CanvasVAE: Learning to Generate Vector Graphic Documents

Kota Yamaguchi
CyberAgent
yamaguchi_kota@cyberagent.co.jp

Abstract

Vector graphic documents present visual elements in a resolution free, compact format and are often seen in creative applications. In this work, we attempt to learn a generative model of vector graphic documents. We define vector graphic documents by a multi-modal set of attributes associated to a canvas and a sequence of visual elements such as shapes, images, or texts, and train variational autoencoders to learn the representation of the documents. We collect a new dataset of design templates from an online service that features complete document structure including occluded elements. In experiments, we show that our model, named CanvasVAE, constitutes a strong baseline for generative modeling of vector graphic documents.

1. Introduction

In creative workflows, designers work on visual presentation via vector graphic formats. 2D vector graphics represent images in a compact descriptive structure; instead of spatial array of pixels, graphic documents describe a canvas and arrangement of visual elements such as shapes or texts in a specific format like SVG or PDF. Vector graphics are crucial in creative production for its resolution-free representation, human interpretability, and editability. Because of its importance in creative applications, there has been a long but active history of research on tracing vector graphic representation from a raster image [28, 14, 2, 20, 26].

In this work, we study a generative model of vector graphic documents. While raster-based generative models show tremendous progress in synthesizing high-quality images [10, 24, 12], there has been relatively scarce studies on vector graphic documents [33, 3, 17]. Although both raster and vector graphics deal with images, vector graphics do not have canvas pixels and cannot take advantage of the current mainstream approach of convolutional neural networks without rasterization, which is typically not differentiable [19]. Learning a generative model of vector graphics therefore imposes us unique challenges in 1) how to represent complex data structure of vector graphic formats in a unified manner, 2) how to formulate the learning problem, and 3) how to evaluate the quality of documents.

We address the task of generative learning of vector graphics using a variational auto-encoder (VAE) [13], where we define documents by a multi-modal combination of canvas attributes and a sequence of element attributes. Unlike conditional layout inference [33, 15, 17], we consider unconditional document generation including both a canvas and variable number of elements. Our architecture, named CanvasVAE, learns an encoder that projects a given graphic document into a latent code, and a decoder that reconstructs the given document from the latent code. We adopt Transformer-based network [5] in both the encoder and the decoder to process variable-length sequence of elements in a document. The learned decoder can take randomly sampled latent code to generate a new vector graphic document. For our study, we collect a large-scale dataset of design templates for our study that offers complete document structure and content information. In evaluation, we propose to combine normalized metrics for all attributes to measure the overall quality of reconstruction and generation. We compare several variants of CanvasVAE architecture and show that a Transformer-based model constitutes a strong baseline for the vector graphic generation task.

We summarize our contributions in the following.

1. We propose the CanvasVAE architecture for the task of unconditional generative learning of vector graphic documents, where we model documents by a structured, multi-modal set of canvas and element attributes.
2. We build Crello dataset, which is a dataset consisting of large number of design templates and features complete vector information including occluded elements.
3. We empirically show that our Transformer-based variant of CanvasVAE achieves a strong performance in both document reconstruction and generation.

2. Related work

Generative layout modeling There has been several attempts at conditional layout modeling where the goal is to generate bounding box arrangements given certain inputs. LayoutVAE [11] learns a two-stage autoregressive VAE that takes a label set and generates bounding boxes for each label, for scene image representation. For design applications, Zheng *et al*. [33] report a generative model for magazine layout conditioned on a set of elements and meta-data, where raster adversarial networks generate layout maps. Lee *et al*. [15] propose a three-step approach to predict a layout given an initial set of elements that accepts partial relation annotation. Li *et al*. [16, 17] learn a model that refines the geometry of the given elements, such that the refined layout looks realistic to a discriminator built on a differentiable wire-frame rasterizer. Tan *et al*. [30] propose text-to-scene generation that explicitly considers a layout and attributes. Wang *et al*. [31] consider a reinforcement learning approach to select appropriate elements for the given document. For UI layout domain, Manandhar *et al*. [22] propose to learn UI layout representation by metric learning and raster decoder. Li *et al*. [18] recently report an attempt in multi-modal representation learning of UI layout.

In contrast to conditional layout generation, we tackle on the task of *unconditional* document generation including layout and other attributes. Gupta *et al*. recently report autoregressive model for generating layout [7] but without learning a latent representation and instead relies on beam search. READ [25] is the only pilot study similar to our unconditional scenario, although their recursive model only considers labeled bounding box without content attributes. Arroyo *et al*. [1] very recently report a layout generation model. Our model fully works in symbolic vector data without explicit rasterization [33, 16], which allows us to easily process data in a resolution free manner.

Vector graphic generation Although our main focus is document-level generation, there has been several important work in stroke or path level vector graphic modeling that aims at learning to generate resolution-free shapes. Sketch RNN is a pioneering work on learning drawing strokes using recurrent networks [8]. SPIRAL is a reinforcment adversarial learning approach to vectorize a given raster image [6]. Lopes *et al*. learn an autoregressive VAE to generate vector font strokes [21]. Song *et al*. report a generative model of Bézier curves for sketch strokes [29]. Carlier *et al*. propose DeepSVG architecture that consists of a hierarchical auto-encoder that learns a representation for a set of paths [3]. We get many inspirations from DeepSVG especially in our design of oneshot decoding architecture.

