Layout Generation and Baseline Implementation

Chennamolla Vinay Swapnil Saurav Tanishk Gupta Tushar Jain
Yash Jain Yash Mittal

A layout is a set of graphical elements that are placed together in an aesthetic and meaningful way to convey some information. Layouts are important graphical communication tools for graphical designs and user interfaces. Producing layouts has been a topic of interest and research for quite some time. However, this has been a challenging task as the present real-world layouts are highly variable and generating a layout with proper semantic relationship between elements was a challenging task.

1. Layout VAE

1.1. Introduction

LayoutVAE is a variational autoencoder based model . It is a probabilistic and autoregressive model which generates the scene layout using latent variables in lower dimensions . It is capable of generating different layouts using the same data point.

1.2. Methodology

We have implemented LayoutVAE in PyTorch 1.9.0. The model comprises two parts CountVAE and BBoxVAE the model is trained in such a way that both these parts can be used separately for predictions. In the original paper the model was not used for predicting for structured layouts datasets. Also, there was only very little information about the loss functions.

1.3. Model

The model consists of the following layers:

Layer	Description	Hyperparameters
Conditional Embedding	It takes conditioning information like current label, label set and previous predictions and creates conditional embedding	Embedding Dimension (By default, 128)
Posterior Encoder	Encodes the ground truth data and conditional embedding together into latent space.	Latent Dimension
Prior Encoder	Takes conditional embedding and encodes it into latent space.	Latent Dimension
Decoder	It takes latent variables as input and output the distribution of bounding boxes.	Latent Dimension
Reparameterization	Reparameterization trick for gaussian distribution.[7]	None

Poisson Sampling	Samples from Poisson distribution	None

The model architecture for the layout VAE is shown in the figure below.

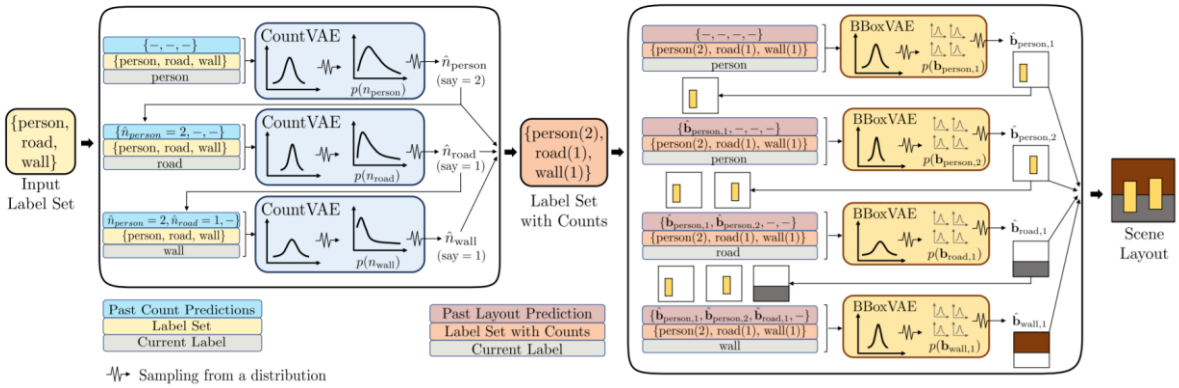


Fig 1.1 :Layout VAE Model (As per [2])

