Neural Design Network: Graphic Layout Generation with Constraints Supplementary Material

Hsin-Ying Lee*[2], Lu Jiang[1], Irfan Essa[1,4],
Phuong B Le[1], Haifeng Gong[1], Ming-Hsuan Yang[1,2,3], Weilong Yang[1]

[1]Google Research [2]University of California, Merced
[3]Yonsei University [4]Georgia Institute of Technology

1 Overview

In this supplementary material, we first describe the graph convolution network we used in this work. Second, we further illustrate the training process of the relation prediction module. Third, we present the detailed configurations of the proposed framework. Finally, we show additional results.

2 Graph Convolution Network

Our graph convolution network takes as inputs features of nodes and edges, and outputs the updated features. Given as input a graph $G = (O, E)$ with vectors of objects $f_{o_i} \in \mathbb{R}^{D_{in}}$, where $i = 1 \sim |O|$, and vectors of relations $f_{r_i} \in \mathbb{R}^{D_{in}}$, where $i = 1 \sim |E|$, we aim to output the updated vectors $f'_{o_i} \in \mathbb{R}^{D_{out}}$ and $f'_{r_i} \in \mathbb{R}^{D_{out}}$, where D_{in} and D_{out} are dimension of input and output vectors, respectively. The graph convolution kernel consists of three functions g_s, g_p, and g_o to deal with features of subject, predicate, and object, respectively. These function take as input a triple $(f_{o_i}, f_{r_k}, f_{o_j})$ for an edge and outputs updated vectors. For relations, the updated vectors is obtained directly from g_p:

$$f'_{r_k} = \{g_p(f_{o_i}, f_{r_k}, f_{o_j}) : (o_i, r_k, o_j) \in E\}. \tag{1}$$

For objects, since an object may be involved in multiple relationships, the updated vectors should take all participated edges into consideration. We use a symmetric pooling function p to take as input a set of vectors and output a single vector. Concretely, for o_i, the feature sets for o_i as subject, the feature sets for o_i as object, and the final updated vectors can be formulated as:

$$\begin{aligned} F_i^s &= \{g_s(f_{o_i}, f_{r_k}, f_{o_j}) : (o_i, r_k, o_j) \in E\} \\ F_i^o &= \{g_o(f_{o_j}, f_{r_k}, f_{o_i}) : (o_j, r_k, o_i) \in E\} \\ f'_{o_i} &= p(F_i^s, F_i^o). \end{aligned} \tag{2}$$

Figure 1 illustrates an example for a single graph convolution layer.

* Work done during their internship at Google Research.

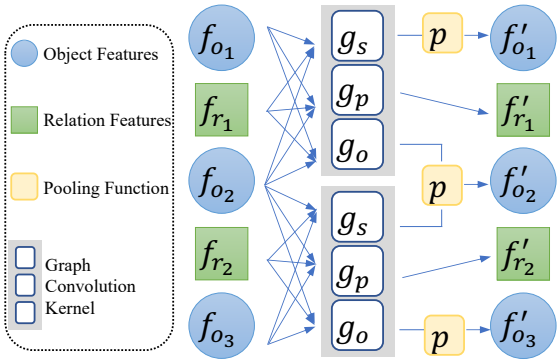

Fig. 1: **Illustration of graph convolution layer.** The example consists of three objects with two edges. The features are passed into the graph convolution kernel consisting of three functions. The updated relation features are directly obtained from the output of the graph convolution kernel. The updated object features are obtained with a pooling function.

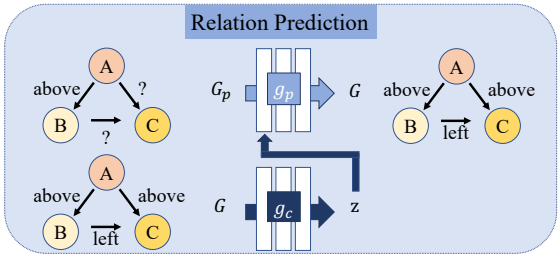

Fig. 2: **Relation prediction module.** During the training process, given a paired of a graph with partial relations G_p and a graph with complete relation G, a graph convolution network g_c first encode G into a latent vector z, then we perform a conditional graph-to-graph translation with $g_p : \hat{G} = g_p(G_p, z)$.

