Constrained Graphic Layout Generation via Latent Optimization

Kotaro Kikuchi
Waseda University
Shinjuku-ku, Tokyo, Japan
kiku-koh@ruri.waseda.jp

Edgar Simo-Serra
Waseda University
Shinjuku-ku, Tokyo, Japan
ess@waseda.jp

Mayu Otani
CyberAgent
Shibuya-ku, Tokyo, Japan
otani_mayu@cyberagent.co.jp

Kota Yamaguchi
CyberAgent
Shibuya-ku, Tokyo, Japan
yamaguchi_kota@cyberagent.co.jp

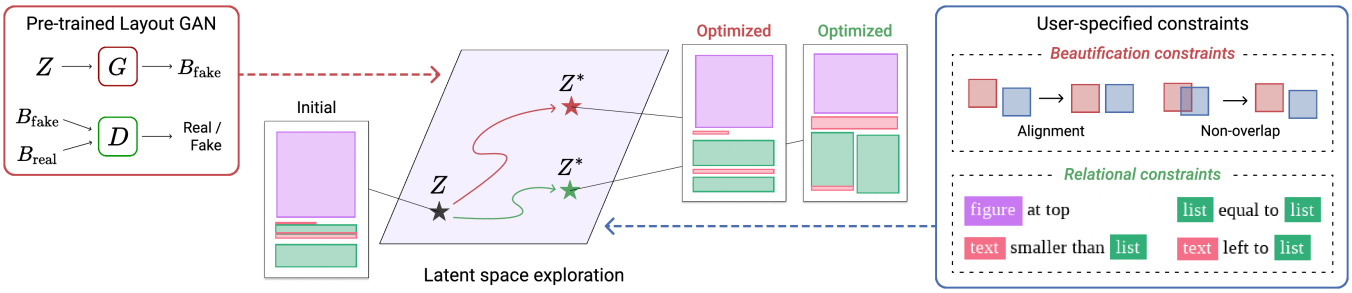

Figure 1: Overview of our Constrained Layout Generation via Latent Optimization (CLG-LO) framework. Given a pre-trained Generative Adversarial Network (GAN) for layout generation and user-specified constraints, CLG-LO explores the latent code to find a layout that satisfies the constraints. CLG-LO can reuse the same GAN for varying constraints without re-training.

ABSTRACT

It is common in graphic design humans visually arrange various elements according to their design intent and semantics. For example, a title text almost always appears on top of other elements in a document. In this work, we generate graphic layouts that can flexibly incorporate such design semantics, either specified implicitly or explicitly by a user. We optimize using the latent space of an off-the-shelf layout generation model, allowing our approach to be complementary to and used with existing layout generation models. Our approach builds on a generative layout model based on a Transformer architecture, and formulates the layout generation as a constrained optimization problem where design constraints are used for element alignment, overlap avoidance, or any other user-specified relationship. We show in the experiments that our approach is capable of generating realistic layouts in both constrained and unconstrained generation tasks with a single model. The code is available at https://github.com/ktrk115/const_layout.

CCS CONCEPTS

• **Human-centered computing** → *Interaction design process and methods*; • **Applied computing** → *Computer-aided design.*

MM '21, October 20–24, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3475497

KEYWORDS

layout generation, generative adversarial network, constrained optimization, latent space exploration

ACM Reference Format:
Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. 2021. Constrained Graphic Layout Generation via Latent Optimization. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21), October 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475497

1 INTRODUCTION

Visual media contents are organized using design layouts to facilitate the conveying of information. Design layout consists of the arrangement of the size and position of the elements to be displayed, and is a critical part of graphic design. In general, articles start with a text title, followed by headings and the main text, usually in a top to bottom order. Mobile user interfaces arrange navigation, images, texts, or buttons cleanly in a given display resolution with fluid layouts. The semantic relationships, priority, and reading order of elements is carefully decided by graphic designers while considering the overall visual aesthetics of the design. Inexperienced designers often face the difficulty of producing high-quality presentations while conveying the designated message and maintaining fundamental design considerations such as alignment or overlap. Design constraints can be internal, derived from the one's design experience and preference, or external, such as visual media regulations and client requirements. Automatic search of plausible layout candidates, such as we propose in this paper, can greatly aid in the design process.

Several attempts have been made to automatically generate graphic layouts in the computer graphics community [23, 24]. Recent studies [1, 12, 17] using unconstrained deep generative models have shown to be able to generate plausible layouts thanks to large scale datasets of design examples. Some work explicitly introduce design constraints like alignment or overlap avoidance by additional losses or conditioning [16, 18]. However, one drawback of integrating constraints in the learning objective is that a model must be fit to a new condition or a new loss when there appears a new constraint a user wishes to incorporate. We instead opt to perform the optimization in the latent space of the generative model, being complementary to and allowing for the usage of existing off-the-shelf models.

In this work, we propose a novel framework, which we call Constrained Layout Generation via Latent Optimization (CLG-LO), that defines constrained layout generation as a constrained optimization problem in the latent space of the model. An overview of the proposed framework is illustrated in Fig. 1. In our approach, we use a Generative Adversarial Network (GAN) trained in the unconstrained setting and model user specifications as a constrained optimization program. We optimize the latent code of the unconstrained model with an iterative algorithm to find a layout that satisfies the specified constraints. Our framework allows the user to use a single pre-trained GAN and incorporate various constraints into the layout generation as needed, eliminating the computationally expensive need of re-training of the model.

Although our approach can work with off-the-shelf generative layout models, in addition to CLG-LO framework, we also propose a Transformer [32] based layout GAN model, which we name LayoutGAN++. Relationships between elements can be well captured by Transformers in both the generator and the discriminator. With the help of representation learning of the discriminator through auxiliary layout reconstruction [19], LayoutGAN++ significantly improves the performance of the LayoutGAN [17] for unconstrained layout generation.

We validate our proposed methods using three public datasets of graphic layouts. We design two constrained generation settings similar to real use cases. In the unconstrained generation task, LayoutGAN++ obtains comparable or better results than the existing methods. Using LayoutGAN++ as the backend model, CLG-LO shows significant improvements in the constrained generation task.

We summarize our contributions as follows:

- A framework to generate layouts that satisfies given constraints by optimizing latent codes.
- An architecture and methodology for layout GAN that allows for stable training and generation of high-quality layouts.
- Extensive experiments and state-of-the-art results using public datasets for unconstrained and constrained layout generation.