1.3.1. Model Blocks and Layers

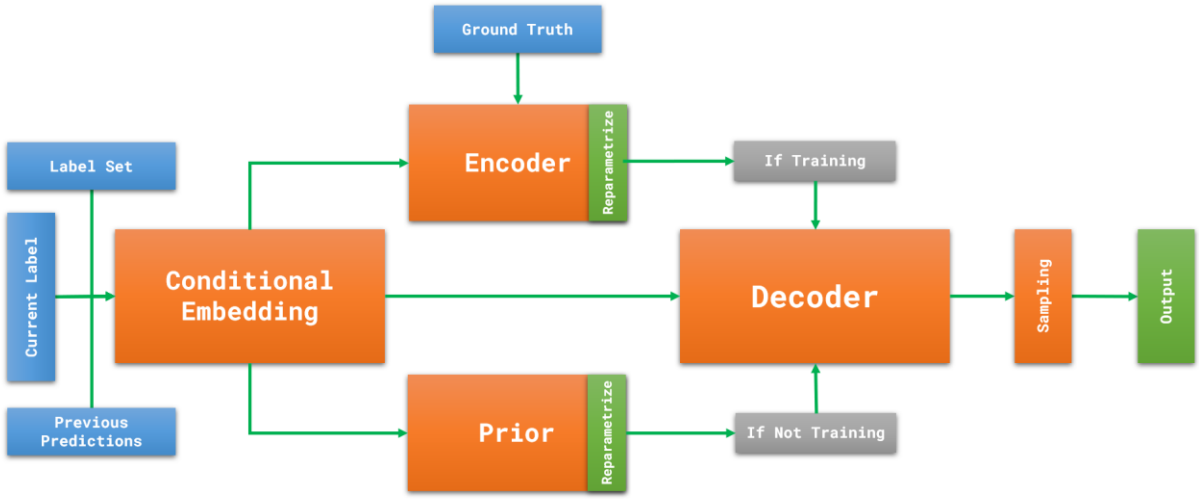


Fig 1.2. Flow Diagram of Both Count and BBox VAE

Conditional Embedding		
Layer	Dimension	Activation
FC1(Label set)	M → 128	ReLU
FC2	128 →128	ReLU
FC3(Current Label)	M → 128	ReLU
FC4	128 →128	ReLU
FC5(Previous predictions)	M → 128	ReLU
FC6	128 →128	ReLU
Concatenation (FC2,FC4,FC6)	128 →384	None
FC7	384 →128	None

Decoder		
Layer	Dimension	Activation
Concatenation (Latent Space +Conditional embedding)	32+128→160	None
FC1	160→ 128	ReLU
FC2	128 → 64	ReLU
FC3(Output)	64 → N[1]	None

Prior		
Layer	Dimension	Activation
FC1(Conditional Embedding)	128 → 32	ReLU
FC2	32→ 32	None
FC3(Latent Space)	32→ 32	None

Encoder		
Layer	Dimension	Activation
FC1(Ground Truth)	N[1] → 128	ReLU
FC2(Conditional Embedding)	128 → 128	None
FC3	256 → 32	ReLU
FC4	32→ 32	None
FC5(Latent Space)	32→ 32	None

Table 1: Structure of 4 different parts of model

1.4. CountVAE

This is the first part of the LayoutVAE model; it takes the label set as input and predicts the counts of bounding boxes for corresponding labels. The input is provided as multilabel encoding. For example, if $\boldsymbol{L}$ be a label set than of length $\boldsymbol{M}$ where M is number of classes than L_i= 1 if we want to predict the counts of i^{th} class. Loss functions we used are KL Divergence and poisson negative log likelihood loss (Available in pytorch losses).

1.5. BBoxVAE

This the second part of the model was BBoxVAE with LSTM based

[1] N = 1 for CountVAE and 4 for BBoxVAE

Embedding Generation. Similar to CountVAE here also previous predictions along with the label set and label counts are used as conditioning info for current predictions. Loss functions we used are KL Divergence and MSE.

1.6. Hyperparameters

Originally in the paper the model was not used upon structured layouts so we experimented with various hyperparameters in order to get better results however the number of parameters is not very large in this model. We varied the latent dimension and found that the latent dimension of 32 works pretty well.