3. Vector graphic representation

3.1. Document structure

In this work, we define vector graphic documents to be a single-page canvas and associated sequence of visual elements such as texts, shapes, or raster images. We represent a document $X = (X_c, X_E)$ by a set of canvas attributes $X_c = \{\mathbf{x}_k | k \in \mathcal{C}\}$ and a sequence of elements $X_E = \{X_e^1, X_e^2, \cdots, X_e^T\}$, where $X_e^t = \{\mathbf{x}_k^t | k \in \mathcal{E}\}$ is a set of element attributes. We denote a set of canvas and element attribute indices by $\mathcal{C}$ and $\mathcal{E}$, respectively. Canvas attributes represent global document properties, such as canvas size or document category. Element attributes indicate element-specific configuration, such as position, size, type of the element, opacity, color, or a texture image if the element represents a raster image. In addition, we explicitly include the element length in the canvas attributes X_c. We represent elements by a sequence, where the order reflects the depth of which elements appear on top. The actual attribute definition depends on datasets we describe in the next section.

3.2. Datasets

Crello dataset Crello dataset consists of design templates we obtained from online design service, *crello.com*. The dataset contains designs for various display formats, such as social media posts, banner ads, blog headers, or printed posters, all in a vector format. In dataset construction, we first downloaded design templates and associated resources (e.g., linked images) from *crello.com*. After the initial data acquisition, we inspected the data structure and identified useful vector graphic information in each template. Next, we eliminated mal-formed templates or those having more than 50 elements, and finally obtained 23,182 templates. We randomly partition the dataset to 18,714 / 2,316 / 2,331 examples for train, validation, and test splits.

In Crello dataset, each document has canvas attributes $X_c = \{\mathbf{x}_{\text{length}}, \mathbf{x}_{\text{width}}, \mathbf{x}_{\text{height}}, \mathbf{x}_{\text{group}}, \mathbf{x}_{\text{category}}, \mathbf{x}_{\text{format}}\}$ and element attributes $X_e^t = \{\mathbf{x}_{\text{type}}^t, \mathbf{x}_{\text{position}}^t, \mathbf{x}_{\text{size}}^t, \mathbf{x}_{\text{color}}^t, \mathbf{x}_{\text{opacity}}^t, \mathbf{x}_{\text{image}}^t\}$. Table 1 summarizes the detail of each attribute. Image and color attributes are exclusive; we extract color for text placeholders and solid backgrounds, and we extract image features for shapes and image elements in the document. Except for image features, we quantize numeric attributes to one-hot representations, such as element position, size, or colors, because 1) discretization implicitly enforces element alignment, and 2) attributes often do not follow normal distribution suitable for regression. The image feature allows us content-aware document modeling, and also is useful for visualization purpose.

We obtain image features using a raster-based convolutional VAE that we pre-train from all the image and shape elements in the Crello dataset. We do not use ImageNet pre-trained model here, because ImageNet does not contain alpha channels nor vector shapes. For pre-training of the VAE, we rasterize all the image and shape elements in 256×256 pixel canvas with resizing, and saves in RGBA raster format. From the rasterized images, we learn a VAE

Table 1: Attribute descriptions for vector graphic data

Dataset	Attribute of	Name	Type	Size	Dim	Description
Crello	Canvas	Length	Categorical	50	1	Length of elements up to 50
		Group	Categorical	7	1	Broad design group, such as social media posts or blog headers
		Format	Categorical	68	1	Detailed design format, such as Instagram post or postcard
		Width	Categorical	42	1	Canvas pixel width available in crello.com
		Height	Categorical	47	1	Canvas pixel height available in crello.com
		Category	Categorical	24	1	Topic category of the design, such as holiday celebration
	Element	Type	Categorical	6	1	Element type, such as vector shape, image, or text placeholder
		Position	Categorical	64	2	Left and top position each quantized to 64 bins
		Size	Categorical	64	2	Width and height each quantized to 64 bins
		Opacity	Categorical	8	1	Opacity quantized to 8 bins
		Color	Categorical	16	3	RGB color each quantized to 16 bins, only relevant for solid fill and texts
		Image	Numerical	1	256	Pre-trained image feature, only relevant for shapes and images
RICO	Canvas	Length	Categorical	50	1	Length of elements up to 50
	Element	Component	Categorical	27	1	Element type, such as text, image, icon, etc.
		Position	Categorical	64	2	Left and top position each quantized to 64 bins
		Size	Categorical	64	2	Width and height each quantized to 64 bins
		Icon	Categorical	59	1	Icon type, such as *arrow*, *close*, *home*
		Button	Categorical	25	1	Text on button, such as *login* or *back*
		Clickable	Categorical	2	1	Binary flag indicating if the element is clickable

consisting of MobileNetV2-based encoder [27] and a 6-layer convolutional decoder. After pre-training the convolutional VAE, we obtain a 256-dimensional latent representation using the learned image encoder for all the image and shape elements in Crello dataset.

In contrast to existing layout datasets that mainly consider a set of labeled bounding boxes [4, 33, 34], our Crello dataset offers complete vector graphic structure including appearance for occluded elements. This enables us to learn a generative model that considers the appearance and attributes of graphic elements in addition to the layout structure. Also, Crello dataset contains canvas in various aspect ratio. This imposes us a unique challenge, because we have to handle variable-sized documents that raster-based models do not work well with.

RICO dataset RICO dataset offers a large number of user interface designs for mobile applications with manually annotated elements [4]. We use RICO dataset to evaluate the generalization ability of our CanvasVAE. All the UI screenshots from RICO have a fixed resolution of 2560×1440 pixels, and there is no document-wise label. We set canvas attributes to only have element length: $X_c = \{\mathbf{x}_{\text{length}}\}$, and for each element, we model $X_e^t = \{\mathbf{x}^t_{\text{component}}, \mathbf{x}^t_{\text{position}}, \mathbf{x}^t_{\text{size}}, \mathbf{x}^t_{\text{icon}}, \mathbf{x}^t_{\text{button}}, \mathbf{x}^t_{\text{clickable}}\}$. Most of the pre-processing follows Crello dataset; we quantize numeric attributes to one-hot representations. Table 1 summarizes the attributes we consider in this work.

