Learning Structural Similarity of User Interface Layouts using Graph Networks

Dipu Manandhar[1], Dan Ruta[1], and John Collomosse[1,2]

[1] CVSSP, University of Surrey — Guildford, UK.
[2] Adobe Research, Creative Intelligence Lab — San Jose, CA.
{d.manandhar, d.ruta}@surrey.ac.uk, collomos@adobe.com

Abstract. We propose a novel representation learning technique for measuring the similarity of user interface designs. A triplet network is used to learn a search embedding for layout similarity, with a hybrid encoder-decoder backbone comprising a graph convolutional network (GCN) and convolutional decoder (CNN). The properties of interface components and their spatial relationships are encoded via a graph which also models the containment (nesting) relationships of interface components. We supervise the training of a dual reconstruction and pair-wise loss using an auxiliary measure of layout similarity based on intersection over union (IoU) distance. The resulting embedding is shown to exceed state of the art performance for visual search of user interface layouts over the public Rico dataset, and an auto-annotated dataset of interface layouts collected from the web. We release the codes and dataset[3].

1 Introduction

Layout is fundamental to user experience (UX) design, where arrangements of user interface components form the blueprints for interactive applications. Vast repositories of UX layouts are openly shared online. The ability to easily search these repositories offers an opportunity to discover and re-use layouts, democratizing access to design expertise.

This paper contributes a novel technique for visually searching UX designs, leveraging a graph based representation that integrates both the properties of interface components and their spatial relationships. Representation learning for UX design is challenging, as layouts typically exhibit complex geometry and even nesting of interface components; properties we encode explicitly within our representation. We propose a triplet architecture to learn a metric search embedding for layout similarity from this representation, leveraging a hybrid encoder-decoder backbone that combines a graph convolutional network (GCN) encoder with a convolutional network (CNN) decoder.

Representation learning is a fundamental computer vision task, that has previously been tackled for UX layout search by leveraging pixel-based (raster) renderings of designs, for example to train auto-encoders (AEs) [20]. Whilst

[3] https://github.com/dips4717/gcn-cnn

such unsupervised representations are convenient to train, they are typically inaccurate at recalling detail in the design, do not explicitly encode common UX design properties (such as component nesting), and do not encourage metric properties in the search embedding. We mitigate against this using a dual loss that combines a reconstruction loss and a triplet loss that weakly supervises learning via a weighted auxiliary metric, based upon intersection over union (IoU). Our core technical contributions are two-fold:

1. Graph Representation for UI Layout. We encode the semantic and geometric properties of user interface controls and their geometric relationships via a graph. We encode this representation via a GCN with self-attention to learn a latent representation for UI layout.

2. GCN-CNN Architecture for Layout Search. We present a novel siamese GCN-CNN architecture for learning a metric embedding for layout search. The embedding delivers state-of-the art results on two UX design datasets.

We demonstrate a search application of our learned embedding using the public RICO dataset of mobile UX designs [7]. We also search a new dataset of UX designs collected from the web, annotated automatically via a Faster-RCNN detector trained on RICO, that we release as a further contribution of this work.

2 Related Work

Layout has been primarily studied through the lens of automated design and reflow tasks [16] within the domains of document pagination and graphic design. The prediction of aesthetic score for document layout is a well studied problem with early work exploring heuristics relating to white space and content balance [14], with such metrics being leveraged to drive layout decisions in [8, 9]. Subsequently, optimization strategies leveraging learnable metrics from exemplar designs [22, 23] and from gaze [34] has been explored. Representation learning was explored for synthesis in LayoutGAN, where layouts were learned using a differentiable wireframe renderer [19] operating over a list of layout components and their geometric parameters. Whilst a variable length representation is unsuitable for search via deep metric learning, LayoutGAN showed generation of several document layout types, including UX designs. Generative approaches to layout were also explored for salience guided reflow of graphic designs [5].

Layout search is more sparely researched, and limited public datasets exist. Component detection has been combined with learned design heuristics to parse graphic layouts for re-use [35, 30] and even for code generation [2]. Rico is a crowd-annotated dataset [7] of mobile app screenshots, and is most closely aligned to our work in that it also proposed a classical MLP autoencoder to learn a latent space for search – using rasterized representations of UX layout. Liu et al. similary explored convolutional autoencoders for layout search on Rico [20]. Whilst such embeddings do not require supervision to train, they are not constrained to metric properties suitable for similarity search and require layout rasterization as an intermediate step to build the search index. Our work is unique in leveraging a graph convolutional network (GCN) [3] to encode a graph-based representation