2 RELATED WORK

2.1 Layout Generation

There has been several studies on generating layout, both with or without user specification. Classic optimization approaches [23, 24] manually designed energy functions with a large number of constraints that a layout should satisfy. Recent works have utilized neural networks to learn a generative model of layout. LayoutVAE trained two types of Variational Auto-Encoders (VAE) to generate bounding boxes to the given label set [12]. LayoutGAN trained relational generator by employing a wireframe renderer that rasterize bounding boxes and allows for training with a pixel-based discriminator [17]. Later, LayoutGAN was extended to include attribute conditioning [18]. Zheng et al. [37] reported a raster layout generator conditioned on the given images, keywords, and attributes. READ [27] trained a hierarchical auto-encoder to generate document layout structures. Lee et al. [16] proposed graph-based networks called Neural Design Networks (NDN) that explicitly infer element relations from partial user specification. Very recently, Gupta et al. [8] described a Transformer-based model to generate layout in various domains. Also, Arroyo et al. [1] reported a VAE model that generated layouts using self-attention networks. Apart from graphic design layouts, there has also been research on generating indoor scene layouts [10, 29, 35].

Our work considers both unconstrained generation [1, 8] and constrained generation [16, 18]. We build our unconstrained layout generator based on LayoutGAN [17], and apply user layout specification as constraints to a learned generator. Unlike NDN [16], we only need a single model to generate constrained layouts.

2.2 Latent Space Exploitation

With the recent progress in image synthesis using deep generative models [13, 14], much of the research utilizing the latent space have been made in the image domain. In real image editing, the mainstream research involves projecting the target image into the latent space and performing non-trivial image editing with user input on the learned manifold [2, 39, 40]. Pan et al. [25] also used the natural image priors learned by GAN and applied them to various image restoration tasks such as inpainting and colorization in a unified way. Menon et al. [21] search through the latent space of high-resolution facial photos to achieve super-resolution of low-quality photos.

The utilization of latent variables in deep generative models have been less studied in non-image domains. Umetani [31] proposed an interactive interface that uses a learned auto-encoder to find the shape of a 3D model by adjusting latent variables. Schrum et al. [30] proposed an interface consisting of interactive evolutionary search and direct manipulation of latent variables for the game level design. Chiu et al. [5] proposed a method to efficiently explore latent space in a human-in-the-loop fashion using a learned generative model, and validated it in the tasks of generating images, sounds, and 3D models.

Our layout generation approach shares the concept of latent space exploration, and we seek to find a latent representation of layout such that the resulting layout satisfies user-specified constraints.

3 APPROACH

Our goal is to generate a semantically plausible and high-quality design layout from a set of element labels and constraints specified by the user. We first train an unconstrained generative model of layout denoted *LayoutGAN++*, and later utilize the model for constrained generation tasks.

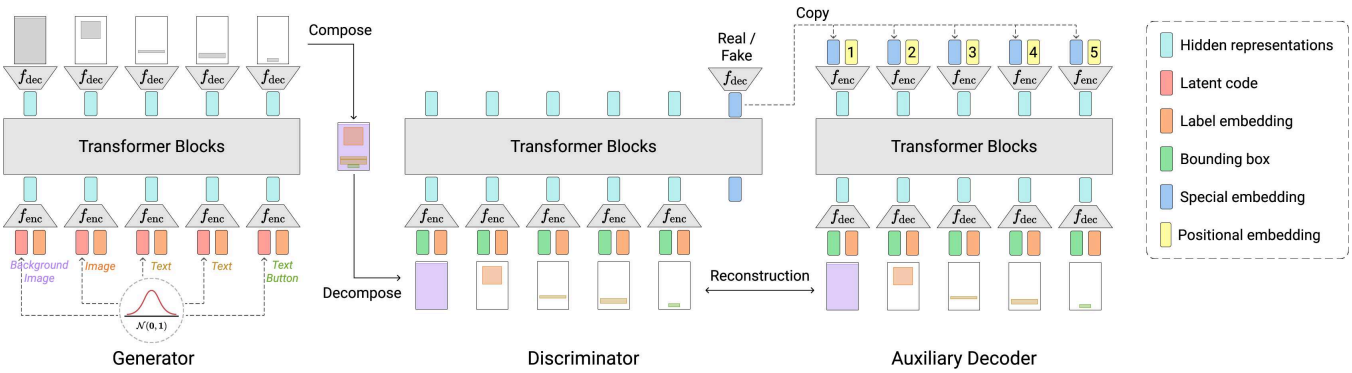

Figure 2: Overview of our proposed LayoutGAN++ model.

3.1 LayoutGAN++

In unconstrained generation, we take a set of elements and assign size and location to each element. We follow LayoutGAN [17] and formulate our model, which we refer *LayoutGAN++*, in the following. Formally, our generator $G\colon (Z, L) \mapsto B$ takes a set of randomly-generated codes $Z = \{\mathbf{z}_i\}_{i=1}^N$ and a conditional multiset of labels $L = \{\!\{l_i\}\!\}_{i=1}^N$ as input, and outputs a set of bounding boxes $B = \{\mathbf{b}_i\}_{i=1}^N$, where $\mathbf{b}_i \in [0,1]^4$ represents the position and size of the element in normalized coordinates. N is the number of elements in a layout, and the subscript i in Z, L, and B refers to the same i-th element. The definition of a label l depends on the dataset; *e.g.*, text or table elements in PubLayNet dataset. Our discriminator $D\colon (B, L) \mapsto r \in [0,1]$ takes the generated bounding boxes B and conditional labels L as input, and outputs a scalar value which quantifies the realism of layout, as well as attempts at reconstructing the given bounding boxes from the internal representation. We show in Fig. 2 the overall architecture of our model.

3.1.1 Generator. Our generator consists of the following:

$$\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{1}$$

$$\mathbf{h}_i = f_{\text{enc}}(\mathbf{z}_i, l_i; \theta), \tag{2}$$

$$\{\mathbf{h}'_i\} = \text{Transformer}(\{\mathbf{h}_i\}; \theta), \tag{3}$$

$$\mathbf{b}_i = f_{\text{dec}}(\mathbf{h}'_i; \theta), \tag{4}$$

where f_{enc}, f_{dec} are multi-layer perceptrons, $\mathbf{h}_i$ and $\mathbf{h}'_i$ are hidden representations for each element, and θ is the parameters for the generator. We adopt the Transformer block [32] to learn relational representation among elements, in contrast to LayoutGAN [34] that utilizes a dot product-based non-local block with a residual connection.