4. CanvasVAE

Our goal is to learn a generative model of vector graphic documents. We aim at learning a VAE that consists of a probabilistic encoder and a decoder using neural networks.

VAE basics Let us denote a vector graphic instance by X and a latent code by $\mathbf{z}$. A VAE learns a generative model $p_\theta(X, \mathbf{z}) = p_\theta(X|\mathbf{z})p_\theta(\mathbf{z})$ and an approximate posterior $q_\phi(\mathbf{z}|X)$, using variational lower bounds [13]:

$$\begin{aligned}\mathcal{L}(X;\theta,\phi) =& \mathbb{E}_{q_\phi(\mathbf{z}|X)}\left[\log p_\theta(X|\mathbf{z})\right] \\ &- \mathrm{KL}(q_\phi(\mathbf{z}|X)||p_\theta(\mathbf{z})),\end{aligned} \tag{1}$$

where ϕ is the parameters of the inference model and θ is the parameters of the generative model.

We model the approximate variational posterior $q_\phi(\mathbf{z}|X)$ by a Gaussian distribution with a diagonal covariance:

$$q_\phi(\mathbf{z}|X) \equiv \mathcal{N}(\mathbf{z}; \mu_\phi(X), \sigma_\phi(X)^2\mathbf{I}), \tag{2}$$

where $\mu_\phi(X)$ and $\sigma_\phi(X)$ are the encoder outputs that we model by a neural network. We set the prior over the latent code $\mathbf{z}$ to be a unit multivariate Gaussian $p_\theta(\mathbf{z}) \equiv \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. We also model the data likelihood $p_\theta(X|\mathbf{z})$ using a neural network. Fig 1 illustrates our CanvasVAE encoder and decoder architecture.

Encoder Our encoder takes a vector graphic input X and predicts the parameters of approximate prior $\mu_\phi(X)$, $\sigma_\phi(X)$. We describe the encoder in the following:

$$\mathbf{h}_c = \textstyle\sum_{k \in \mathcal{C}} f_k(\mathbf{x}_k; \phi), \tag{3}$$

$$\mathbf{h}_e^t = \textstyle\sum_{k \in \mathcal{E}} f_k(\mathbf{x}_k^t; \phi) + \mathbf{x}^t_{\text{position},\phi}, \tag{4}$$

$$\mathbf{h}_{\text{enc}} = \frac{1}{T} \textstyle\sum_t^T B(\{\mathbf{h}_e^t\}, \mathbf{h}_c; \phi), \tag{5}$$

$$\mu_\phi(X) = f_\mu(\mathbf{h}_{\text{enc}}; \phi), \tag{6}$$

$$\sigma_\phi(X) = f_\sigma(\mathbf{h}_{\text{enc}}; \phi). \tag{7}$$

The encoder first projects each canvas attribute $\mathbf{x}_k$ using a feed-forward layer f_k to the dimensionality, and adds up to

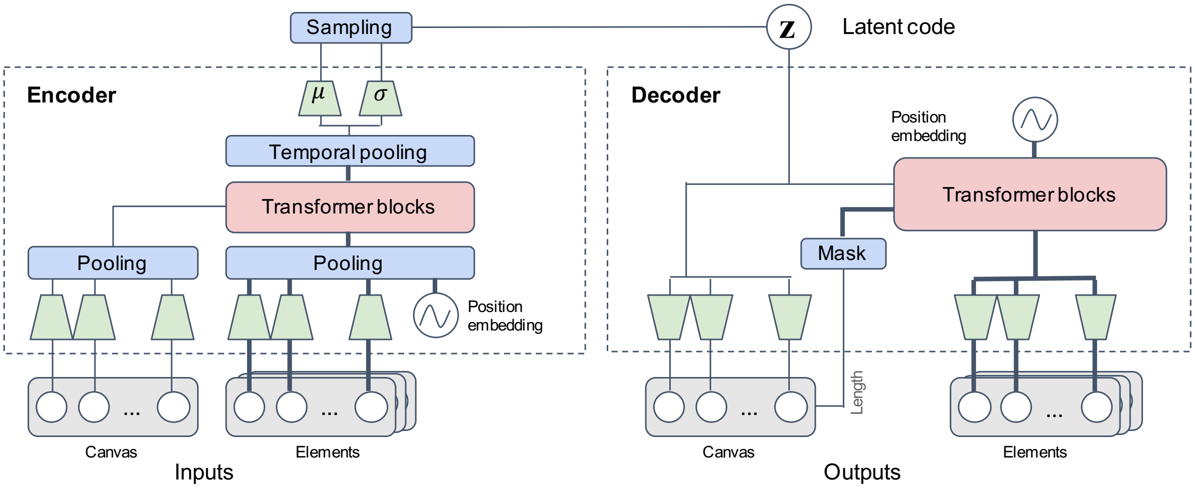

Figure 1: CanvasVAE architecture.

make the hidden side input $\mathbf{h}_c$ to the Transformer block. Similarly, the encoder projects each element attribute $\mathbf{x}_k^t$ using a feed-forward layer to the same dimensionality, and adds up together with the position embedding $\mathbf{x}_{\text{position},\phi}^t$ to make the hidden input $\mathbf{h}_e^t$ to the Transformer block for each step t. The positional embedding $\mathbf{x}_{\text{position},\phi}^t$ provides information on absolute position within the element sequence [5]. We learn the positional embedding during training. B is a variant of Transformer model that adds a side input between the self attention and the feed-forward layer, which is similar to the decoder block of DeepSVG [3]. We stack up multiple Transformer blocks to transform input embedding $\mathbf{h}_c$ and $\mathbf{h}_e^t$ to produce temporally pooled internal representation $\mathbf{h}_{\text{enc}}$. f_μ and f_σ are the last feed-forward layer of the encoder to produce $\mu_\phi(X)$ and $\sigma_\phi(X)$.