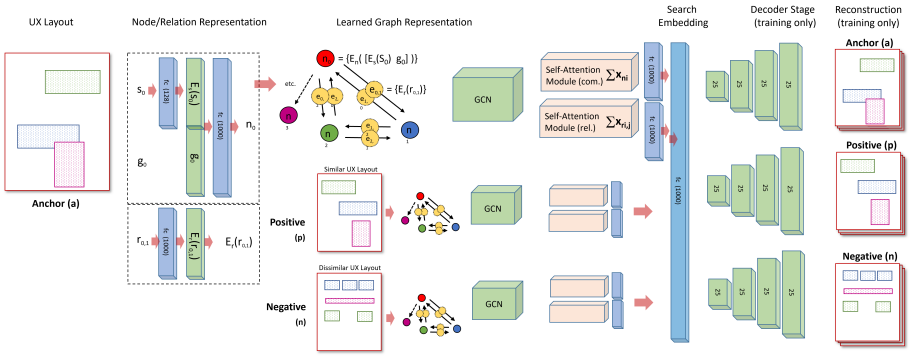

Fig. 1. Proposed GCN-CNN encoder-decoder architecture to learn a latent search embedding for UX layout. The input is a set of bounding boxes with associated class labels, encoding the relative positions and types of UI component. A combination of linear layers, GCN and self-attention map these to the latent space. At training time, a 25-channel raster representation of the UI is decoded from the latent space. Triplets of layouts are presented through this backbone in siamese architecture, to learn the latent representation which forms the search embedding.

of UX layout, which we show to signficantly outperform raster layout encoders both in unsupervised training. This performance is even more pronounced when combining this approach with triplet (siamese) learning [29] commonly used to learn deep metric embeddings for visual search of photographs [32, 25, 10] and sketches [27, 4].

Graph convolutional networks (GCN) [3, 28, 6] have recently gained popularity in analysing non-Euclidean data for deep learning, e. g. social graphs, communication and traffic networks [36]. Scene graphs are emerging as a robust representations for encoding objects and their relationships, and embedded via have been applied to automatic captioning [11], scene [17] and action recognition [13], and image synthesis from coarse layout descriptions [31, 1]. GCNs have also been explored to search for visually similar scenes in [33]. Our work also exploits GCN for visual similarity, addressing for the first time search of UX layouts by encoding user interface components and their geometric relationships.

3 Method

The architecture of the proposed GCN-CNN framework is shown in Fig. 1. It consists of triplet (siamese) backbone which constitutes a graph-based encoder encoding the input UI layout into a latent space, and a transposed convolutional decoder that reconstructs a multi-channel raster rendering of the layout. Our network is trained using a dual loss,

$$L_{total} = \sum_{x \in \{a,p,n\}} L_{rec}(x, x') + \lambda L_{tri}(a, p, n) \tag{1}$$

where (a, p, n) is a triplet of anchor, positive, and negative UI layouts. (a, p) forms a positive pair representing similar layouts, and (a, n) is a negative pair dissimilar layouts. $L_{rec}(x, x')$ is reconstruction loss which may used to train the GCN-CNN in an unsupervised way, but we show performs better when combined with $L_{tri}(a, p, n)$ as triplet loss trained in a weakly supervised manner to also encourage the metric property in search embedding. We now describe in greater detail our graph representation for layout (subsec.3.1), its encoding via the network (subsec. 3.2) and the training methodology and loss (subsec. 3.3).

3.1 Graph representation

We describe a UI layout with its components and their geometric properties. Formally, we represent UI layout, with height h and width w, as a spatial graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{c_1, \cdots, c_i, \cdots, c_\kappa\}$ is set of nodes representing its κ UI components, and $\mathcal{E} = \{e_{11}, \cdots, e_{ij}, \cdots e_{\kappa\kappa}\}$ is the set of edges that denoting the existence of a relationship between them. Each node carries two types of information. The first feature is associated with semantic property s_i; a one-hot vector denoting the UI component class. Second, geometric property $\mathbf{g}_i$ capturing the spatial location of the component in UI are encoded; we adapt the scheme of [12]. Let (x_i, y_i) and (w_i, h_i) be the centroid, width and height of the component c_i, and $A_i = \sqrt{w_i h_i}$, then the geometric feature g_i is

$$\mathbf{g}_i = \left[\frac{x_i}{w}, \frac{y_i}{h}, \frac{w_i}{w}, \frac{h_i}{h}, \frac{A_i}{wh}\right]. \tag{2}$$

Next, we define the edges features $\mathbf{r}_{ij}$ associated with edge e_{ij} using the pairwise geometric features between components c_i and c_j given by Eq.(3)

$$\mathbf{r}_{ij} = \left[\psi_{ij}, \theta_{ij}, \frac{\Delta x}{A_i}, \frac{\Delta y}{A_i}, \frac{w_j}{w_i}, \frac{h_j}{h_i}, \frac{1}{D}\sqrt{\Delta x^2 + \Delta y^2}\right] \tag{3}$$

where $\Delta x = x_j - x_i$ and $\Delta y = y_j - y_i$ are the x- and y- shifts between the components and constant $D = \sqrt{w^2 + h^2}$ normalises against the diagonal. In addition, the feature $\mathbf{r}_{ij}$ incorporates various geometric relations such as relative distance, aspect ratios, orientation $\theta = atan2\left(\frac{\Delta y}{\Delta x}\right) \in [-\pi, \pi]$. We explicitly include a *containment feature* ψ_{ij} taking into account the Intersection over Union (IoU) between components capturing the nesting of the UI components:

$$\psi_{ij} = \frac{M(c_i) \cap M(c_j)}{M(c_i) \cup M(c_j)} \tag{4}$$

where M(.) indicates the mask of the single component (ψ_{ij} is computable via bounding box intersection without rasterization). We explore both undirected and directed graph representations for UIs. For undirected representation, we create a single edge between two components c_i and c_j i.e. $\mathcal{E} = \{e_{ij}\}$ for $\forall i, j = 1, 2, ..., \kappa$ such that $j \geq i$. In directed representation, we create all the possible edges between the nodes i. e. two directed edges are created between the pair c_i and c_j as shown in Fig. 1. For the associated geometric features, note that $\mathbf{r}_{ij} \neq \mathbf{r}_{ji}$.