3.1.2 Discriminator. Our discriminator has a similar architecture to our generator.

$$\mathbf{h}_i = f_{\text{enc}}(\mathbf{b}_i, l_i; \phi), \tag{5}$$

$$\mathbf{h}'_{\text{const}} = \text{Transformer}(\mathbf{h}_{\text{const}}, \{\mathbf{h}_i\}; \phi), \tag{6}$$

$$y = f_{\text{dec}}(\mathbf{h}'_{\text{const}}; \phi), \tag{7}$$

where $\mathbf{h}_{\text{const}}$ is a special learnable embedding appended to the hidden element representations, $\mathbf{h}'_{\text{const}}$ is the corresponding output for the learnable embedding after the Transformer block, y is the quantity to evaluate the reality of the given input, and ϕ is the parameters of the discriminator. We do not employ the wireframe renderer of LayoutGAN [34], because we find that the raster domain discriminator becomes unstable with limited dataset size. We compare with LayoutGAN in our experiments.

3.1.3 Auxiliary Decoder. We empirically find that in well-aligned layout domains such as documents, the discriminator is trained to be sensitive to alignment and less sensitive to positional trends, *i.e.*, it only cares if the elements are aligned, and does not care about unusual layouts such as placing the header element at the bottom. Following the self-supervised learning of Liu et al. [19], we apply additional regularization to the discriminator so that the discriminator becomes aware of the positional trends. We add an auxiliary decoder to reconstruct the bounding boxes given to the discriminator from the internal representation $\mathbf{h}'_{\text{const}}$:

$$\mathbf{h}_i = f_{\text{enc}}(\mathbf{h}'_{\text{const}}, \mathbf{p}_i; \xi), \tag{8}$$

$$\{\mathbf{h}'_i\} = \text{Transformer}(\{\mathbf{h}_i\}; \xi), \tag{9}$$

$$\hat{\mathbf{b}}_i, \hat{l}_i = f_{\text{dec}}(\mathbf{h}'_i; \xi), \tag{10}$$

where $\mathbf{p}_i$ is a learnable positional embedding initialized with the uniform distribution of $[0,1]$, $\hat{\mathbf{b}}_i \in \hat{B}$ is a reconstructed bounding box, $\hat{l}_i \in \hat{L}$ is a reconstructed label, and ξ is the parameters of the auxiliary decoder.

3.1.4 Training objective. The objective function of our model is the following:

$$\min_{\theta} \max_{\phi,\xi} E_{(B,L)\sim P_{\text{data}}} \Big[D(B, L; \phi) - \mathcal{L}_{\text{rec}} \Big(B, L, \hat{B}(\phi, \xi), \hat{L}(\phi, \xi) \Big) \Big] + E_{Z\sim\mathcal{N}, L\sim P_{\text{data}}} \big[1 - D\big(G(Z, L; \theta), L; \phi\big) \big] \tag{11}$$

where we denote the reconstruction loss by $\mathcal{L}_{\text{rec}}$. The reconstruction loss measures the similarity between two sets of bounding boxes and labels, and we employ mean squared error for bounding boxes, and cross entropy for labels. We compute the reconstruction loss by first sorting the bounding boxes in lexicographic order of the ground-truth positions [4].

3.2 Constrained Layout Generation via Latent Optimization (CLG-LO)

Let us consider when there are user-specified constraints, such as *an element A must be above an element B*. From the perspective of the generator, such constraints restricts the available output space. We formulate the generation with user specification in a constrained optimization problem. Given a pre-trained generator $\hat{G}$ and discriminator $\hat{D}$, and a set of constraints C, we define the constrained minimization problem regarding latent codes Z:

$$\begin{aligned} \min_{Z} \quad & -\hat{D}\big(\hat{G}(Z, L), L\big) \\ \text{s.t.} \quad & c_n\big(\hat{G}(Z, L)\big) = 0, \quad n = 1, \ldots, |C|. \end{aligned} \tag{12}$$

The intuition is that we seek to find bounding boxes that looks as realistic as possible to the discriminator and satisfies the user-specified constraints. Once the optimal latent codes Z^* is found, we can obtain bounding boxes B^* that satisfy the constraints as follows:

$$B^* = \hat{G}(Z^*, L). \tag{13}$$

We use the augmented Lagrangian method [22], which is one of the widely used algorithms for solving nonlinear optimization problems. In this method, the constrained problem is transformed into an unconstrained problem that optimizes the augmented Lagrangian function, which combines the Lagrangian and penalty functions. Let us rewrite $f(Z) = -\hat{D}\big(\hat{G}(Z, L), L\big)$ and $h_n(Z) = c_n\big(\hat{G}(Z, L)\big)$ in Eq. (12) for brevity, then we define the following augmented Lagrangian function L_A,

$$L_A(Z; \lambda, \mu) = f(Z) + \sum_{n=1}^{|C|} \lambda_n h_n(Z) + \frac{\mu}{2} \sum_{n=1}^{|C|} h_n(Z)^2, \tag{14}$$

where λ are the Lagrange multipliers and $\mu > 0$ is a penalty parameter to weight the quadratic functions.

In this method, the Lagrange multipliers are updated according to the extent of constraint violation, and the penalty parameter is gradually increased to make the impact of the constraints larger. Let k be the current iteration, the update equations are expressed as:

$$\lambda_n^{k+1} = \lambda_n^k + \mu_k h_n(Z_k) \tag{15}$$

$$\mu_{k+1} = \alpha \mu_k, \tag{16}$$

where α is a predefined hyperparameter.

Algorithm 1 summarizes the procedure of our method. We repeat the main loop until the amount of constraint violation is sufficiently small or the iteration count reaches the maximum number of iterations $k_{\max}$. We set $\alpha = 3$, $\mu_0 = 1$, $\lambda^0 = \mathbf{0}$, and $k_{\max} = 5$ in the experiments. For the inner optimizer, we use either Adam [15] with a learning rate of 0.01 or CMA-ES [9] with a initial sigma value of 0.25, and both run for 200 iterations. We compare in Sec 4.4 which optimizer yields a better solution.

In practice, optimizing the output value of the discriminator directly may yield an adversarial example, *i.e.*, the discriminator considers it as the real, but perceptually degraded. To avoid this, we clamp the output value of the discriminator based on a certain threshold. Specifically, we use $f(Z_0)$ as the threshold, and $f'(Z) = \max\big(f(Z) - f(Z_0), 0\big)$ instead of $f(Z)$ in Eq. (14).