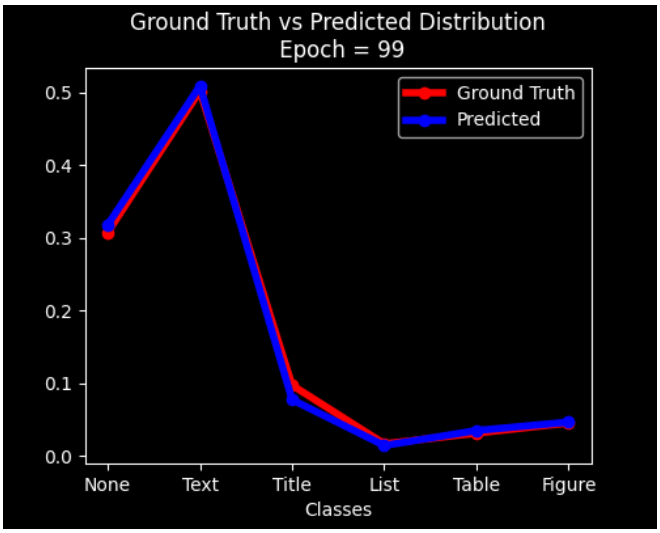
1.7. Hardware used

We trained the model on Google Collab with GPU hardware acceleration.
For a batch size of 256 with 50% PubLayNet data and validation split of 0.01 CountVAE required around 2-3 seconds and BBoxVAE required 10-15 seconds per epoch.

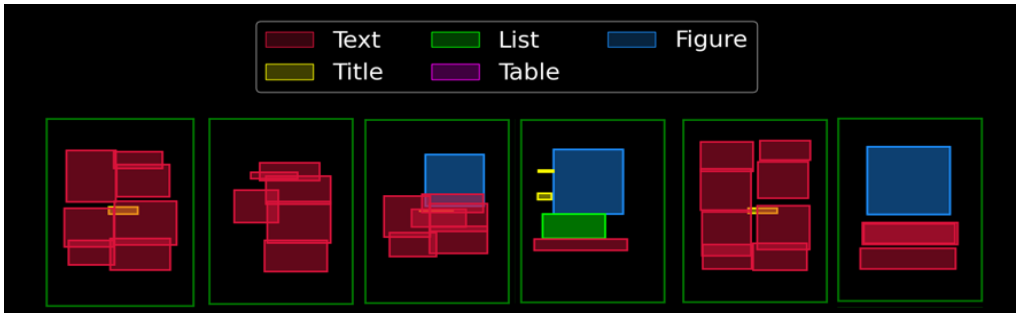
	Datasets			
Parameters	PubLayNet		RICO	
	CountVAE	BBoxVAE	CountVAE	BBoxVAE
Data Size	50%	50%		
Learning Rate	10^{-6}	10^{-4}		
Epochs	100	150		

1.8. Results

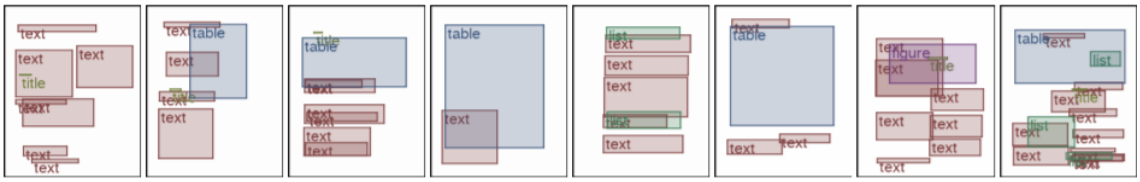
1.8.1. CountVAE



1.8.2. BBoxVAE



(a)



(b)

Fig 1.3 : (a) some results of LayoutVAE we obtained (b) Results of Layout VAE shown in (Layout Transformer) [3]

1.9. Discussion

1.9.1. What was hard

The original model was not made for structured layouts so we did not get any help from the results mentioned in the paper. We looked for the results in other papers like Layout Transformer[3]. Also, we faced difficulties in implementing the model in an autoregressive manner. Initially our idea was to implement the model in TensorFlowV2 but later on we switched to PyTorch because of its ease.

1.9.2. Changes that made the day!

Sorting(mentioned in paper) :- Without sorting we found that model was only able to predict single column layouts and it failed miserably on double or multiple columns. But once we sorted the data left to right it started to predict double column layouts. This reduced the alignment loss by one-third but also increased the overlap and IoU.

2. Layout Transformer

2.1. Introduction

Layout Transformer is a model proposed for generating structured layouts which can be used for documents, websites, apps, etc. It uses the decoder block of the Transformer Model, which is able to capture the relation of the document boxes with the previously predicted boxes (or inputs). Since it is an auto-regressive model, it can be used to generate entirely new layouts or to complete existing partial layouts.