3.2 GCN-CNN Encoder-Decoder

Layout Encoder We propose a hybrid GCN-CNN encoder-decoder architecture to learn the latent space in an unsupervised manner. The GCN encoder maps the layout graph into embedding space. The node features $\mathbf{n}_i$ in the graph hold both the semantic class label s_i as well as the geometric property $\mathbf{g}_i$ of the UI component c_i. The semantic class s_i is first encoded into N_s trainable embeddings, $N_s = 25$ being the number of semantic classes of UI components (subsec 4.1). The geometric feature $\mathbf{g}_i$ is concatenated with the semantic embedding, and projected by a linear layer to obtain the node features $\mathbf{n}_i$

$$\mathbf{n}_i = E_n\left([E_s(s_i)\ \ \mathbf{g}_i]\right)) \tag{5}$$

where E_s is the embedding layer that learns the UI class embeddings and E_n is a linear layer that projects the semantic and geometric features into node feature $\mathbf{n}_i$. Similarly, the edge features $\mathbf{r}_{ij}$ are projected by $E_r(\mathbf{r}_{ij})$. Next, the node features and the edge (relation) features are operated by graph convolutional networks $g_n(\cdot)$ and $g_r(\cdot)$. The node and relational feature outputs of the GCN network are computed by

$$\mathbf{x}_{n_i} = g_n(\mathbf{n}_i) \tag{6}$$

$$\mathbf{x}_{\mathbf{r}_{ij}} = g_r([\mathbf{n}_i\, E_r(\mathbf{r}_{ij})\, \mathbf{n}_j]) \tag{7}$$

The relation graph network g_r operates on tuples $< \mathbf{n}_i, E_r(\mathbf{r}_{ij}), \mathbf{n}_j >$ passing the information through the graph to learn the overall layout. Both $g_n(\cdot)$ and $g_r(\cdot)$ are learned via fully-connected layer passed through ReLU (Fig. 1, left). We obtain two set of features from GCNs.

$$\mathcal{X}_n = \{\mathbf{x}_{n_1}, \mathbf{x}_{n_2}, ...\mathbf{x}_{n_\kappa}\} \quad \text{and} \quad \mathcal{X}_r = \left\{\mathbf{x}_{r_{11}}, \mathbf{x}_{r_{12}}, ...\mathbf{x}_{r_{\kappa'}}\right\} \tag{8}$$

where κ and κ' are the number of components (node features) and the total number of the relationship features which vary for different UI layouts. Next, the sets of features are passed through self-attention modules which learn to pool the node features and relational features given by

$$\mathbf{f}_n^{att} = \sum_{i=1}^{\kappa} \alpha_{n_i} \mathbf{x}_{n_i} \quad \text{and} \quad \mathbf{f}_r^{att} = \sum_{i=1}^{\kappa'} \alpha_{r_i} \mathbf{x}_{r_i}; \tag{9}$$

$$\alpha_{n_i} = \frac{\exp(\mathbf{w}_n^\intercal \mathbf{x}_{n_i})}{\Sigma_{l=1}^{\kappa} \exp(\mathbf{w}_n^T \mathbf{x}_{n_l})} \quad \text{and} \quad \alpha_{r_i} = \frac{\exp(\mathbf{w}_r^\intercal \mathbf{x}_{r_i})}{\Sigma_{l=1}^{\kappa'} \exp(\mathbf{w}_r^T \mathbf{x}_{r_l})} \tag{10}$$

where, α_{n_i} and α_{r_i} are attention weights learned with $\mathbf{w}_n^\intercal$ and $\mathbf{w}_r^\intercal$ parameters.

Subsec 4.4 compares learnable pooling via this self-attention module, with 'Average' pooling commonly used in CNN encoder-decoder architectures to read-out latent features. Further, we also explore 'Inverse' pooling where the weights are inversely proportional to the area of UI components; the motivation being to prioritize the small UI components and capture them well into the representation. In all cases, we obtain a $d-$dimensional latent embedding that encodes the UI layout; $\mathbf{f}_e = E_e([\mathbf{f}_n^{att}, \mathbf{f}_r^{att}])$ where E_e is the final linear layer that outputs the embeddings. We also explore choice of d in subsec 4.4.