ALGORITHM 1: Constrained layout generation via latent code optimization

Input: pre-trained generator $\hat{G}$, pre-trained discriminator $\hat{D}$, labels L, constraints C, initial Lagrange multipliers λ^0, initial penalty parameter μ_0
Output: bounding boxes B^*

$Z_0 \leftarrow Z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$k \leftarrow 0$
repeat
  `// Inner optimization (Eq. (14))`
  $Z^* \leftarrow \mathrm{argmin}_Z\, L_A(Z; \lambda^k, \mu_k, \hat{G}, \hat{D}, L, C)$ starting at Z_k
  Update the Lagrange multipliers by Eq. (15) to obtain λ^{k+1}
  Update the penalty parameter by Eq. (16) to obtain μ_{k+1}
  $Z_k \leftarrow Z^*$
  $k \leftarrow k + 1$
until *stopping criteria is fulfilled*;
$B^* \leftarrow \hat{G}(Z^*, L)$
return B^*

Table 1: Statistics of the datasets used in our experiments and the splits using for evaluation.

Dataset	# label types	Max. # elements	# train.	# val.	# test.
Rico [7, 20]	13	9	17,515	1,030	2,061
PubLayNet [38]	5	9	160,549	8,450	4,226
Magazine [37]	5	33	3,331	196	392

4 EXPERIMENTS

We evaluate the proposed method on both unconstrained and constrained layout generation tasks. We first describe the datasets and evaluation metrics, and then explain the experimental setup for each task.

4.1 Dataset

We evaluate layout generation on different types of graphic designs. We use three publicly available datasets: Rico [7, 20] provides UI designs collected from mobile apps, PubLayNet [38] compiles a dataset of document images, and Magazine [37] collects magazine pages. Following the previous studies [16, 17], we exclude elements whose labels are not in the 13 most frequent labels in the Rico dataset, and exclude layouts with more than 10 elements in both the Rico and PubLayNet datasets. For the PubLayNet dataset, we use 95% of the official training split for training, the rest for validation, and the official validation split for testing. For Rico and Magazine, since there is no official data split, we use 85% of the dataset for training, 5% for validation, and 10% for testing. We summarize the statistics of the datasets in Table 1.

4.2 Evaluation Metrics

We use four metrics to measure the quality of the generated layouts: Fréchet Inception Distance (FID) [11], Maximum Intersection over Union (IoU), Alignment, and Overlap.

Table 2: Comparison of FID scores computed using feature extractors trained with various objectives. In particular we compare feature extractors trained with classification loss (Class), reconstruction loss (Recon), and a combination of both (Class+Recon). We compute the FID score between real layouts and variants that have added noise, have been vertically flipped, and nearest neighbors from the validation set.

Layout variants	Class	Recon	Class+ Recon
Real	-	-	-
Added noise	186.64	37.99	127.57
Vertically flipped	3.37	97.91	100.34
Nearest neighbour	0.29	12.52	11.80

4.2.1 FID. To compute FID, we need to define the representative features of layouts. We follow the approach of Lee et al. [16], and train a neural network to classify between real layouts and noise added layouts, and use the intermediate features of the network. One difference from [16] is that we incorporate the auxiliary decoder in Sec 3.1.3 learning such that the trained network is aware of both alignment and positions. In Table 2, we show a comparison of FIDs across networks learned with different objectives; *Class* is real/fake classification only, *Recon* is auxiliary reconstruction only, and *Class+Recon* is learned with both objectives. The combination of both objectives improves the sensitivity to different layout arrangements.

4.2.2 Maximum IoU. Maximum IoU is defined between two collections of generated layouts and references. We first define IoU based similarity between two layouts $B = \{\mathbf{b_i}\}_{i=1}^{N}$ and $B' = \{\mathbf{b'_i}\}_{i=1}^{N}$. We consider the optimal matching between B and B', then compute the average IoU of bounding boxes. Let $\pi \in \mathcal{S}_N$ be a one-by-one matching, and $\mathcal{S}_N$ be a set of possible permutations for size N. Note that we only consider matches between two bounding boxes with the same label, *i.e.*, $l_i = l_{\pi(i)} (1 \le i \le N)$. The similarity with respect to the optimal matching is computed as

$$g_{\mathrm{IoU}}(B, B', L) = \max_{\pi \in \mathcal{S}_N} \frac{1}{N} \sum_{i=1}^{N} \mathrm{IoU}(\mathbf{b}_i, \mathbf{b'}_{\pi(i)}), \tag{17}$$

where $\mathrm{IoU}(\cdot,\cdot)$ computes IoU between bounding boxes. To evaluate the similarity between generated layouts $\mathcal{B} = \{B_m\}_{m=1}^{M}$ and references $\mathcal{B}' = \{B'_m\}_{m=1}^{M}$, we compute the average similarity on the optimal matching:

$$\mathrm{MaxIoU}(\mathcal{B}, \mathcal{B}', \mathcal{L}) = \max_{\pi \in \mathcal{S}_M} \frac{1}{M} \sum_{m=1}^{M} g_{\mathrm{IoU}}(B_m, B'_{\pi(m)}, L_m), \tag{18}$$

where we only consider matches between two layouts with an identical label set, *i.e.*, $L_m = L_{\pi(m)} (1 \le m \le M)$. We use the solver [6] provided by SciPy [33] to solve the assignment problems.

4.2.3 Alignment and overlap. We use the *Alignment* and *Overlap* metrics used in the previous work [18]. We modify the original metrics by normalizing with the number of elements N.

4.3 Unconstrained Layout Generation

4.3.1 Setup. We use LayoutGAN [17] and NDN [16] as baselines. Although LayoutGAN is intended for the unconditional setting, we adapt the model to be conditioned on a label set input. We refer to the model using the wireframe rendering discriminator as **LayoutGAN-W** and the one using the relation-based discriminator as **LayoutGAN-R**. NDN first generates the position and size relations between elements, then generates bounding boxes based on the relations, and finally modifies the misalignment of the boxes. We denote it as **NDN-none** to match the designation in their paper, as our setting does not specify the relations. We reimplement all the baselines as since the official codes for the baselines are not publicly available[1]. We implement our LayoutGAN++ with PyTorch [26]. We train the model using the Adam optimizer with 200,000 iterations with a batch size of 64 and a learning rate of 1e-5, taking six hours with a GPU of NVIDIA GeForce RTX 2080Ti. Our Transformer modules consist of 8 blocks, and in each block, we set the input/output dimension to 256, the dimension of the hidden layer to 128, and the number of multi-head attentions to 4.