3 Relation Prediction Module

The Figure 2 in the paper presents the generation process of three modules in the testing stage. We further illustrate the training process of the relation prediction module in Figure 2.

4 Model Architecture

The proposed framework contains three multilayer perceptrons (h_{bb}^{enc}, h_{bb}^{dec}, and h_{pred}) and five graph convolution networks (g_c, g_p, g_{enc}, g_{update}, and g_{ft}). We present the architecture of sub-networks in Table 1.

Table 1: **Detailed configuration of the proposed network.** We use the following abbreviation: gconv(D_{in}, H, D_{out}) is a graph convolution with input dimension D_{in}, hidden dimension H, and output dimension D_{out}. FC($D_{in} \rightarrow D_{out}$ is a fully-connected layer with input dimension D_{in} and output dimension D_{out}.

Layer	g_c, g_{ft}
1	gconv(64, 512, 128)
2	gconv(128, 512, 128)
3	gconv(128, 512, 128)
4	gconv(128, 128, 128))

Layer	g_p
1	gconv(160, 512, 128)
2	gconv(128, 512, 128)
3	gconv(128, 512, 128)
4	gconv(128, 128, 128))

Layer	g_{enc}
1	gconv(64, 512, 128)
2	gconv(128, 512, 128)
3	gconv(128, 512, 128)

Layer	g_{update}
1	gconv(128, 512, 128)

Layer	h_{pred}
1	FC(128→512)
2	FC(512→$\|R\|$))

Layer	h_{bb}^{enc}
1	FC(4→128)
2	FC(128→128))
3	FC(128+128→32))
4_{mu}	FC(32→32))
4_{var}	FC(32→32))

Layer	h_{bb}^{dec}
1	FC(32+128→128)
2	FC(64))
3	FC(64→4))

5 Additional Results

We present additional results in Figure 3.

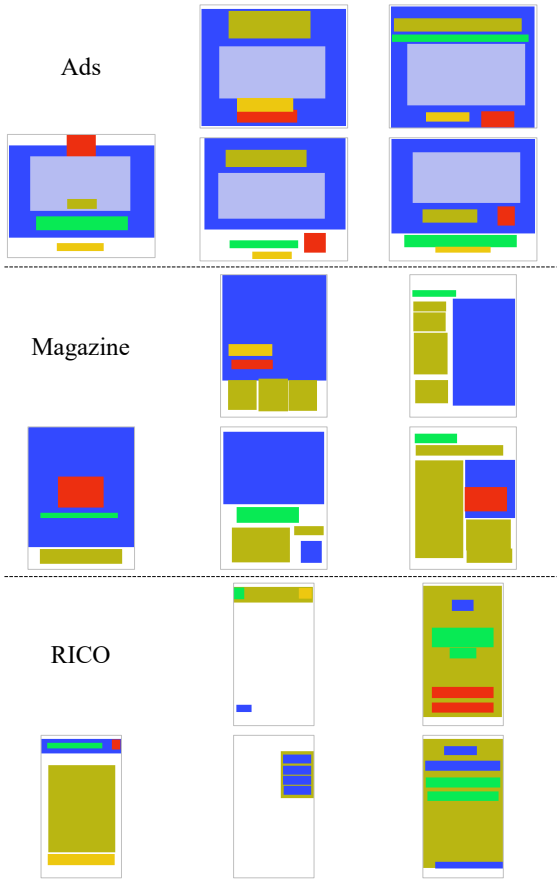

Fig. 3: **Additional results.** Samples generated by the proposed framework. All layouts are generated given desired components and randomly decided partial constraints.