Decoder Our decoder takes a sampled latent code $\mathbf{z}$ and produces reconstruction $\hat{X}$ from $p_\theta(X|\mathbf{z}) = \prod_k p_\theta(\mathbf{x}_k|\mathbf{z}) \prod_t p_\theta(\mathbf{x}_k^t|\mathbf{z})$. We describe our decoder by:

$$\mathbf{h}_{\text{dec}}^t = B(\{\mathbf{x}_{\text{position},\theta}^t\}, \mathbf{z}; \theta), \tag{8}$$

$$p_\theta(\mathbf{x}_k|\mathbf{z}) = f_k(\mathbf{x}_k, \mathbf{z}; \theta), \tag{9}$$

$$p_\theta(\mathbf{x}_k^t|\mathbf{z}) = f_k(\mathbf{x}_k^t, \mathbf{h}_{\text{dec}}^t; \theta), \tag{10}$$

where f_k is the last feed-forward network for the attribute k. Our decoder uses the same Transformer block B with the encoder. The decoder has the positional embedding $\mathbf{x}_{\text{position},\theta}^t$ to feed the absolute position information for sequence reconstruction.

At generation, we apply stochastic sampling to $\mathbf{z}$ and obtain a maximum likelihood estimate from our decoder head $p_\theta(\mathbf{x}_k|\mathbf{z})$ for categorical attributes, or regression output for numerical attributes. To generate a sequence from the latent code $\mathbf{z}$, we have to first decide the number of elements in the document. We predict the length T from $p_\theta(\mathbf{x}_{\text{length}}|\mathbf{z})$, and feed the masking information to the Transformer block to exclude out-of-sequence elements in self-attention, and drop extra elements at the final reconstruction.

Loss function We derive the loss function for our CanvasVAE from the variational lower bounds (Eq 1). For a sample X in our dataset, the loss for each document is given by:

$$\begin{aligned}\mathcal{L}(X, \hat{X}; \theta, \phi) = &\sum_{k \in \mathcal{C}} \mathcal{L}_k(\mathbf{x}_k, \hat{\mathbf{x}}_k) + \sum_{k \in \mathcal{E}} \sum_t^T \mathcal{L}_k(\mathbf{x}_k^t, \hat{\mathbf{x}}_k^t) \\ &+ \lambda_{\text{KL}} \text{KL}(\mathcal{N}(\mathbf{z}; \mu_\phi, \sigma_\phi) || \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})) \\ &+ \lambda_{\text{L2}}(|\phi|^2 + |\theta|^2),\end{aligned} \tag{11}$$

where λ_{KL} and λ_{L2} are hyper-parameters to weight the regularization terms. $\mathcal{L}_k$ is a loss term for attribute k. We use cross entropy for categorical attributes and mean squared error for numeric attributes. At training time, we use teacher-forcing; we discard the predicted length $\hat{T}$ and force the ground truth length T in the decoder.

5. Experiments

We evaluate our CanvasVAE in reconstruction and generation scenarios. In reconstruction, we evaluate the overall capability of our encoder-decoder model to reproduce the given input. In generation scenario, we evaluate the decoder capability in terms of the quality of randomly generated documents.

5.1. Evaluation metrics

5.1.1 Reconstruction metrics

We have to be able to measure the similarity between two documents to evaluate reconstruction quality. Unlike raster images, there is no standard metric to measure the distance between vector graphic documents. Our loss function (Eq 11) is also not appropriate due to teacher-forcing of sequence length. Considering the multi-modal nature of vector graphic formats, we argue that an ideal metric should be able to evaluate the quality of all the modalities in the document at a uniform scale, and that the metric can handle variable length structure. We choose the following two metrics to evaluate the document similarity.

Structural similarity For document X_1 and X_2, we measure the structural similarity by the mean of normalized scores for each attribute k:

$$S(X_1, X_2) = \frac{1}{|\mathcal{C}'| + |\mathcal{E}|} \left[\sum_{k \in \mathcal{C}'} s_k(\mathbf{x}_{k,1}, \mathbf{x}_{k,2}) + \sum_{k \in \mathcal{E}} s_k(\{\mathbf{x}^t_{k,1}\}, \{\mathbf{x}^t_{k,2}\}) \right], \tag{12}$$

where $s_k \in [0, 1]$ is a scoring function, and $\mathcal{C}' = \mathcal{C} \backslash \{\mathrm{length}\}$. We exclude length from the canvas attributes because element scores take length into account. For canvas attributes, we adopt *accuracy* as the scoring function since there are only categorical attributes in our datasets.

For categorical element attributes, a scoring function must be able to evaluate variable length elements. We use BLEU score [23] that is a precision-based metric often used in machine translation. BLEU penalizes a shorter prediction by the brevity term: $\exp\left(\min\left(0, 1 - \frac{|T|}{|\hat{T}|}\right)\right)$, where $\hat{T}$ is the predicted element length. We use unigram BLEU for evaluation, because vector graphic elements do not exhibit strong ordinal constraints and elements can be swapped as long as they do not overlap. For the image feature that is the only numerical element attribute in Crello, we use the cosine similarity in [0, 1] scale between the average-pooled features over sequence, multiplied by the brevity term of BLEU score. Note that our structural similarity is not symmetric because BLEU is not symmetric.