4.3.2 Results. We summarize the quantitative comparison in Table 3 and the qualitative comparison in Fig. 3. Since all the comparison methods are stochastic, we report the mean and standard deviation of five evaluations with the same trained model. Regarding LayoutGAN [17], we find that LayoutGAN-W is unstable to train, and failed to reproduce the results as good as in their paper despite our efforts, which is similarly reported in the recent studies [1, 8]. Our results show that LayoutGAN-R is much stable to train, and outperforms LayoutGAN-W. Our LayoutGAN++ achieves comparable to or better results than the current state-of-the-art method NDN-none [16], in particular, results on the Rico dataset are similar, while results on the PubLayNet dataset and Magazine dataset are favourable to our approach.

4.4 Layout Generation with Beautification Constraints

The goal of this setting is to generate a well-aligned layout with no overlapping, which can serve as a post-processing to beautify the result of the unconstrained layout generation. We conduct the experiment with the PubLayNet dataset, in which most of the layouts are aligned and have little overlap.

[1]The authors of LayoutGAN provide only the code for point layout experiment in https://github.com/JiananLi2016/LayoutGAN-Tensorflow, not for bounding boxes.

Table 3: Quantitative comparison of unconstrained layout generation. The values of Alignment and Overlap are multiplied by 100× for visibility. Comparisons are provided on three different datasets (Rico, PubLaynet, and Magazine). For reference, the FID and Max. IoU computed between the validation and test data, and the Alignment and Overlap computed with the test data are shown as *real data*.

Dataset	Rico				PubLayNet				Magazine			
Model	FID ↓	Max. IoU ↑	Alignment ↓	Overlap ↓	FID ↓	Max. IoU ↑	Alignment ↓	Overlap ↓	FID ↓	Max. IoU ↑	Alignment ↓	Overlap ↓
LayoutGAN-W [17]	162.75±0.28	0.30±0.00	0.71±0.00	174.11±0.22	195.38±0.46	0.21±0.00	1.21±0.01	138.77±0.21	159.20±0.87	0.12±0.00	**0.74**±0.02	188.77±0.93
LayoutGAN-R [17]	52.01±0.62	0.24±0.00	1.13±0.04	69.37±0.66	100.24±0.61	0.24±0.00	0.82±0.01	45.64±0.32	100.66±0.35	0.16±0.00	1.90±0.02	111.85±1.44
NDN-none [16]	**13.76**±0.28	0.35±0.00	**0.56**±0.03	**54.75**±0.29	35.67±0.35	0.31±0.00	0.35±0.01	**16.50**±0.29	23.27±0.90	0.22±0.00	1.05±0.03	**30.31**±0.77
LayoutGAN++	14.43±0.13	**0.36**±0.00	0.60±0.12	59.85±0.59	**20.48**±0.29	**0.36**±0.00	**0.19**±0.00	22.80±0.32	**13.35**±0.41	**0.26**±0.00	0.80±0.02	32.40±0.89
Real data	4.47	0.65	0.26	50.58	9.54	0.53	0.04	0.22	12.13	0.35	0.43	25.64

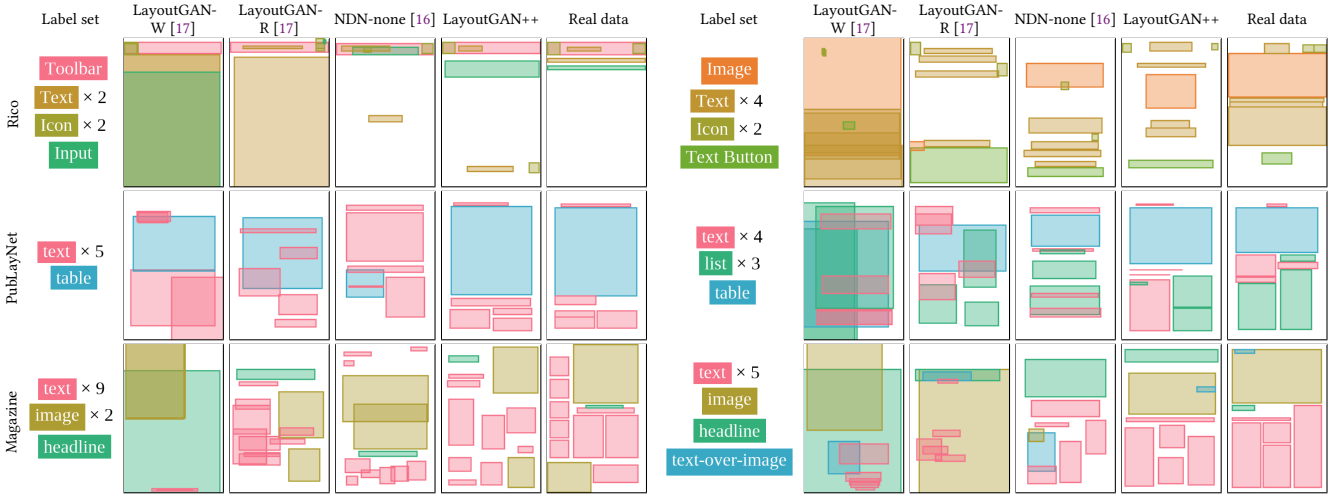

Figure 3: Qualitative comparison of unconstrained layout generation. Label set indicates the total number of labels and their type for each conditional generation result. On the right we show the real data from which the label set was taken.

4.4.1 Constraints. Let g_{align} be the function that computes the Alignment metric, we express the alignment constraint as

$$c_{\text{align}}(B) = \max\left(g_{\text{align}}(B) - \tau, 0\right), \tag{19}$$

where τ is a threshold parameter. We set $\tau = 0.004$ in our experiment. We use the Overlap metric as the non-overlapping constraint c_{ovrlp}.

4.4.2 Setup. We use a pre-trained LayoutGAN++ model within our proposed CLG-LO framework to perform the constrained task. We follow the same settings as in Section 4.3 for training LayoutGAN++. We compare two different inner optimizers, Adam [15] and CMA-ES [9]. The mean runtime for CLG-LO was 13.6 seconds with Adam (SD: 11.2) and 1.45 seconds with CMA-ES (SD: 1.75).

