot well model the spatial relations among different shapes while the sequential model suffers from accumulated errors. In contrast, the LayoutGAN can generate meaningful tangrams like fox and person, although others may be hard to interpret, as shown in the penultimate row.

4.5 Mobile App Layout Designs

We further conduct experiments on mobile app layout design, using the RICO dataset [4]. Each mobile app layout is a screenshot comprising several elements. Each element is labeled as one of several different possible categories, such as toolbar, text button, icon, image and text. The app layout design problem is to model the structural relations between different elements, and to arrange their positions

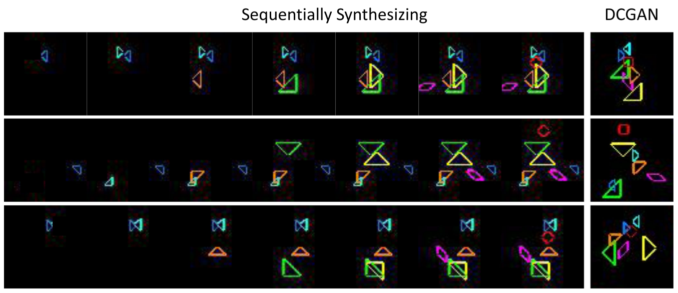


Fig. 11. Baseline results of tangram graphic design.

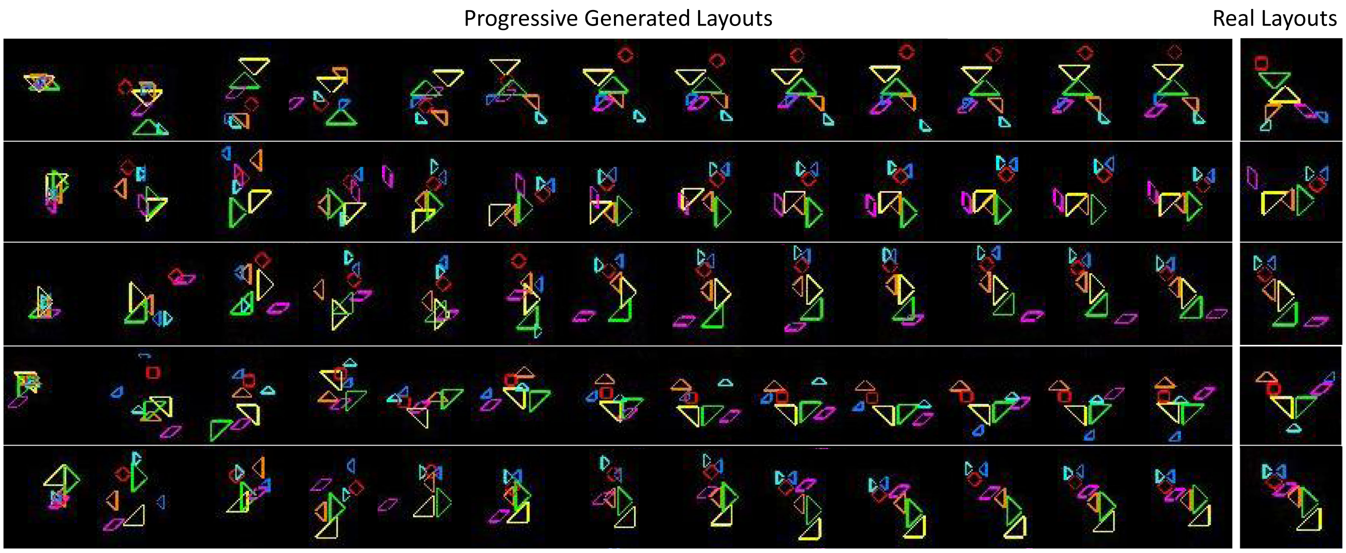


Fig. 12. Training progression of tangram graphic design generation (from left to right).