In Crello, the presence of *image* and *color* attributes depend on the element *type*, and $\{\mathbf{x}^t_k\}$ can become empty. We exclude empty attributes from $\mathcal{E}$ in the calculation of eq 12 if either X_1 or X_2 include empty attributes.

We evaluate reconstruction performance by the average score over the document set $\mathcal{X} = \{X_i\}$:

$$S_{\mathrm{reconst}}(\mathcal{X}) = \frac{1}{|\mathcal{R}|} \sum_i S(X_i, \hat{X}_i). \tag{13}$$

Layout mean IoU We also include evaluation by mean intersection over union (mIoU) on labeled bounding boxes [22] to analyze layout quality. We use *type* attribute in Crello dataset and *component* attribute in RICO dataset as a primary label for elements. To compute mIoU, we draw bounding boxes on a canvas in the given element order, compute the IoU for each label, then average over labels. Since we quantize position and size of each element, we draw bounding boxes on a 64×64 grid canvas. Similar to Eq 13, we obtain the final score by dataset average.

The mIoU metric ignores attributes other than element position, element size, and element label. Content attributes such as image or color have no effect on the mIoU metric. Although Crello dataset has variable-sized canvas, we ignore the aspect ratio of the canvas and only evaluate on the relative position and size of elements.

5.1.2 Generation metric

Similar to reconstruction, there is no standard approach to evaluate the similarity between sets of vector graphic documents. It is also not appropriate to use a raster metric like FID score [9] because our document can not be rasterized to a fixed resolution nor is a natural image. We instead define the following metric to evaluate the distributional similarity of vector graphic documents.

For real and generated document sets $\mathcal{X}_1$ and $\mathcal{X}_2$, we first obtain descriptive statistics for each attribute k, then compute the similarity between the two sets:

$$S_{\mathrm{gen}}(\mathcal{X}_1, \mathcal{X}_2) = \frac{1}{|\mathcal{C}| + |\mathcal{E}|} \sum_{k \in \mathcal{C} \cup \mathcal{E}} d_k(a_k(\mathcal{X}_1), a_k(\mathcal{X}_2)), \tag{14}$$

where a_k is an aggregation function that computes descriptive statistics of attribute k, and d_k is a scoring function. For categorical attributes, we use histogram for a_k and normalized histogram intersection for d_k. For numerical attributes, we use average pooling for a_k and cosine similarity for d_k.

5.2. Baselines

We include the following variants of our CanvasVAE as baselines. Since there is no reported work that is directly comparable to CanvasVAE, we carefully pick comparable building blocks from existing work.

AutoReg LSTM We replace Transformer blocks and temporal pooling in our model (Eq 5, 8) with an LSTM. The side input to the transformer blocks is treated as initial hidden states. Also, we introduce autoregressive inference procedure instead of our one-shot decoding using positional embedding; we predict an element at t given the elements until $t - 1$ in the decoder. The initial input is a special beginning-of-sequence embedding, which we learn during training. Our autoregressive LSTM baseline has a decoder

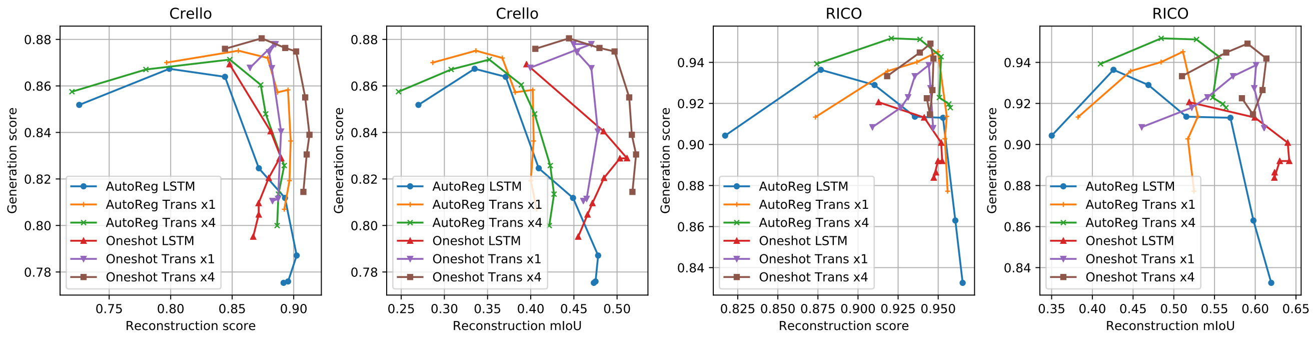

Figure 2: Performance curves in terms of S_{reconst} vs. S_{gen} and mIoU vs. S_{gen} over λ_{KL} in validation splits. Top-right models show better performance in both reconstruction and generation.