Since there is no directly comparable methods in the literature for this setting, we design a baseline called **CAL** that uses constraints as additional losses, referring to the similar work [18]. To instantiate CAL, we train LayoutGAN++ with both the alignment constraint c_{align} and the non-overlapping constraint c_{ovrlp} added to the generator objective, which encourages a generated layout that satisfies the constraints, but does not explicitly enforce them.

4.4.3 Results. We summarize the quantitative comparison in Table 4. The base model is LayoutGAN++ without beautification. We

Table 4: Quantitative results with beautification constraints. Base model refers to the unconstrained LayoutGAN++. The values of Alignment and Overlap are multiplied by 100× for visibility.

Model	FID ↓	Max. IoU ↑	Alignment ↓	Overlap ↓
Base model	20.48±0.29	0.36±0.00	0.19±0.00	22.80±0.32
CAL	**13.31**±0.17	**0.38**±0.00	0.16±0.00	14.27±0.19
CLG-LO w/ Adam	21.79±0.38	0.36±0.00	0.16±0.00	1.18±0.04
CLG-LO w/ CMA-ES	22.97±0.38	0.36±0.00	**0.14**±0.00	**0.02**±0.00

can see that CAL performs better in terms of Alignment and Overlap than the baseline, thanks to the added losses. FID and Maximum IoU are also improved, which may be due to the inductive bias expressed as the added losses, making GAN easier to train. Our CLG-LO further improves Alignment and Overlap significantly with almost no degradation in terms of FID and Maximum IoU. As for the choice of inner optimizer, CMA-ES seems to perform better than Adam. We suspect that due to the augmented Lagrangian function (Eq. (14)) having many local solutions, and thus a population-based global gradient-free optimization method, *e.g.*, CMA-ES, is more suitable than a gradient-based method, *e.g.*, Adam.

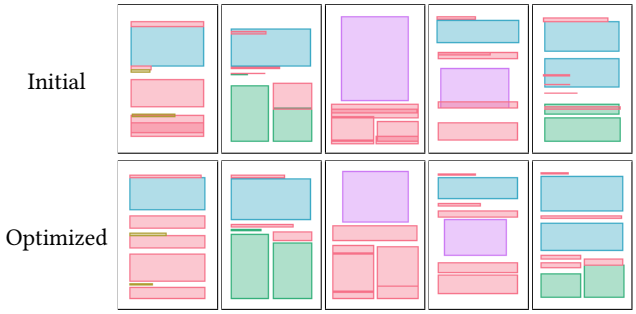

Figure 4: Qualitative results with beautification constraints for CLG-LO w/ CMA-ES. Initial unconditioned generation results are shown in the top row and the optimized results are shown in the bottom row.

We show the optimization results by CLG-LO using CMA-ES as the inner optimizer in Fig. 4. Our framework successfully found aligned and non-overlapping layouts. We have set the initial sigma parameter of CMA-ES smaller to explore around the initial latent code, which leads to the optimized layout not changing significantly from the initial layout.

4.5 Layout Generation with Relational Constraints

In this setting, we consider a scenario where the user specifies the location and size relationships of elements in the layout. We consider three size relations, *smaller*, *larger* and *equal*, and five location relations, *above*, *bottom*, *left*, *right*, and *overlap*. We also define the relation to the canvas, *e.g.*, positioning at the top of the canvas. We determine the relations from the ground-truth layout and use its subset as constraines. We change percentages of the relations used as constraints and report the rate of violated constraints.

4.5.1 Constraints. The size constraint c_{size} is defined as the sum of cost functions of all size relations. For example, suppose the user specifies that the j-th element has to be larger than the i-th element, then the cost function of *larger* relation is defined by:

$$g_{\text{lg}}(\mathbf{b}_i, \mathbf{b}_j) = \max\big((1+\gamma)a(\mathbf{b}_i) - a(\mathbf{b}_j), 0\big), \tag{20}$$

where $a(\cdot)$ is a function that calculates the area of a given bounding box, and γ is a tolerance parameter shared across the size relations. We set $r = 0.1$ in our experiment.

We also define the location constraint c_{loc} in the same way. For example, suppose the user specifies that the j-th element has to be above the i-th element, then the cost function of *above* relation is defined by:

$$g_{\text{ab}}(\mathbf{b}_i, \mathbf{b}_j) = \max\big(y_{\text{b}}(\mathbf{b}_j) - y_{\text{t}}(\mathbf{b}_i), 0\big), \tag{21}$$

where $y_{\text{t}}(\cdot)$ and $y_{\text{b}}(\cdot)$ are functions that return the top and bottom coordinates of a given bounding box, respectively.

4.5.2 Setup. We compare our CLG-LO against NDN [16]. In CLG-LO, we use CMA-ES for the inner optimizer, as it worked well in the experiments with beautification constraints. The rest of the settings follow the experiment with beautification constraints, but for a fair comparison, we did not use the beautification constraints themselves. The mean runtime for CLG-LO was 1.96 seconds (SD: 3.48).

4.5.3 Results. We show the qualitative results in Fig. 5 and the quantitative comparison in Table 5. We report the results for a setting that uses 10% of all relations in Table 5, which is what we believe would be representative of a realistic usage scenario. A typical example that uses roughly 10% relations is the upper left one in Fig. 5. Our CLG-LO performed comparable to or better than NDN, and in particular showed significant improvement in the constraint violation metric. This is as to be expected because NDN does not guarantee the inferred result satisfies the constraints, whereas our method tries to find a solution that satisfies as many of the constraints as possible through iterative optimization.

We also show in Fig. 6 the experimental results of varying the percentage of relations used. We can find that NDN performs better as increasing the number of relations used, which is reasonable since its layout generation module is trained with the complete relational graph of the ground-truth layout. On the other hand, our CLG-LO performs unfavorably as increasing the number of relations used, because it becomes harder to find a solution that satisfies the constraints. A practical remedy when no solution is found could be to store a layout for each iteration of the main loop in Algorithm 1, and let the user choose one based on the trade-off between constraint satisfaction and layout quality. We note, however, that our method performs best in realistic scenarios where the number of user-specified relations is few.

5 CONCLUSIONS AND DISCUSSION

In this paper, we proposed a novel framework called Constrained Layout Generation via Latent Optimization (CLG-LO), which performs constrained layout generation by optimizing the latent codes of pre-trained GAN. While existing works treat constraints as either additional objectives or conditioning, requiring re-training when unexpected constraints are involved, our framework can flexibly incorporate a variety of constraints using a single unconstrained GAN. While our approach is applicable to most generative layout design models, we also present a new layout generation model called LayoutGAN++ that is able to outperform existing approaches in unconditioned generation. Experimental results on both unconstrained and constrained generation tasks using three public datasets support the effectiveness of the proposed methods.