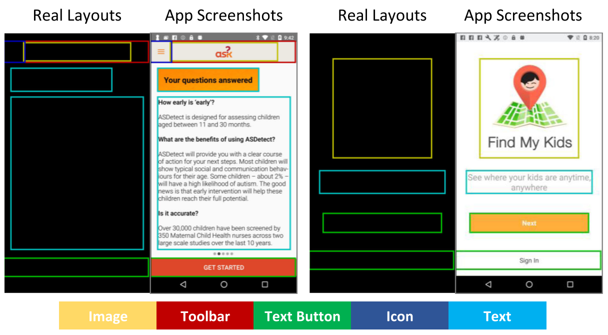
and sizes to form reasonable layouts following some particular patterns; for example, elements are often asymmetrically arranged, icons often appear inside toolbars, and so on. Each element in a specific category can be represented by a bounding box with a unique color. Fig. 13 provides some sample layouts and their corresponding real mobile app screenshots.

In this experiment, we consider mobile app layouts with no more than nine bounding boxes, and containing only elements belonging to the five possible classes listed above. These are the five most frequent classes in the dataset, and other classes were not frequent enough to train on. In Fig. 14, the first row shows layouts generated by DCGAN [8], and the last two rows show the results from LayoutGAN, along with most-similar real layouts retrieved from the training set. Both the wireframe and the corresponding mask layouts are provided for better visualization. One can see that compared to DCGAN, LayoutGAN better captures different mobile app layout patterns, producing well-designed mobile app layouts.

4.6 Webpage Layout Design From Hand-Drawn Sketches

Hand drawing is one of the simplest and most effective ways for developers and designers to quickly express and communicate design ideas, and generating graphic layouts directly from simple hand-drawn prototypes could significantly streamline the workflow for creating and testing designs. In this experiment, we consider the problem of webpage layout from sketches. We train LayoutGAN to convert low-fidelity hand-drawn sketches into well-aligned webpage layouts, as shown in Fig. 15. In particular, we first apply object detection methods to detect design elements from hand-drawn sketches by representing all elements as bounding boxes with different classes. Due to the variability in actual hand-drawn sketches and the localization inaccuracy introduced during the detection stage, the output bounding boxes from the detector are usually not well aligned. We then apply LayoutGAN to the detection results as input for producing well-aligned output. Specifically, we first add some random perturbations to the bounding boxes of real layouts and train a LayoutGAN with the perturbed layouts as generator input to recover the real ones. Due to data limitation, we also incorporate L1 distance computed by the geometric parameters of perturbed layouts and those of real layouts during training. Then, we apply the trained LayoutGAN to optimize detection results.

For training data, we start with the pix2code dataset [50], consisting of around 1,700 screenshots of synthetically generated websites for well-designed webpage layouts, and use the data by Kumar *et al.* [51] to obtain the corresponding hand-drawn-like sketches. Each webpage is comprised of



How early is 'early'?

ASDetect is designed for assessing children aged between 11 and 30 months.

What are the benefits of using ASDetect?

ASDetect will provide you with a clear course of action for your next steps. Most children will show typical social and communication behav-iours for their age. Some children – about 2% – will have a high likelihood of autism. The good news is that early intervention will help these children reach their full potential.

Is it accurate?

Over 30,000 children have been screened by 350 Maternal Child Health nurses across two large scale studies over the last 10 years.

Fig. 13. Visualization of mobile app layout samples and their corresponding real mobile app screenshots.

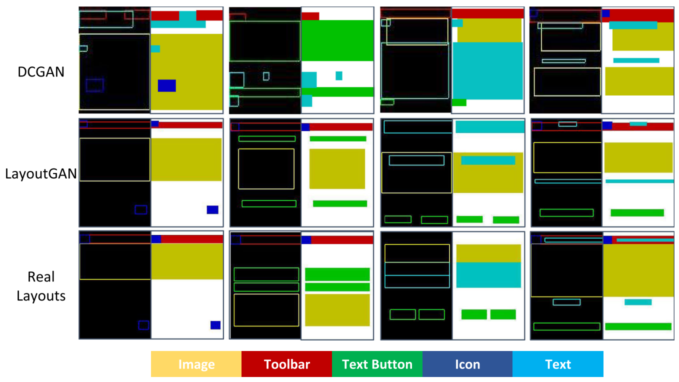


Fig. 14. Comparisons of mobile app layouts from DCGAN, LayoutGAN, and the real data.