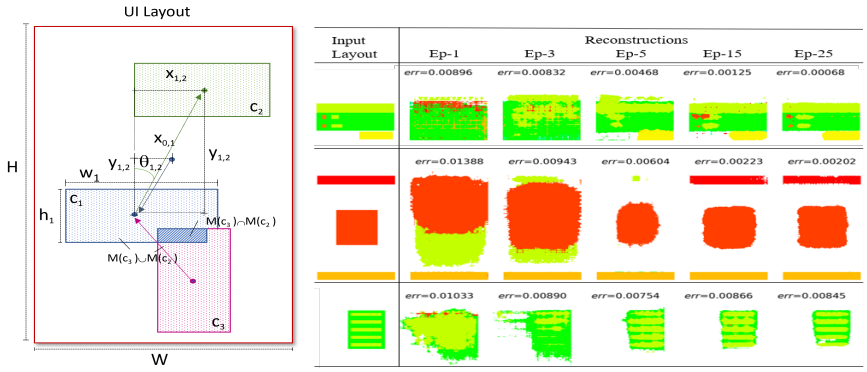

Fig. 2. Left: Schematic of UI layout, showing the features encoded in our graph representation for individual interface components c_i and their geometric relationships. Right: Visualization of raster reconstructions rendered from a RICO graph layout input by the GCN-CNN network. The 25-C decoded raster and input UI layout are projected to false color RGB space where different colors represent different UI components. Layouts are progressively reconstructed with higher fidelity (lower error) as the GCN-CNN optimizes the reconstruction loss (Eq. (11)). Note the input layout images are here for visualization only, and are not actual input to the network.

Layout Decoder The embedding $\mathbf{f}_e$ encoded by the GCN encoder are decoded into an image raster using transposed convolutional network which have been studied for vision-related problems such as semantic segmentation [21] and saliency detection [18]. Typically, the transposed convolution (also called deconvolution) network learns increasingly localized representations while upsizing the feature maps. Our decoder network consists of 4 deconvolutional layers each consisting of 25 filters with receptive field 3×3 followed by ReLU activations. We use unpooling layer (Upsample) before each deconvolution operation to progressively increase the spatial dimension of features. Since the upsampling operation is often prone to information loss, we also experiment with the strided convolution operation with stride-2 that upsizes the feature maps without requiring to upsample/unpool features. We later show that the strided deconvolution outperforms upsampling (subsec. 4.4).

The decoder outputs a raster $\rho' \in \mathbb{R}^{H \times W \times N_s}$, N_s being the number of the semantic UI component classes, H and W are height and width empirically set to 256 and 128 respectively. N_s is set to 25, the number annotated semantic classes in RICO. We refer to the decoder output as 25-C raster in the remainder of the paper. We train the entire GCN-CNN network end-to-end, using mean square error (MSE) as the reconstruction loss (L_{rec}) between the output rater and its groundtruth layout rasterized to ρ to match the dimension of the output.

$$L_{rec}(\rho, \rho') = \Sigma_{m=1}^{25} \Sigma_{n=1}^{H} \Sigma_{p=1}^{W} (\rho_{mnp} - \rho'_{mnp})^2 \tag{11}$$

In Fig. 2, we project the 25-C raster into false color RGB-space visualizing the maximum likelihood class. This illustrates how the GCN-CNN encoder decoder

progressively learn to reconstruct the UI components in their respective locations in the layout.

3.3 Metric Learning via Triplet Training

In order to learn metric properties in embedding space which is desirable for the effective search, we propose to train a triplet-based siamese architecture of the GCN-CNN encoder-decoder as shown in Fig. 1. We refer this framework as GCN-CNN-TRI in the remainder of the paper. The input to the network is a triplet of UI layout graphs denoted by $(\mathcal{G}_a, \mathcal{G}_p, \mathcal{G}_n)$ which are anchor, positive and negative UI layouts. We subsequently denote the triplet by (a, p, n) for conciseness. Our aim is to map the similar UI layouts (a, p) into closer points in the embedding space, and separate the dissimilar ones (a, n).

Triplets commonly are selected using the ground-truth labels to form anchor-positive-negative in typical metric learning frameworks [24, 15, 29]. However, in our case, we do not have labels for UIs on layout similarity. We propose to use average intersection over union (IoU) between component bounding boxes of two layouts as a weak label for selecting the triplets. We select two layout as anchor-positive pairs if their IoU value is greater than a threshold, which is empirically set to 0.6 upon visual observations. We select any layout as negative if the IoU value is below 0.4. The triplet loss for the selected layouts (a, p, n) is given by

$$L_{tri}(a, p, n) = \Big[\|f_e^{(a)} - f_e^{(p)}\|_2 - \|f_e^{(a)} - f_e^{(n)}\|_2 + \nu\Big]_+ \tag{12}$$

where $(f_e^{(a)}, f_e^{(p)}, f_e^{(n)})$ are encoded embedding for (a, p, n), $\nu = 0.2$ is a positive margin, and $[x]_+ = \max(x, 0)$.