The paper also emphasized on the fact that this model performs better than the existing models (at that time) and is better in the following aspects:

- Able to generate layouts of arbitrary lengths
- Gives better alignment due to the discretized grid
- Is able to effectively capture the relationships between boxes in a single layout, which gives meaningful layouts

2.2. Methodology

We implemented the Layout Transformer Model using TensorFlow v2. The mathematical expressions were not given explicitly either in the Layout Transformer Paper, or in the Self Attention Paper. To get an aid, we found a YouTube Channel, who explained each and every layer of the Transformer block very well, along with their mathematical expressions. We used some of the pre-defined layers of the keras along with some self-implemented layers to construct the whole model.

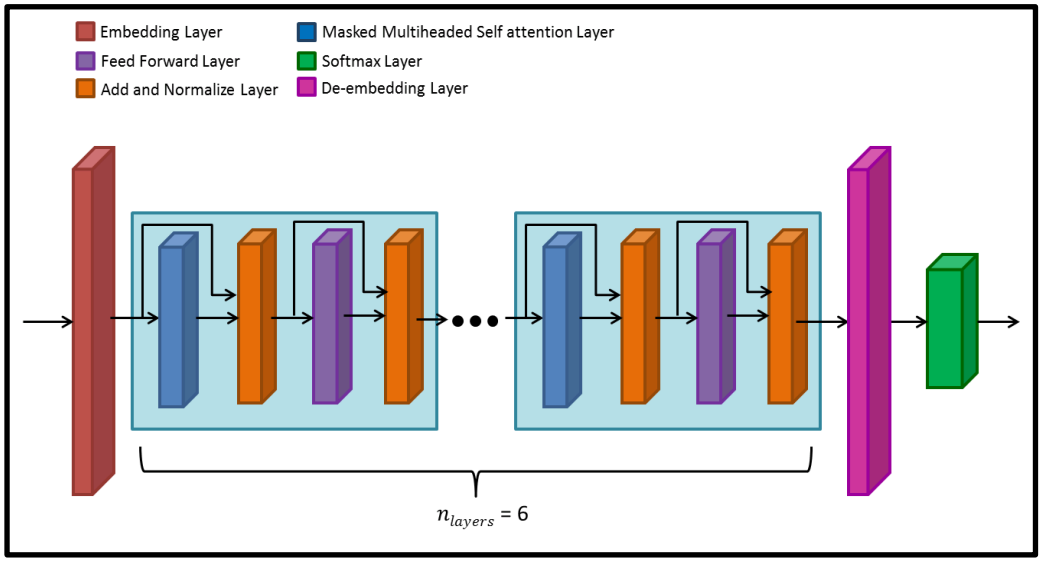
2.2.1. Model

The model consists of the following layers:

Layer	Description	Hyperparameters
Embedding	Projects each one hot encoded input box to high dimensional space	n_{units} = Output dimension
Masked Multi-headed Self Attention	A self-attention layer with multiple heads and masking (to get the relation only from the previous boxes)	n_{heads} = Number of heads
Feed Forward	A layer which has two dense layers along with ReLU activation in between and a	d_{ff} = Dimension of in-between dense layer dropout = Dropout rate

	dropout	
Add and normalize	Layer for adding and normalizing (acts as a skip connection)	None
De-embedding	Converts the high dimensional space to original dimensions	n_{units} = Output dimension
SoftMax Activation	Converts the outputs to probability distribution	None

The model architecture for the layout transformer is shown in the figure below.



The Embedding and de-embedding layers were implemented using the inbuilt dense layers of TensorFlow.

For the masked multi-headed self-attention layers, the below expressions were used.

$$X = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, \quad Y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}$$

$$Q = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}, \quad K = \begin{bmatrix} k_1 & k_2 & \dots & k_n \end{bmatrix}, \quad V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

$$Query : Q = W_Q X$$
$$Keys : K = W_K X$$
$$Values : V = W_V X$$
$$Z = K^T Q / \sqrt{d}$$

Weights and averages using column wise SoftMax:

$$Weights : W = softmax(Z)$$

$$W = \begin{bmatrix} 1 & sm_1(Z_{12}, Z_{22}) & sm_1(Z_{13}, Z_{23}, Z_{33}) & sm_1(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ 0 & sm_2(Z_{12}, Z_{22}) & sm_2(Z_{13}, Z_{23}, Z_{33}) & sm_2(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ 0 & 0 & sm_3(Z_{13}, Z_{23}, Z_{33}) & sm_3(Z_{14}, Z_{24}, Z_{34}, Z_{44}) & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

$$New\ embedding : Y = VW$$

These new embeddings are separately generated using n_{heads} numbers of W_K, W_Q and W_V weight matrices.