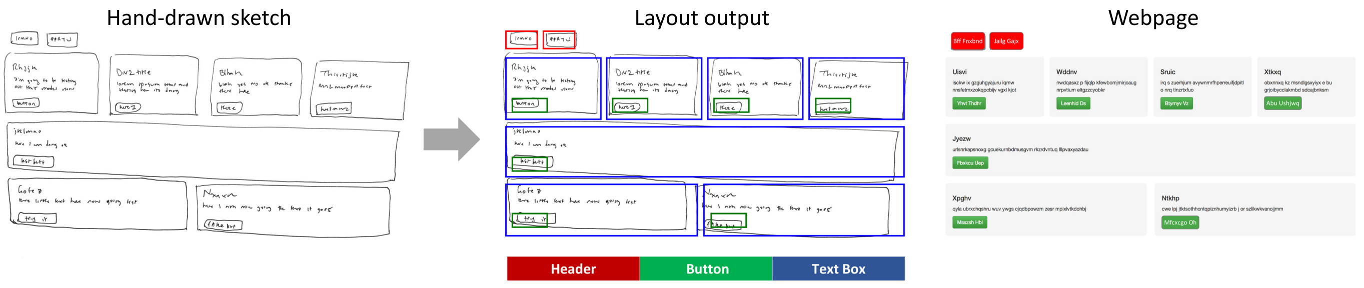


Fig. 15. Webpage layout optimization from hand-drawn sketches.

TABLE 4
Average Precision of Detection Results

IoU Threshold	mAP (%)	header (%)	text box (%)	button (%)
AP_{50}	99.84	100.00	100.00	99.52
AP_{75}	83.24	80.11	100.00	69.61
AP_{85}	53.39	35.12	90.80	34.24
AP_{90}	16.24	4.84	38.84	5.04

combinations of simple elements such as buttons, text boxes, and headers. For the detection model, we use Faster R-CNN [52] with an ImageNet [53] pre-trained Alexnet model [54] as backbone, for a lightweight implementation. We extract the enclosing bounding boxes of all elements (i.e., button, text box and header) inside the sketches as the ground-truth, and use 1500 sketches for training and 200 sketches for testing. To quantitatively evaluate the trained detector, we report the Average Precision (AP) under different Intersection-over-Union (IoU) thresholds on the test set. As shown in Table 4, the detector achieves high scores under the IoU threshold of 0.5, showing it can successfully detect different types of elements inside the sketches. While the classes containing small-size elements such as the header and button shows marked decrease in AP when increasing the IoU threshold, reflecting the common problem in detection, i.e., not able to locate small-size objects precisely, which may aggravate the misalignment of different bounding boxes in detection output.

We use LayoutGAN to further optimize bounding box layouts based on the detection results. The first two columns in Fig. 16 show input hand-drawn sketches and original detection results respectively. One can see that different bounding boxes output by the detector are not well aligned due to the originally misaligned hand-drawn input and localization inaccuracy by the detector. While the LayoutGAN is able to adjust the location and size of different elements from a global scope, producing optimized well-aligned layouts, as shown in the third column of Fig. 16.

For quantitative evaluations of the optimized layouts, we first report the AP under a harsh IoU threshold of 0.9 by taking well-designed webpage layouts as criterion, in which condition the detection results would have a low mAP close to zero. In addition, we compare with the regression model which refines detection results to bounding boxes of well-designed layouts through a MLP. We also report results of using CNN-based methods (the detection model) to directly regress bounding boxes of well-designed layouts from input hand-drawn sketches. As shown in Table 5, the LayoutGAN achieves higher scores compared to both regression baselines, showing the superiority of the LayoutGAN in optimizing layouts. As different bounding boxes for each class in each row of real webpage layouts are usually top-aligned, we further introduce a metric of alignment index to quantitatively evaluate the optimized layouts, which is calculated by first computing the standard deviation of top coordinates of all the bounding boxes in each row for each class and then getting the average values across rows. Table 6 provides the alignment index of original