We train our overall framework using both reconstruction loss and triplet loss Eq.(1) typically requiring 50 epochs to converge; setting $\lambda = 0$ for first half of training, and $\lambda = 10^{-1}$ for the second using Adam optimizer with initial learning rate of 10^{-3}. The trained embedding can be efficiently compared using L2-distance to search similar layouts. We show that training with weakly supervised triplet loss consistently boosts the layout search performance of the proposed method (subsec. 4.4).

4 Experiments and Discussion

We evaluate the proposed layout search technique for UX designs (GCN-CNN-TRI), benchmarking against several ablations of our method, and two existing baselines using unsupervised non-graph based representations [7, 20].

4.1 Datasets

We evaluate over RICO [7]; the largest publicly available dataset of UX designs containing 66K screenshots of mobile apps curated by crowd-sourcing and mining

9.3K free Android apps. The screenshots are annotated using bounding boxes to create semantic view hierarchies which are each assigned to one of $N_s = 25$ classes $\mathcal{S} = [s_1, \ldots, s_{25}]$ of user interface (UI) component. We partition the dataset into 53K training samples $\mathcal{T}$, reserving a test set of 13K samples as the corpus of layouts $\mathcal{L}$ for search. An additional 50 samples are held out as a queryset $\mathcal{Q} = [Q_1, \ldots, Q_{50}]$ to retrieve UIs from the search corpus. We also evaluate over GoogleUI; a new dataset of 18.5K UX design obtained by harvesting UX designs from the web, and annotated with a FasterRCNN detector trained using $\mathcal{T}$. The purpose is to explore how well our model transfers to automatically parsed layouts from image data (subsec 4.6).

4.2 Evaluation Metrics

For each query Q_i we obtain a ranked list of layouts $R(Q_i) = [L_1, \ldots, L_k]$ for each layout in test set $\mathcal{L}$ up to result rank k. Annotating $\mathcal{L}$ is infeasible for all $\mathcal{Q}$, therefore we measure accuracy via two measures of precision over the top $k = [1, 5, 10]$ results. For baseline comparisons, we also provide a subjective evaluation via Amazon Mechnical Turk (AMT).

Mean Intersection over Union (MIoU). The mean average of the Intersection over Union (IoU) score for all queries, taken across all classes $s_j \in \mathcal{S}$:

$$\text{MIoU}(\mathcal{Q}; \mathcal{L}) = \frac{1}{\mathcal{Q}} \sum_{Q_i \in \mathcal{Q}} \sum_{j=1}^{25} \frac{S_j(Q_i) \bigcap S_j(L_i)}{S_j(Q_i) \bigcup S_j(L_i)} \tag{13}$$

where $S_j(.)$ is region of the layout occupied by components of class s_j.

Mean Pixel Accuracy (MPixAcc). We rasterize the layout L_i to a $W \times H \times N_s$) and compute the pixel-wise mean accuracy across all N_s channels against the rasterized query Q_i. This score is averaged for all queries $Q_i \in \mathcal{Q}$.

Precision @ k (P@k). For comparative evaluation with baselines, we also compute P@k curves by crowd-sourcing the relevance of each ranked result.

$$\text{P@k}(\mathcal{Q}; \mathcal{L}) = \frac{1}{k\mathcal{Q}} \sum_{Q_i \in \mathcal{Q}} \sum_{j=1}^{k} rel(L_j, Q_i) \tag{14}$$

where $rel(L_k, Q_i)$ is a binary indicator for the relevance of L_k given query Q_i:

4.3 Baseline comparisons

We compare our proposed technique with the raster-based methods proposed in [7][20] for UX design similarity search. Deka et al. [7] used an MLP-based autoencoder (AE) to reconstruct images obtained by rasterizing semantic UIs. Liu et al. [20] employed a convolutional auto-encoder (CAE) to learn the embeddings. Table 1 shows layout retrieval performances in terms of top$k$- MIoU and MPixAcc. Our GCN-CNN achieves a top-10 MIoU of 48.3% and MPixAcc of 56.5%, which is further boosted by triplet training (GCN-CNN-TRI) to 51.3% and 61.0%

Table 1. Performance comparison of baselines to the proposed method both unsupervised (GCN-CNN) and with triplet supervision (GCN-CNN-TRI) over RICO. Quantified via MIoU and MPixAcc at $k = [1, 5, 10]$. The final column reports Precision @ k for $k = [1, 5, 10, 20]$ for crowd-annotated results.