$$Y_1 = V_1 W_1; Y_2 = V_2 W_2; \cdots; Y_{n_{heads}} = V_{n_{heads}} W_{n_{heads}}$$

They are now concatenated and multiplied with another weight matrix W_O to get the original dimension space.

$$W_O \times \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_{n_{heads}} \end{bmatrix} = Y$$

The add, normalize and SoftMax layers were implemented using the inbuilt layers of TensorFlow.

2.2.2. Datasets

2.2.3. Hyperparameters

We used the hyperparameters mentioned in the paper [3]. Following are the values:

Parameter	Value	Description
$n_{anchors}$	32×32	Number of grid cells (corresponding to locations and size of bounding boxes).
d	512	Model dimension to which the inputs are converted using the embedding layer.
n_{layers}	6	Number of self-attention layers.
n_{heads}	8	Number of heads in each of the masked self-attention layers.
d_{ff}	2048	Number of units in the feedforward layer.
$P_{dropout}$	0.1	Dropout at the end of each feedforward layer for regularization.

2.2.4. Hardware used

2.3. Results

The outputs for the PubLayNet dataset are shown below. The model was trained for 10000 documents, 300 epochs and 1e-5 learning rate. Callbacks were used to change the LR in training.

2.4. Discussion

2.4.1. What was hard

Lack of Mathematical expressions

What is embedding layer

2.4.2. Changes that made the day!

Sorting
Label Smoothing

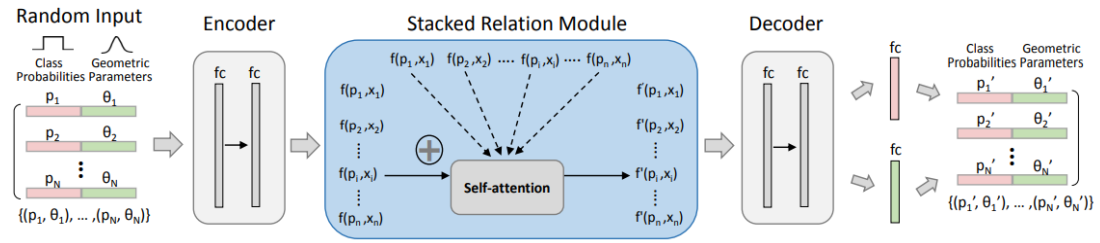
3. Layout GAN

3.1. Introduction

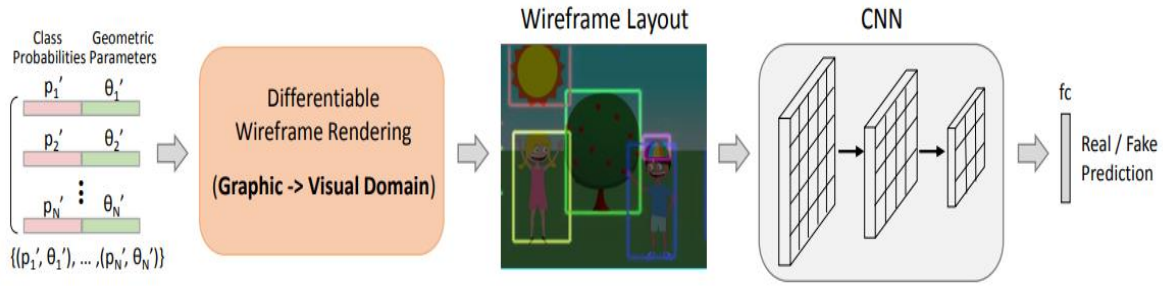
A layout is a set of graphical elements that are placed together in an aesthetic and meaningful way to convey some information. Layouts are important graphical communication tools for graphical designs and user interfaces. Producing layouts has been a topic of interest and research for quite some time. However, this has been a challenging task as the present real-world layouts are highly variable and generating a layout with proper semantic relationship between elements was a challenging task.