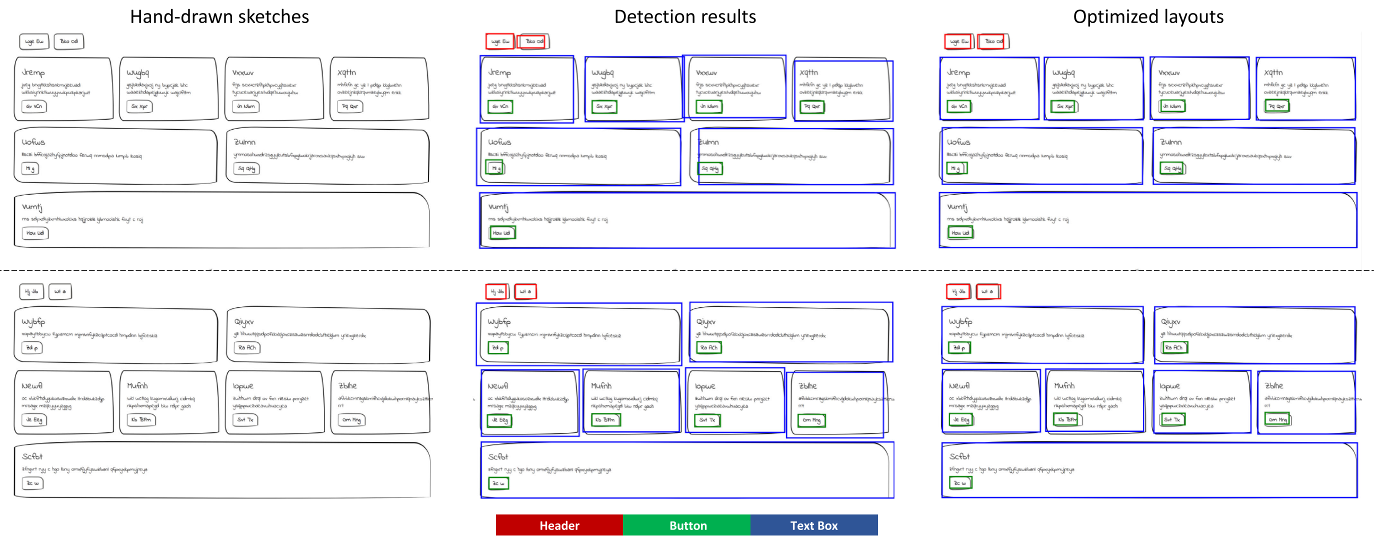


Fig. 16. Visualization of input hand-drawn sketches, detection results, and optimized webpage layouts.

TABLE 5
Average Precision of Optimized Layouts (IoU=0.9)

Methods	mAP (%)	header (%)	text box (%)	button (%)
CNN-based	33.26	36.37	54.01	9.42
MLP	40.50	30.98	78.28	12.25
LayoutGAN	78.13	64.89	84.85	84.66

TABLE 6
Spatial Analysis of Optimized Webpage Layouts

Methods	header (%)	text box (%)	button (%)
Detection	2.98	7.54	4.71
LayoutGAN	0.21	3.97	2.93
Well-designed	0.00	0.00	2.18

detection output and the optimized results from LayoutGAN. We also provide the values of well-designed webpage layouts for a reference in the last row. The LayoutGAN successfully decreases the alignment index based on the detection output and make the results closer to well-designed ones, validating the effectiveness of LayoutGAN for layout optimization.

5 Conclusion

In this paper, we proposed a novel LayoutGAN for generating layouts of relational graphic elements. Different from traditional GANs that generate images in pixel-level, the LayoutGAN can directly output a set of relational graphic elements. A novel differentiable wireframe rendering layer was proposed to rasterize the generated graphic elements to wireframe images, making it feasible to leverage CNNs as discriminator for better layout optimization from visual domain. Future works include determining the number of input elements and their classes and adding a content representation, such as text, icon and picture to each graphic element.

References

[1] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *Proc. Int. Conf. Learn. Representations*, 2018.

[2] C. L. Zitnick and D. Parikh, "Bringing semantics into focus using visual abstraction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 3009–3016.