While our approach is able to significantly outperform existing approaches in many cases, given the non-convexity and complexity of the optimization problem as the objective and constraint functions in Eq. (12) involve a complex nonlinear neural network, we have no guarantees on the convergence of the approach. When the number of constraints becomes large (Figure 6), the optimizer can have issues finding a good solution, and underperform existing approaches. However, in general, most users will not specify very large number of constraints, and in those situations, our approach significantly outperforms existing approaches. We believe that this effect can be mitigated by improving the optimization approach itself, using piece-wise convex approximations, or improving the initialization of the optimization variables. It may also be practical to design an interaction that asks the user to remove or change difficult constraints.

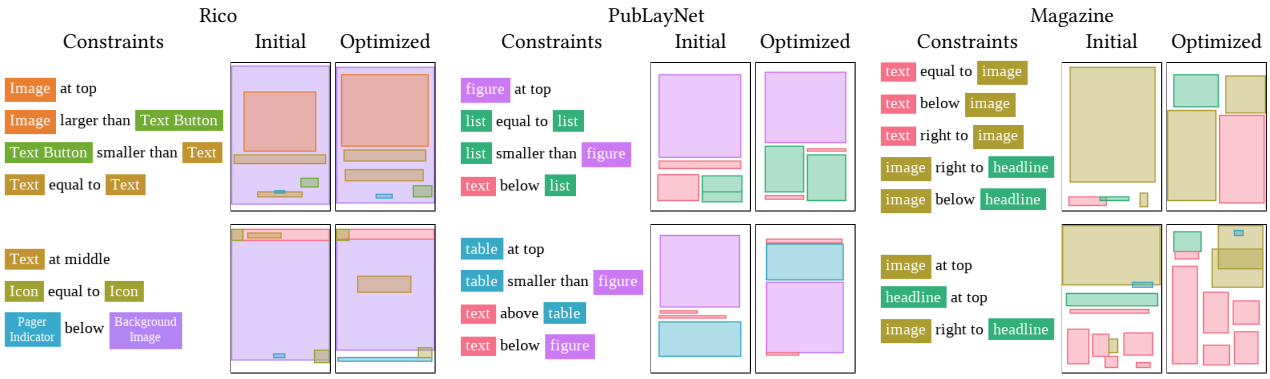

Figure 5: Qualitative results with relational constraints for the three datasets for our prposed CLG-LO w/ CMA-ES. In each column, for each result we show the constraints on the left, the initial unconstrained generation result in the middle, and the optimized result on the right.

Table 5: Quantitative results with relational constraints when 10% of all the relational constraints are used. The values of Alignment are multiplied by 100× for visibility.

Dataset	Rico			PubLayNet			Magazine		
Model	Max. IoU ↑	Alignment ↓	Const. violation (%) ↓	Max. IoU ↑	Alignment ↓	Const. violation (%) ↓	Max. IoU ↑	Alignment ↓	Const. violation (%) ↓
NDN [16]	**0.36**±0.00	**0.56**±0.03	12.75±0.27	0.31±0.00	0.36±0.00	17.30±0.54	0.23±0.00	1.04±0.05	14.85±0.44
CLG-LO	**0.36**±0.00	0.77±0.09	**0.84**±0.13	**0.36**±0.00	**0.23**±0.01	**4.61**±0.17	**0.26**±0.00	**0.79**±0.03	**1.77**±0.39

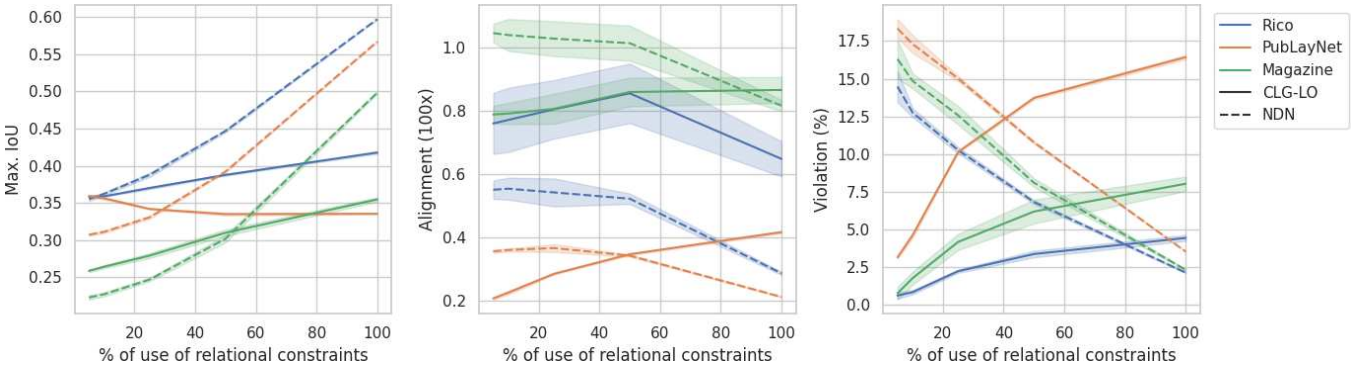

Figure 6: Quantitative results with relational constraints. The different colors correspond to each of the three datasets. The solid lines denotes CLG-LO, and the dashed lines denotes NDN. Higher is better for Max. IoU, and lower is better for Alignment and Violation. Our proposed CLG-LO approach often outperforms NDN when only a small part of relations is specified.

Our optimization-based approach allows us to flexibly change not only the constraint function, but also the objective function. For example, if we wish to limit the amount of change, we can add the distance between the boxes before and after the optimization as a penalty to the objective function. Our approach can also be applied to any model that can generate diverse plausible layouts through manipulating latent variables. Note that when used with VAE-based models [1, 12, 16] that do not have an explicit function to measure the quality of the generated layout, it becomes a constraint satisfaction problem. Our approach still works in such cases, but if the quality of the outcome is problematic, it may be necessary to train an additional measurement network like a discriminator.

There are many open directions for improvement such as incorporating models that approximate human perception as constraints [3, 36] in order to generate more aesthetically pleasing results. Exploring latent codes considering the diversity of layouts is another exciting direction [28], allowing for efficient design exploration with a variety of alternatives. Also, it is worth investigating whether or not our proposed CLG-LO approach can be applied generation problems other than that of layout designs.