Recently a paper concerning layout generation was released concerning layout , Layout GAN which uses a GAN network , with the generator taking randomly sampled inputs (class probabilities and geometric parameters) as parameters , arranging them and thus producing refined geometric and class parameters.

It has two kinds of discriminators mentioned , Relational discriminator which has a similar structure to the generator has effective results but cannot detect the spatial relationships well between the elements.
On the other hand, Wireframe rendering discriminator uses a wireframe rendering and CNNs with a fully connected dense layer for real/fake prediction and works as a visual judge for distinguishing visual patterns.



Generator Model architecture



Wireframe Rendering Discriminator Model architecture

3.2. Scope of reproducibility

In this review we examine and reproduce the work of the proposed model Layout GAN. The aim is to reproduce the results obtained by the authors and the results obtained from the paper for the purpose of comparing it with other layout Generation methods .
Authors have not published the datasets they have used for single column layouts

3.3. Methodology

Some of the source code which we have used for reproducing the paper was available on Tensorflow-1 as official implementation in the following GitHub repository[link]. our implementation was made using Tensorflow-2 following mostly the same architecture and hyperparameters. Official implementation only includes the code for sanity test on MNIST while there were no results provided in the repository for layout generation.

3.4. Model descriptions

3.4.1. Generator

The generator takes as input graphic elements with randomly sampled class probabilities from Uniform distribution and geometric parameters from Gaussian distribution.an encoder consists of four 1x1 cov2d followed by batch normalization and activation at the end. Followed by two stacked relational modules and a decoder consists of again four 1x1 cov2d and again followed by two stacked relational modules and sigmoid activation layer was used at the end to obtain geometric parameters and probabilities.

3.4.2. Discriminator

Discriminator renders the given input and the rendered input will be passed to a CNN which was implemented as two 1x1 cov2d with batch normalization and leaky relu activation followed by a flatten layer and two fully connected layers.
For rendering the input, we have used the same function which was available in the original implementation.

3.5. Dataset

Others have shown results on single column layouts dataset which was not publicly available so for layouts generation we have used publaynet dataset. We have extracted layouts which have at most 9 classes and are of single column. We have trained on 70,000 such layouts.

3.6. Hyperparameters

The authors of the original paper mention specific values for some of the hyperparameters. However, for other hyperparameters we experimented with , we obtained highly variable with imperfect results.
Some specific hyperparameters were :

- The geometric parameters were sampled from normal gaussian distribution with mean 0.5 and standard deviation 0.15. For the class probability distribution, we experimented with random one hot encoding as sampling from uniform distribution from (0.0, 1.0).
- The Generator learning rate of 10^{-5} .
- The discriminator learning rate of 10^{-5} with exponential decay with a decay rate of 0.1 for every 20 epochs.
- We optimized the generator two times and the discriminator one time for each batch.

3.7. Experimental setup and code

3.8. Computational requirements

All the experiments were run on the HPC servers provided by Indian institute of technology, Ropar. The system provides NVIDIA Tesla P100 which was used to train the models. Training the model for 30 epochs took around 2 hrs.

3.9. Results

MNIST Obtained

Generator being trained twice the Discriminator

g_lr = 10^{-5} **, d_lr =** 10^{-5}

MNIST Obtained

Discriminator being trained twice the Generator (with WGAN loss and weight clipping)

g_lr = 10^{-3} **, d_lr =** 10^{-6}

MNIST Reported

3.10. Discussion

3.10.1. What was hard

- Training hyperparameters were not clearly mentioned in the paper.
- Loss used was not clearly mentioned in the paper.
- Very Unstable to train, small changes lead to insignificant results.(specially for the PubLayNet dataset)
- Not clearly mentioned about how many self-relation modules have to be used for layout datasets.