[3] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, "Semantic scene completion from a single depth image," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 190–198.

[4] B. Deka *et al.*, "Rico: A mobile app dataset for building data-driven design applications," in *Proc. 30th Annu. ACM Symp. User Interface Softw. Technol.*, 2017, pp. 845–854.

[5] X. Yang, E. Yumer, P. Asente, M. Kraley, D. Kifer, and C. Lee Giles, "Learning to extract semantic structure from documents using multimodal fully convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5315–5324.

[6] M. Jaderberg *et al.*, "Spatial transformer networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2017–2025.

[7] J. Li, J. Yang, A. Hertzmann, J. Zhang, and T. Xu, "LayoutGAN: Generating graphic layouts with wireframe discriminators," in *Proc. Int. Conf. Learn. Representations*, 2019.

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015, *arXiv:1511.06434*.

[9] H. Zhang *et al.*, "StackGAN++: Realistic image synthesis with stacked generative adversarial networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1947–1962, Aug. 2019.

[10] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 613–621.

[11] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz, "MoCoGAN: Decomposing motion and content for video generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1526–1535.

[12] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee, "Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1696–1704.

[13] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 82–90.

[14] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 2625–2634.

[15] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 664–676, Apr. 2017.

[16] S. Reed and N. De Freitas, "Neural programmer-interpreters," 2015, *arXiv:1511.06279*.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[19] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7354–7363.

[20] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," 2014, *arXiv:1410.5401*.

[21] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with PixelCNN decoders," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4797–4805.

[22] A. van den Oord *et al.*, "WaveNet: A generative model for raw audio," in *Proc. 9th ISCA Speech Synthesis Workshop*, 2016, pp. 125–125.

[23] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," 2015, *arXiv:1511.06391*.

[24] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, Art. no. 6.

[25] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 77–85.

[26] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 40–49.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[28] N. Hurst, W. Li, and K. Marriott, "Review of automatic document formatting," in *Proc. 9th ACM Symp. Document Eng.*, 2009, pp. 99–108.

[29] R. Kumar, J. O. Talton, S. Ahmad, and S. R. Klemmer, "Bricolage: Example-based retargeting for web design," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2011, pp. 2197–2206.

[30] P. O'Donovan, A. Agarwala, and A. Hertzmann, "Learning layouts for single-page graphic designs," *IEEE Trans. Vis. Comput. Graphics*, vol. 20, no. 8, pp. 1200–1213, Aug. 2014.

[31] P. O'Donovan, A. Agarwala, and A. Hertzmann, "DesignScape: Design with interactive layout suggestions," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2015, pp. 1221–1224.

[32] X. Pang, Y. Cao, R. W. Lau, and A. B. Chan, "Directing user attention via visual flow on web designs," *ACM Trans. Graphics*, vol. 35, no. 6, 2016, Art. no. 240.

[33] A. Swearngin, M. Dontcheva, W. Li, J. Brandt, M. Dixon, and A. J. Ko, "Rewire: Interface design assistance from examples," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2018, Art. no. 504.

[34] Z. Bylinskii *et al.*, "Learning visual importance for graphic designs and data visualizations," in *Proc. 30th Annu. ACM Symp. User Interface Softw. Technol.*, 2017, pp. 57–69.

[35] P. O'Donovan, J. Lībeks, A. Agarwala, and A. Hertzmann, "Exploratory font selection using crowdsourced attributes," *ACM Trans. Graphics*, vol. 33, no. 4, 2014, Art. no. 92.

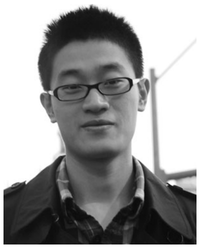
[36] P. O'Donovan, A. Agarwala, and A. Hertzmann, "Color compatibility from large datasets," *ACM Trans. Graphics*, vol. 30, no. 4, 2011, Art. no. 63.