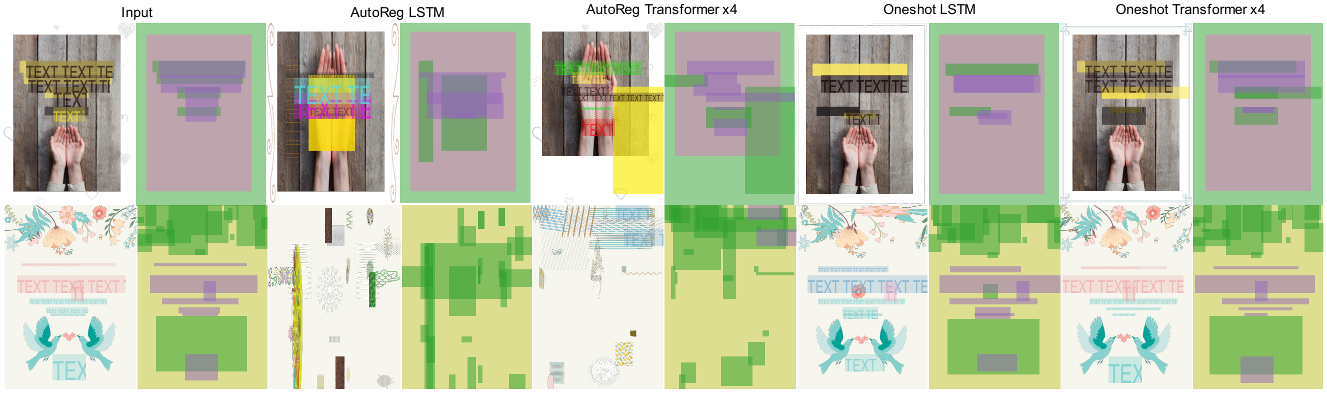

Figure 3: Crello reconstruction comparison. For each item, a left picture shows visualization of the design template with colored text placeholders and textured elements, and a right color map illustrates element types. The legend of types are the following: green = *vector shape*, magenta = *image*, purple = *text placeholder*, and yellow = *solid fill*.

similar to LayoutVAE [11], although we do not stochastically sample at each step nor have conditional inputs but have an encoder for the latent code.

AutoReg Transformer Similar to Autoregressive LSTM, we introduce the autoregressive inference but use Transformer blocks. The decoding process is similar to LayoutTransformer [7], but we add an encoder for the latent code. We also explicitly predict sequence length instead of an end-of-sequence flag to terminate the prediction.

Oneshot LSTM We use positional embedding but replace Transformer blocks with an LSTM. We use a bidrectional LSTM for this one-shot model because positional embedding allows both past and future information for prediction.

Oneshot Transformer Our CanvasVAE model described in Sec 4. We also compare the number of Transformer blocks at 1 and 4 for ablation study.

5.3. Quantitative evaluation

For each baseline, we report the test performance of the best validation model in terms of S_{gen} that we find by a grid search over λ_{KL}. For other hyper-parameters, we empirically set the size of latent code $\mathbf{z}$ to 512 for Crello and 256 for RICO, and $\lambda_{\text{L2}} = 1e-6$ in all baselines. We train all baseline models using Adam optimizer with learning rate fixed to $1e-3$ for 500 epochs in both datasets. For generation evaluation, we sample $\mathbf{z}$ from zero-mean unit normal

Table 2: Test performance (%).

Dataset	Model	S_{reconst}	mIoU	S_{gen}
Crello	AutoReg LSTM	79.75	33.52	86.44
	AutoReg Trans x1	85.47	33.86	87.48
	AutoReg Trans x4	84.65	35.36	86.60
	Oneshot LSTM	84.95	40.50	86.85
	Oneshot Trans x1	**88.67**	**47.02**	87.57
	Oneshot Trans x4	87.75	45.50	**88.15**
RICO	AutoReg LSTM	87.73	42.51	93.74
	AutoReg Trans x1	**94.96**	51.13	94.40
	AutoReg Trans x4	92.06	48.74	95.11
	Oneshot LSTM	91.01	51.93	92.05
	Oneshot Trans x1	94.35	**60.42**	93.90
	Oneshot Trans x4	94.45	59.47	**95.14**

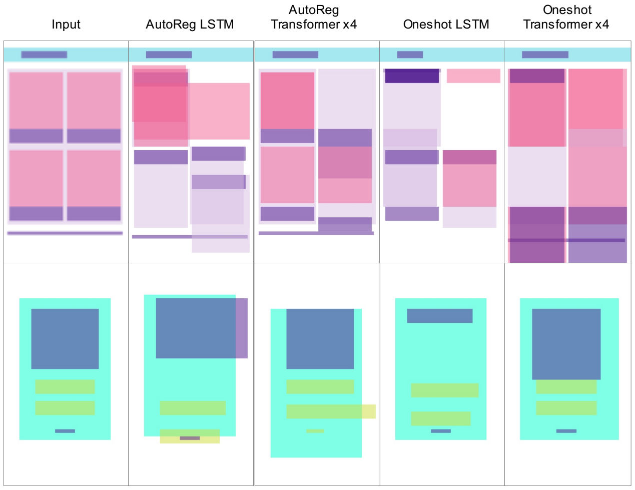

Figure 4: RICO reconstruction comparison. Color indicates the component label of each element. Color is from the original RICO schema [4].

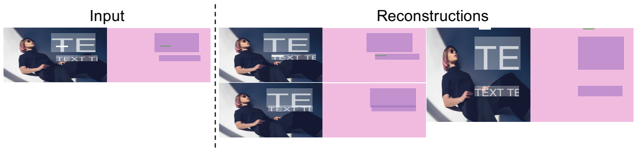

Figure 5: Stochastic sampling and reconstruction examples.

distribution up to the same size to the test split.

Table 2 summarizes the test evaluation metrics of the baseline models. In Crello, oneshot Transformer x1 configuration has the best reconstruction in structure (88.67%) and layout mIoU (47.02%), while oneshot Transformer x4 has the best generation score (88.15%). In RICO, autoregressive Transformer x1 has the best structure reconstruction (94.96%), oneshot Transformer x1 has the best mIoU (60.42%), and oneshot Transformer x4 has the best generation score (95.14%).