3.10.2. Changes that made the day!

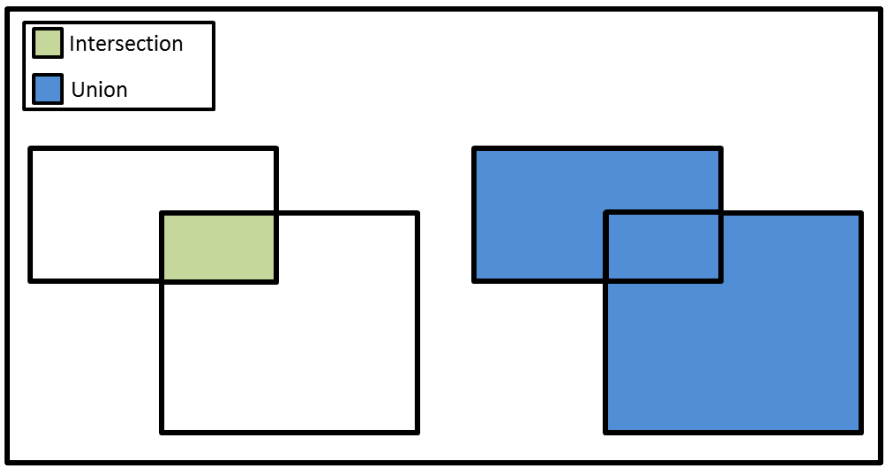
- Adding extra self-relation modules for the PubLayNet dataset .
- Training the Generator more than the discriminator .

4. Quantitative Comparison

The following metrics were used to make comparison between the three models:

4.1. Intersection over Union (IoU)

As the name suggests, the intersection over the union of boxes is calculated pairwise and are then added together. The overall IoU is averaged over all the documents.



For the kth document in the data, the iou L_k is calculated as follows:

$$L_k = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{S_i \cap S_j}{S_i \cup S_j}$$

Where n is the total number of boxes in the document.

For the whole data, the loss(IoU) is calculated as follows:

$$L = \frac{1}{N} \sum_{k=1}^{N} L_k$$

Where N is the total number of documents in the data.

A well-structured document has a very low IoU because there is generally no overlapping.

4.2. Overlapping Loss

Overlapping loss is defined as the ratio of overlapping area by the box area. It is also calculated pairwise, added together and then averaged for all documents. Related expressions are given below:

$$L_k = \sum_{i=1}^{n} \sum_{\forall j \neq i} \frac{S_i \cap S_j}{S_i}$$

$$L = \frac{1}{N} \sum_{k=1}^{N} L_k$$

For well-structured documents (like the PubLayNet Data), the overlapping loss will be low as there is typically no overlapping. However, for other datasets it is not legit. In general, overlapping loss for the predicted data should be near to the overlapping loss of the original data.

4.3. Alignment Loss

The alignment of boxes is also very important. Adjacent elements (boxes) are usually in six possible alignment types: Left, X-center, Right, Top, Y-center and Bottom aligned. Denote $\theta = (x^L, y^T, x^C, y^C, x^R, y^B)$ as the top-left, center and bottom-right coordinates of the predicted bounding box, we encourage pairwise alignment among elements by introducing an alignment loss:

$$L_alg = \sum_{i=1}^{n} min \begin{pmatrix} g(\Delta x^L{}_i), & g(\Delta x^C{}_i), & g(\Delta x^R{}_i), \\ g(\Delta y^L{}_i), & g(\Delta y^C{}_i), & g(\Delta y^R{}_i) \end{pmatrix}$$

Where $g(x) = -log(1-x)$ and $\Delta x^*{}_i = min\,|x^*{}_i - x^*{}_j| \quad \forall\, j \neq i$; (* = L,C,R).

The overall loss was normalized w.r.t. to the original data.

Comparison of Models using above metrics (for PubLayNet Dataset) :

	Overlap	IOU	Alignment
Original Data	1.000000	1.000000	1.000000
Layout GAN	1172.005234	2745.437529	1.164882
LayoutVAE	119.320127	185.864381	3.493406
Layout Transformer	1.090315	1.422297	0.739862

5. References

[1] Layout GAN - Generating Graphic Layouts with Wireframe Discriminators

[2] LayoutVAE - Stochastic Scene Layout Generation from a Label Set

[3] Layout Generation and Completion with Self-Attention

[4] Attribute-conditioned Layout GAN for Automatic Graphic Design

[5] Variational Transformer Networks for Layout Generation

[6] Lennart Svensson : A series of videos on Self Attention

[7] Auto-Encoding Variational Bayes Diederik P Kingma, Max Welling