[37] S. Qi, Y. Zhu, S. Huang, C. Jiang, and S.-C. Zhu, "Human-centric indoor scene synthesis using stochastic grammar," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 5899–5908.
[38] L.-F. Yu, S.-K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. J. Osher, "Make it home: Automatic optimization of furniture arrangement," *ACM Trans. Graphics*, vol. 30, no. 4, 2011, Art. no. 86.
[39] P. Merrell, E. Schkufza, Z. Li, M. Agrawala, and V. Koltun, "Interactive furniture layout using interior design guidelines," *ACM Trans. Graphics*, vol. 30, no. 4, 2011, Art. no. 87.
[40] M. Fisher, D. Ritchie, M. Savva, T. Funkhouser, and P. Hanrahan, "Example-based synthesis of 3D object arrangements," *ACM Trans. Graphics*, vol. 31, no. 6, 2012, Art. no. 135.
[41] K. Wang, M. Savva, A. X. Chang, and D. Ritchie, "Deep convolutional priors for indoor scene synthesis," *ACM Trans. Graphics*, vol. 37, no. 4, 2018, Art. no. 70.
[42] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, Art. no. 4.
[43] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.
[44] J. Johnson, A. Karpathy, and L. Fei-Fei, "DenseCap: Fully convolutional localization networks for dense captioning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4565–4574.
[45] M. Abadi *et al.*,"TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Design Implementation*, 2016, pp. 265–283.
[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[47] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2234–2242.
[48] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6626–6637.
[49] C. L. Zitnick, R. Vedantam, and D. Parikh, "Adopting abstract images for semantic scene understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 4, pp. 627–638, Apr. 2016.
[50] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proc. ACM SIGCHI Symp. Eng. Interactive Comput. Syst.*, 2018, Art. no. 3.
[51] A. Kumar, Automated front-end development using deep learning, 2018. [Online]. Available: https://github.com/ashnkumar/sketch-code
[52] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

Jianan Li received the BS degree from the Beijing Institute of Technology, Beijing, China, in 2013. He is currently working toward the PhD degree in the School of Optoelectronics, Beijing Institute of Technology, Beijing, China. From July 2015 to July 2017, he worked as a joint training PhD student with the National University of Singapore. From October 2017 to April 2018, he worked as an intern with Adobe Research. His research interests mainly include computer vision and real-time image/video processing.

Jimei Yang received the BS degree in electrical engineering and information science from the China Agricultural University, Beijing, China, in 2006, the MEng degree in pattern recognition and intelligent systems from the University of Science and Technology of China, Hefei, China, in 2009, and the PhD degree in computer science from the University of California, Merced, California. He is currently a senior research scientist with Adobe. He was a visiting PhD student in Artificial Intelligence Lab, University of Michigan, Ann Arbor, in 2015. From June 2007 to June 2009, he worked as a research assistant with Institute of Automation, Chinese Academy of Sciences.

Aaron Hertzmann received the BA degree in computer science and art/art history from Rice University, Houston, Texas, in 1996, and the PhD degree in computer science from New York University, New York, in 2001. He is currently a principal scientist with Adobe. He was a professor with the University of Toronto from 2003 to 2013, and has also worked with Pixar Animation Studios, University of Washington, Microsoft Research, Mitsubishi Electric Research Lab, Interval Research Corporation and NEC Research. He was an associate editor of the *ACM Transactions on Graphics*, for ten years. His awards include the MIT TR100 (2004), a Sloan Foundation Fellowship (2006), a Microsoft New Faculty Fellowship (2006), the CACS/AIC Outstanding Young CS Researcher Award (2010), and the Steacie Prize for Natural Sciences (2010), as well as several conference best paper awards. He is a fellow of the IEEE and ACM.


Jianming Zhang received the BS and MS degrees in mathematics from Tsinghua University, Beijing, China, in 2008 and 2011, respectively, and the PhD degree in computer science from Boston University, Boston, Massachusetts, in 2016. He is a research scientist with Adobe. His research area is computer vision and machine learning with a focus on deep learning, visual saliency, and image editing.

Tingfa Xu received the PhD degree from the Changchun Institute of Optics, Fine Mechanics and Physics, Changchun, China, in 2004. He is currently a professor with the School of Optoelectronics, Beijing Institute of Technology, Beijing, China. His research interests mainly include computer vision and optoelectronic imaging.