ACKNOWLEDGMENTS

This work is partially supported by Waseda University Leading Graduate Program for Embodiment Informatics.

REFERENCES

[1] Diego Martin Arroyo, Janis Postels, and Federico Tombari. 2021. Variational Transformer Networks for Layout Generation. arXiv:arXiv:2104.02416

[2] David Bau, Hendrik Strobelt, William Peebles, Jonas Wulff, Bolei Zhou, Jun-Yan Zhu, and Antonio Torralba. 2019. Semantic Photo Manipulation with a Generative Image Prior. *ACM Trans. Graph.* 38, 4, Article 59 (2019), 11 pages.

[3] Zoya Bylinskii, Nam Wook Kim, Peter O'Donovan, Sami Alsheikh, Spandan Madan, Hanspeter Pfister, Fredo Durand, Bryan Russell, and Aaron Hertzmann. 2017. Learning Visual Importance for Graphic Designs and Data Visualizations. *ACM Symp. User Inter. Soft. Tech.* (2017).

[4] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. In *Adv. Neural Inform. Process. Syst.*

[5] Chia-Hsing Chiu, Yuki Koyama, Yu-Chi Lai, Takeo Igarashi, and Yonghao Yue. 2020. Human-in-the-Loop Differential Subspace Search in High-Dimensional Latent Space. *ACM Trans. Graph.* (2020).

[6] David F. Crouse. 2016. On Implementing 2D Rectangular Assignment Algorithms. *IEEE Trans. Aerospace Electron. Systems* (2016).

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *ACM Symp. User Inter. Soft. Tech.*

[8] Kamal Gupta, Vijay Mahadevan, Alessandro Achille, Justin Lazarow, Larry S. Davis, and Abhinav Shrivastava. 2021. Multimodal Attention for Layout Synthesis in Diverse Domains. https://openreview.net/forum?id=L2LEB4vd9Qw

[9] Nikolaus Hansen. 2016. The CMA Evolution Strategy: A Tutorial. arXiv:arXiv:1604.00772

[10] Paul Henderson, Kartic Subr, and Vittorio Ferrari. 2017. Automatic Generation of Constrained Furniture Layouts. *arXiv preprint arXiv:1711.10939* (2017).

[11] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Adv. Neural Inform. Process. Syst.*

[12] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. 2019. LayoutVAE: Stochastic Scene Layout Generation From a Label Set. In *Int. Conf. Comput. Vis.*

[13] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *Int. Conf. Learn. Represent.*

[14] Tero Karras, Samuli Laine, and Timo Aila. 2019. A Style-Based Generator Architecture for Generative Adversarial Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*

[15] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Int. Conf. Learn. Represent.*

[16] Hsin-Ying Lee, Lu Jiang, Irfan Essa, Phuong B. Le, Haifeng Gong, Ming-Hsuan Yang, and Weilong Yang. 2020. Neural Design Network: Graphic Layout Generation with Constraints. In *Eur. Conf. Comput. Vis.*

[17] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. 2019. LayoutGAN: Synthesizing Graphic Layouts with Vector-Wireframe Adversarial Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* (2019).

[18] Jianan Li, Jimei Yang, Jianming Zhang, Chang Liu, Christina Wang, and Tingfa Xu. 2020. Attribute-conditioned Layout GAN for Automatic Graphic Design. *IEEE Trans. Vis. Comput. Graph.* (2020).

[19] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. 2021. Towards Faster and Stabilized GAN Training for High-fidelity Few-shot Image Synthesis. In *Int. Conf. Learn. Represent.*

[20] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning Design Semantics for Mobile Apps. In *ACM Symp. User Inter. Soft. Tech.*

[21] Sachit Menon, Alex Damian, McCourt Hu, Nikhil Ravi, and Cynthia Rudin. 2020. PULSE: Self-Supervised Photo Upsampling via Latent Space Exploration of Generative Models. In *IEEE Conf. Comput. Vis. Pattern Recog.*

[22] Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization.* Springer, Chapter 17.

[23] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. DesignScape: Design with interactive layout suggestions. In *CHI.*

[24] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2014. Learning layouts for single-pagegraphic designs. *IEEE Trans. Vis. Comput. Graph.* (2014).

[25] Xingang Pan, Xiaohang Zhan, Bo Dai, Dahua Lin, Chen Change Loy, and Ping Luo. 2020. Exploiting Deep Generative Prior for Versatile Image Restoration and Manipulation. In *Eur. Conf. Comput. Vis.*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). 262–277.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Adv. Neural Inform. Process. Syst.*

[27] Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. 2020. READ: Recursive autoencoders for document layout generation. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*

[28] Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. 2016. Quality Diversity: A New Frontier for Evolutionary Computation. *Frontiers in Robotics and AI* (2016).

[29] Daniel Ritchie, Kai Wang, and Yu-an Lin. 2019. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *IEEE Conf. Comput. Vis. Pattern Recog.*

[30] Jacob Schrum, Jake Gutierrez, Vanessa Volz, Jialin Liu, Simon Lucas, and Sebastian Risi. 2020. Interactive Evolution and Exploration within Latent Level-Design Space of Generative Adversarial Networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20).* 148–156.

[31] Nobuyuki Umetani. 2017. Exploring Generative 3D Shapes Using Autoencoder Networks. In *SIGGRAPH Asia 2017 Technical Briefs (SA '17).* Article 24, 4 pages.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Adv. Neural Inform. Process. Syst.*

[33] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* (2020).

[34] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local Neural Networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*

[35] Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. 2020. Deep generative modeling for scene synthesis via hybrid representations. *ACM Trans. Graph.* 39, 2 (2020), 1–21.

[36] Nanxuan Zhao, Ying Cao, and Rynson W.H. Lau. 2018. What Characterizes Personalities of Graphic Designs? *ACM Trans. Graph.* (2018).

[37] Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson W.H. Lau. 2019. Content-aware Generative Modeling of Graphic Design Layouts. *ACM Trans. Graph.* (2019).

[38] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. 2019. PubLayNet: Largest Dataset Ever for Document Layout Analysis. In *IEEE Conf. Doc. Anal. Recog.*

[39] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. 2020. In-Domain GAN Inversion for Real Image Editing. In *Eur. Conf. Comput. Vis.*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). 592–608.

[40] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. 2016. Generative Visual Manipulation on the Natural Image Manifold. In *Eur. Conf. Comput. Vis.*