Performance trade-offs We note that the choice of λ_{KL} has a strong impact on the evaluation metric, and that explains the varying testing results in Table 2. We plot in Fig 2 the validation performance as we move λ_{KL} from 2^1 to 2^8 in Crello, and from 2^1 to 2^7 in RICO. The plots clearly show there is a trade-off relationship between reconstruction and generation. A smaller KL divergence indicates smoother latent space, which in turn indicates better generation quality from random sampling of $\mathbf{z}$ but hurts the reconstruction performance. From the plots, oneshot Transformer x4 seems consistently performing well in both datasets except for S_{reconst} evaluation in RICO, where most baselines saturate the reconstruction score. We suspect S_{reconst} saturation is due to over-fitting tendency in RICO dataset, as RICO does not contain high dimensional attributes like image feature. Given the performance characteristics, we conjecture that oneshot Transformer performs the best.

Autoregressive vs. oneshot Table 2 and Fig 2 suggests oneshot models consistently perform better than the autoregressive counterparts in both datasets. This trend makes sense, because autoregressive models cannot consider the layout placement of future elements at the current step. In contrast, oneshot models can consider spatial relationship between elements better at inference time.

5.4. Qualitative evaluation

Reconstruction We compare reconstruction quality of Crello and RICO testing examples in Fig 3 and Fig 4. Here, we reconstruct the input deterministically by the mean latent code in Eq 2. Since Crello dataset has rich content attributes, we present a document visualization that fills in image and shape elements using a nearest neighbor retrieval by image features (Sec 3.2), and a color map of element type bounding boxes. For RICO, we show a color map of *component* bounding boxes.

We observe that, while all baselines reasonably reconstruct documents when there are a relatively small number of elements, oneshot models tend to reconstruct better as a document becomes more complex (Second row of Fig 3).

We also show how sampling $\mathbf{z}$ from $q_\phi(\mathbf{z}|X)$ results in variation in the reconstruction quality in Fig 5. Depending on the input, sampling sometimes leads to different layout arrangement or canvas size.

Interpolation One characteristic of VAEs is the smoothness of the latent space. We show in Fig 6 an example of interpolating latent codes between two documents in Crello dataset. The result shows a gradual transition between two documents that differ in many aspects, such as the number of elements or canvas size. The resulting visualization is discontinuous in that categorical attributes or retrieved images change at specific point in between, but we can still observe some sense of continuity in design.

Generation Fig 7 shows randomly generated Crello design documents from oneshot Transformer x4 configuration. Our CanvasVAE generates documents in diverse layouts and aspect ratios. Generated documents are not realistic in that the quality is not sufficient in immediate use in real-world creative applications, but seem to already serve for inspirational purposes. Also, we emphasize that these generated documents are easily editable thanks to the vector graphic format.

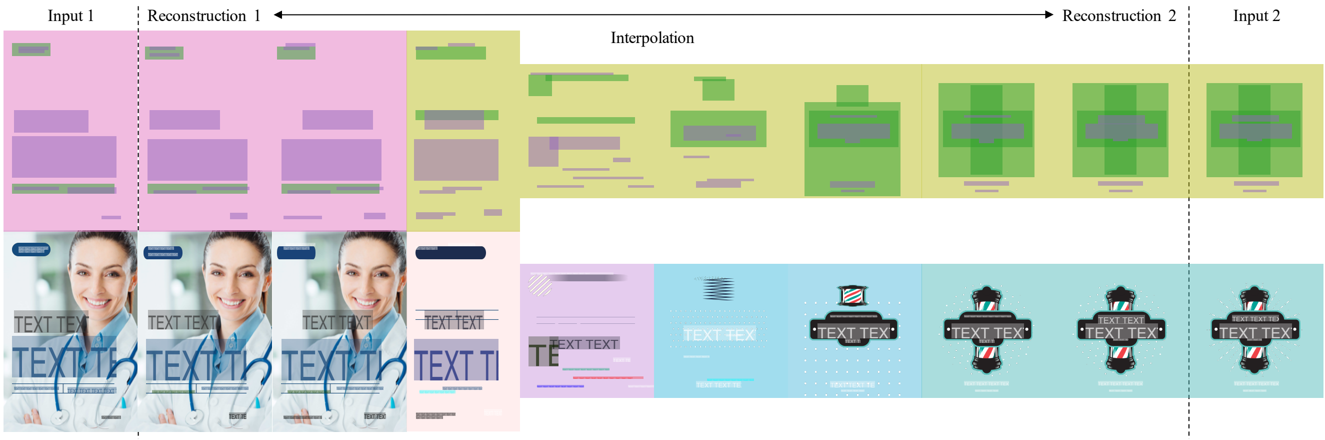

Figure 6: Interpolation example.

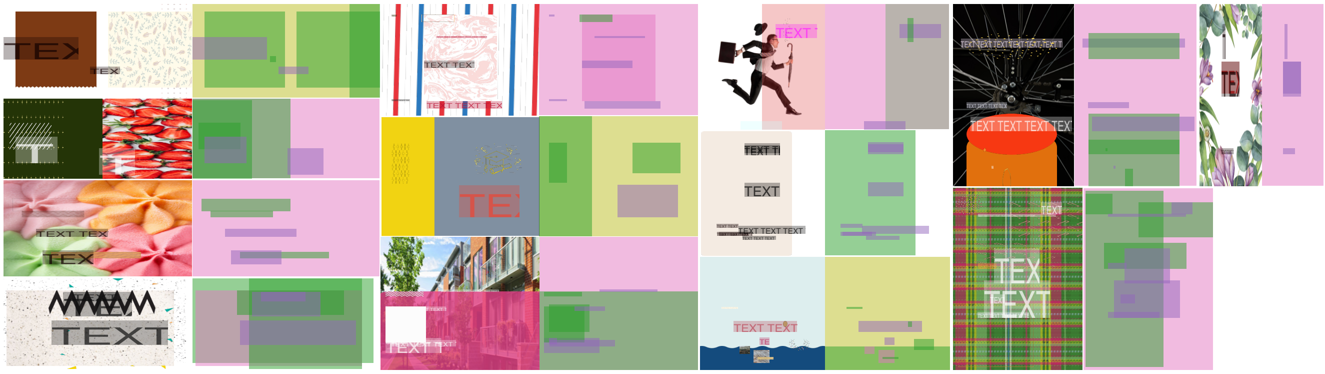

Figure 7: Randomly generated Crello documents.