Method	MIoU (%)			MPixAcc (%)			AMT P@k (%)		
k	1	5	10	1	5	10	1	5	10
AE [7]	43.0	34.7	28.9	46.9	40.6	35.1	18.0	6.0	8.0
CAE [20]	59.5	47.1	43.9	66.6	54.3	50.8	42.0	12.0	12.0
GCN-CNN (Ours)	60.0	51.6	48.3	68.3	58.9	56.5	42.0	26.0	18.0
GCN-CNN-TRI (Ours)	**61.7**	**54.1**	**51.3**	**70.1**	**64.0**	**61.0**	**46.0**	**30.0**	**36.0**

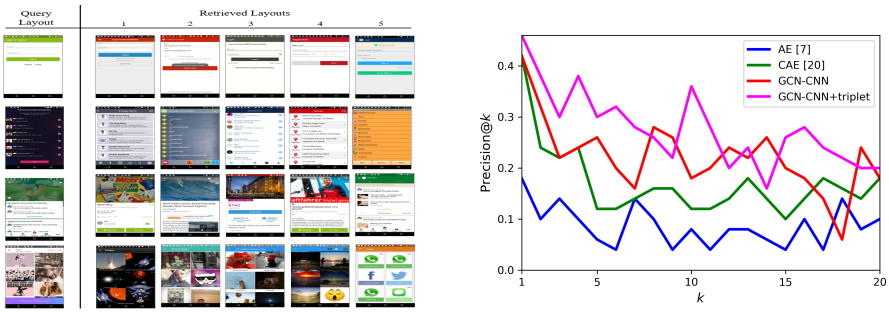

Fig. 3. Quantifying performance over RICO via AMT crowd annotation of results. Left: representative top-5 search results for a random UX design query. Right: Precision @ k curve for the proposed method GCN-CNN(-TRI) and baselines CAE [20] and AE [7]

.

respectively. Our method significantly outperforms existing methods by +22.4% [7] and +7.4% [20] in terms of top-10 MIoU.

We also report a crowd-sourced annotation undertaken on Amazon Mechanical Turk (AMT) in which 67 users (turkers) annotated the top $k = 20$ results for all 50 queries produced by all 4 methods. Turkers were asked to ignore color and the content of any text or visuals, and indicate if the structure of each UI layout matched the query. Representative search results presented to Turkers are given in Fig 3 (left). The question was asked of 5 turkers independently, yielding $5K$ annotations. A result was recorded relevant only when the majority (3 or more turkers) so indicated. Table. 1 (final col.) reports the results for $k = [1, 5, 10]$ and the P@k curve for $k = [1, 20]$ is in Fig. 3 (right). The pattern reflects that of MIoU and MPixAcc, and shows for this metric closer performance of CAE to unsupervised GCN-CNN at $k = 1$, but with the CAE performance falling away as $k > 1$. This reflects the RICO dataset content; in several cases a couple of near duplicate screens from the same app are well-matched by both CAE and GCN-CNN – but beyond these, the fine-grain structural information encoded by the GCN enables more robust matching. Overall the results clearly demonstrate the benefits of the Graph-based backbone for training a layout embedding (GCN-

Table 2. Performance of variants of the proposed method GCN-CNN for (D)irected vs. (U)ndirected graph representation, and (Str)ided vs. (Ups)ampling (dec)oder stage for embedding (dim)ensionality 2048. Unsupervised and (tri)plet supervision are evaluated at $k = [1, 5, 10]$ over RICO. Numbers in parentheses indicate triplet supervision.

Method	Dec.	MIoU (%)			MPixAcc (%)		
top-k		1	5	10	1	5	10
U(+tri)	Ups	58.0(59.0)	50.4(51.6)	48.0(49.4)	65.5(66.6)	59.4(60.7)	57.7(59.3)
U(+tri)	Str	58.9(61.6)	50.9(53.4)	48.1(51.0)	66.3(**70.2**)	59.7(62.5)	57.6(60.6)
D(+tri)	Ups	59.0(60.4)	50.2(52.9)	47.1(50.3)	66.4(69.3)	59.2(62.4)	56.3(59.7)
D(+tri)	Str	**60.0**(**61.7**)	**51.6**(**54.1**)	**48.3**(**51.3**)	**68.1**(70.1)	**61.4**(**64.0**)	**58.0**(**61.0**)

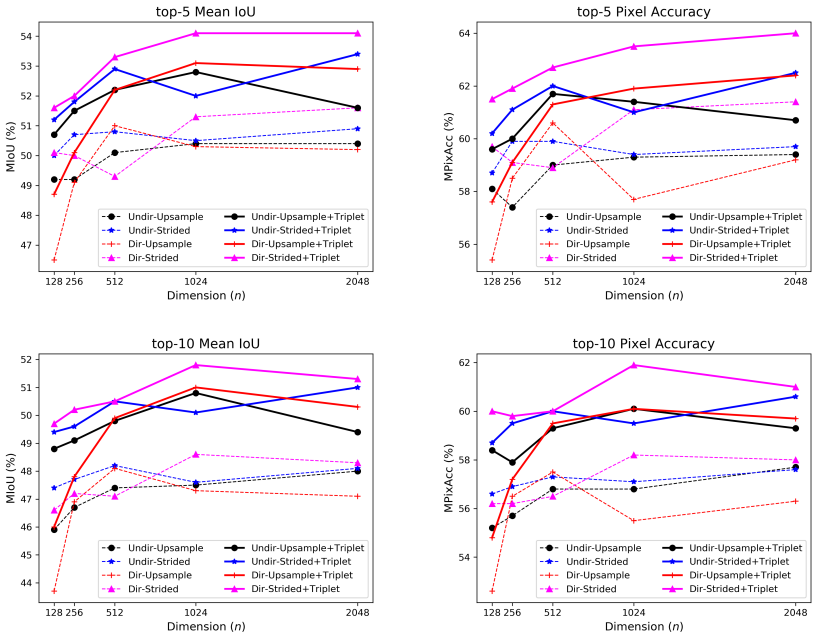

Fig. 4. Performance of the proposed GCN-CNN model for different embedding dimensionalities over RICO. Larger embedding dimensions offer better performances. For both directed and undirected graph, and upsampling and strided decoder model, training GCN-CNN with triplet loss boosts both MIoU and MPixAcc.

CNN), and the boost due to metric learning in GCN-CNN-TRI. Note that our aim is to search for structurally similar UI layouts rather than visually similar screenshots. Please refer to the supplementary material for more retrieval results.

4.4 Ablation studies

We conduct detail ablation studies on the variants of our proposed model in different stages of the framework; embedding dimensionality, the decoder model, the graph representation, and the impact of training supervision. Table 2 and Fig. 4 summarise the overall results. In the following, we break down these factors and outline the key observations.

Dimensionality and Architecture: From Fig. 4, we observe a 1024-D or higher embedding is necessary for sufficient representation of layout, with gains of 1-3% on top-5 and top-10 (for both MIoU and MPixAcc) obtained at 1024D or 2048-D over other lower dimensional embeddings. Extensive experiments were performed over dimensionality to search for best performing configuration and these are tabulated within the supplementary material.

Next, we analyse the impact of replacing strided convolution with up-sampling in the decoder (CNN) stage of the network. Across all configurations, strided convolution delivers improved results as the up-sampling operations often lead to information loss. For example, for directed graph at 2048D dimension, there is an improvements of 1.2% top-10 MIoU and 1.7% top-10 MPixAcc while using the strided convolution. Further experiments supporting this choice are tabulated in the supplementary material.

Graph representation: We contrast the performance of directed and undirected variants of the UI layout graph encoding (subsec. 3.1); note that relational feature between a pair of components $\mathbf{r}_{ij} : c_i \mapsto c_j$ is non-commutative. From Table 2, it is observed that there is a performance gain of $\sim$1-2% on both MIoU and MPixAcc for this best performing embedding size, however for lower dimensionalities which perform more poorly this gain is not present. We conclude that a directed graph supports a best performing configuration.

Metric Learning We compare the performance of the proposed framework with/without triplet training to study the advantage of weakly supervised metric learning. As seen from Table 2, there is consistent improvements (values in parenthesis) for all the variants of the proposed method while training with a dual loss comprised of auxiliary triplet loss as in Eq.(1). The improvement is easily seen in the Fig. 4. We observe clear improvements using triplet training; top-10 MIoU and MPixAcc are improved by 3.2% and 3% respectively to obtain the best performing values.

GCN Readout: We compare three strategies for pooling the GCN features to form the latent representation at the bottleneck of our GCN-CNN encoder/decoder architecture (Table 3). We evaluate our proposed self-attention mechanism for learnable pooling (Attend), with two procedural approaches: Average and Inverse presented in subsec 3.2. Performance is compared for the GCN-CNN backbone with and without triplet supervision, and for low (512) as well as high (2048) dimensional embeddings. Whilst the Average/Inverse strategies perform similarly, the learnable pooling via self-attention (Attend) yields $\sim$ 2% performance gain on both MIoU and MPixAcc metrics.

Containment feature ψ: Table 4 evaluates the benefit of explicitly encoding the containment feature (ψ) within $\mathbf{r}_{ij}$. We report performances for pres-

Table 3. Comparing of different types of readout, pooling GCN features via mean-pooling (Average), inverse area weighted mean-pooling (Inverse) and learned pooling via Self-Attention, over RICO.

Method	Dim.	GCN ReadOut	MIoU (%)			MPixAcc (%)		
top-k		Layer	1	5	10	1	5	10
GCN-CNN	512	Attend	59.4	49.3	47.1	68.3	58.9	56.5
GCN-CNN	512	Average	57.1	50.2	47.4	64.5	58.9	56.5
GCN-CNN	512	Inverse	59.3	50.4	48.3	67.3	59.3	57.7
GCN-CNN	2048	Attend	60.0	51.6	48.3	68.1	61.4	58.0
GCN-CNN	2048	Average	57.4	49.9	48.0	63.6	58.0	56.9
GCN-CNN	2048	Inverse	58.6	50.5	48.3	66.4	59.7	57.7
GCN-CNN-TRI	512	Attend	**63.2**	53.3	50.5	**71.0**	62.7	60.0
GCN-CNN-TRI	512	Average	59.3	53.8	51.3	68.1	62.5	60.3
GCN-CNN-TRI	512	Inverse	60.2	53.2	51.0	68.7	61.6	59.9
GCN-CNN-TRI	2048	Attend	61.7	**54.1**	**51.3**	70.1	**64.0**	**61.0**
GCN-CNN-TRI	2048	Average	60.0	53.7	51.1	69.8	62.3	60.2
GCN-CNN-TRI	2048	Inverse	60.1	53.6	51.1	68.0	62.2	60.1

Table 4. Evaluating performance of undirected (-Undir) vs. proposed directed graph connectivity, and the inclusion (or not) of component containment feature ψ. Unsupervised (GCN-CNN) and triplet supervised (-TRI).