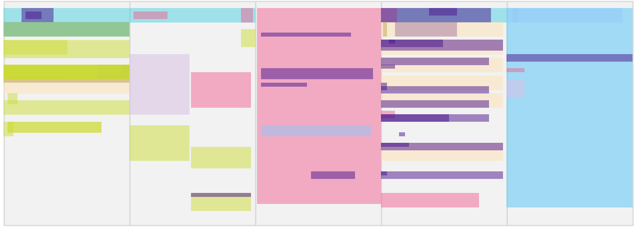

Figure 8: Randomly generated RICO documents.

We also show in Fig 8 randomly generated UIs with RICO dataset. Although sometimes generated layouts include overlapping elements, they show diverse layout arrangements with semantically meaningful structure such as toolbar or list components.

6. Conclusion

We present CanvasVAE, an unconditional generative model of vector graphic documents. Our model learns an encoder and a decoder that takes vector graphic consisting of canvas and element attributes including layout geometry. With our newly built Crello dataset and RICO dataset, we demonstrate CanvasVAE successfully learns to reconstruct and generate vector graphic documents. Our results constitute a strong baseline for generative modeling of vector graphic documents.

In the future, we are interested in further extending CanvasVAE by generating text content and font styling, integrating pre-training of image features in an end-to-end model, and combining a learning objective that is aware of appearance, for example, by introducing differentiable rasterizer [19] to compute raster reconstruction loss. We also wish to look at whether feedback-style inference [32] allows partial inputs such as user-specified constraints [15], which is commonly seen in application scenarios.

References

[1] Diego Martin Arroyo, Janis Postels, and Federico Tombari. Variational transformer networks for layout generation. In *CVPR*, pages 13642–13652, 2021.

[2] Mikhail Bessmeltsev and Justin Solomon. Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics (TOG)*, 38(1):1–12, 2019.

[3] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. DeepSVG: A hierarchical generative network for vector graphics animation. *arXiv preprint arXiv:2007.11301*, 2020.

[4] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. RICO: A mobile app dataset for building data-driven design applications. In *ACM Symposium on User Interface Software and Technology*, pages 845–854, 2017.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *ICML*, pages 1666–1675. PMLR, 2018.

[7] Kamal Gupta, Alessandro Achille, Justin Lazarow, Larry Davis, Vijay Mahadevan, and Abhinav Shrivastava. Layout generation and completion with self-attention. *arXiv preprint arXiv:2006.14615*, 2020.

[8] David Ha and Douglas Eck. A neural representation of sketch drawings. In *ICLR*, 2018. 2018.

[9] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[10] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *CVPR*, pages 1219–1228, 2018.

[11] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. LayoutVAE: Stochastic scene layout generation from a label set. In *CVPR*, pages 9895–9904, 2019.

[12] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *CVPR*, 2020.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] Johannes Kopf and Dani Lischinski. Depixelizing pixel art. In *ACM SIGGRAPH 2011*, pages 1–8. 2011.

[15] Hsin-Ying Lee, Weilong Yang, Lu Jiang, Madison Le, Irfan Essa, Haifeng Gong, and Ming-Hsuan Yang. Neural design network: Graphic layout generation with constraints. *ECCV*, 2020.

[16] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. LayoutGAN: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*, 2019.

[17] Jianan Li, Jimei Yang, Jianming Zhang, Chang Liu, Christina Wang, and Tingfa Xu. Attribute-conditioned layout gan for automatic graphic design. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[18] Toby Jia-Jun Li, Lindsay Popowski, Tom M Mitchell, and Brad A Myers. Screen2vec: Semantic embedding of gui screens and gui components. *arXiv preprint arXiv:2101.11103*, 2021.

[19] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[20] Yunchao Liu and Zheng Wu. Learning to describe scenes with programs. In *ICLR*, 2019.

[21] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *CVPR*, pages 7930–7939, 2019.

[22] Dipu Manandhar, Dan Ruta, and John Collomosse. Learning structural similarity of user interface layouts using graph networks. In *ECCV*, pages 730–746. Springer, 2020.

[23] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[24] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

[25] Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. READ: Recursive autoencoders for document layout generation. In *CVPR Workshops*, pages 544–545, 2020.

[26] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. *arXiv preprint arXiv:2102.02798*, 2021.

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[28] Peter Selinger. Potrace: a polygon-based tracing algorithm. *http://potrace.sourceforge.net/potrace.pdf*, 2003.

[29] Yi-Zhe Song. Béziersketch: A generative model for scalable vector sketches. *ECCV*, 2020.

[30] Fuwen Tan, Song Feng, and Vicente Ordonez. Text2scene: Generating compositional scenes from textual descriptions. In *CVPR*, pages 6710–6719, 2019.

[31] Guolong Wang, Zheng Qin, Junchi Yan, and Liu Jiang. Learning to select elements for graphic design. In *ICMR*, pages 91–99, 2020.

[32] Tianlu Wang, Kota Yamaguchi, and Vicente Ordonez. Feedback-prop: Convolutional neural network inference under partial evidence. In *CVPR*, pages 898–907, 2018.

[33] Xinru Zheng, Xiaotian Qiao, Ying Cao, and Rynson WH Lau. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019.

[34] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. Publaynet: largest dataset ever for document layout analysis. In *ICDAR*, pages 1015–1022. IEEE, 2019.