Method	dim	contFeat	MIoU (%)			MPixAcc (%)		
top-k		(ψ)	1	5	10	1	5	10
			Average			Average		
GCN-CNN-UnDir	2048		56.0	49.5	47.1	63.4	58.7	56.9
GCN-CNN-UnDir	2048	✓	58.9	50.9	48.1	66.3	59.7	57.6
GCN-CNN	2048		61.9	51.0	47.7	70.5	60.6	57.2
GCN-CNN	2048	✓	60.0	51.6	48.3	68.1	61.4	58.0
GCN-CNN-TRI-Undir	2048		59.0	52.2	50.1	65.6	61.3	59.8
GCN-CNN-TRI-Undir	2048	✓	61.6	53.4	51.0	70.2	62.5	60.6
GCN-CNN-TRI	2048		**62.4**	53.4	51.1	**70.5**	63.3	60.8
GCN-CNN-TRI	2048	✓	61.7	**54.1**	**51.3**	70.1	**64.0**	**61.0**

ence/absence of the containment feature, with setting 2048D embedding for the unsupervised GCN-CNN as well as weakly supervised GCN-CNN-TRI. The performance gain using ψ is equally pronounced with and without supervision at around 1-2% for top-5 and top-10 scores, but lower for top-1. This indicates benefit in fine-grain discrimination of UI layouts.

4.5 Cumulative ablation study:

Our best configuration given the variants evaluated in subsec. 4.4 is a directed graph with containment feature (ψ) encoded via a GCN-CNN with self-attention and strided up-convolution, trained via metric learning to yield a 2048-D bottleneck. We perform a further ablation study (Table 5) demonstrating the cumu-

Table 5. Cumulative ablation study over RICO, exploring the benefit of (C)ontainment features and (DI)rected graph connectivity in the representation, and GCN feature pooling (via Self-Attention (SA) vs. Inverse mean-pooling) and (TRI)plet supervision vs. unsupervised training.

Ablation	Dim.	MIoU (%)			MPixAcc (%)		
top-k		1	5	10	1	5	10
GCNCNN	2048	57.3	49.9	47.4	64.5	58.5	56.6
GCNCNN+C	2048	58.7	50.1	48.0	66.3	58.9	57.0
GCNCNN+C+DI	2048	58.6	50.5	48.3	66.4	59.7	57.7
GCNCNN+C+DI+SA	2048	60.0	51.6	48.3	68.1	61.4	58.0
GCNCNN+C+DI+SA+TRI	2048	**61.7**	**54.1**	**51.3**	**70.1**	**64.0**	**61.0**

lative contribution to overall performance for each of these design components in the representation and the GCN-CNN architecture. Adding the containment feature (+C) to the set of relative geometry features and the directed connections in the graph (+DI) contribute around +1% accuracy. Pooling via self-attention (+SA) adds a further 1%, and the triplet supervision adds around 3% further.

4.6 Searching auto-parsed layouts

We evaluate the transferability of our RICO-trained model to GoogleUI; UI layouts automatically parsed from UX designs on the web. Google Image search retrieved 21K images with keywords 'mobile User Interface Design' and 'mobile UX Design'. AMT was used to segment images into individual layouts and discard false positives yielding 18.5K individual UI layouts for the search corpus. A query set of 50 layouts were randomly held out.

We train a Faster-RCNN detector [26] on the RICO classes $\mathcal{S}$ using RICO training data ($\mathcal{T}$) and annotate all 18.5K GoogleUI images using a threshold of probability > 0.5 to identify bounding boxes and labels for the UI components present. GoogleUI contains very few near-duplicates and has noisier annotation due to automation. Using our best performing GCN-CNN-TRI configuration we build a search index and evaluate performance in Table 6, which exceeds both the AE [7] and CAE [20] baselines by 15-20% and 10-15% respectively across the top $k = [1, 10, 15]$ results for both the MIoU and MPixAcc metrics. Representative GoogleUI search results are given in Fig. 5. Compared to RICO, the performances of all the methods likely due to inaccuracy in annotation. However, it is interesting to note that the performance improvements of the proposed method over existing methods [7][20] have been significantly increased (Table 1 vs. Table 6) indicating that graph-based layout representations may be more robust to noisy data encountered in the wild.

5 Conclusion

We proposed a novel search embedding to measure similarity of user interface layouts. Our representation is learned using a hybrid encoder-decoder backbone

Table 6. Performance comparison on automatically annotated UX designs from the web (GoogleUI). Comparing baselines to the proposed GCN-CNN-TRI framework on GoogleUI. Quantified via MIoU and MPixAcc.

Method	MIoU (%)			MPixAcc (%)		
k	1	5	10	1	5	10
AE [7]	30.3	30.7	31.1	36.2	36.9	36.7
CAE [20]	41.2	40.3	39.3	44.8	43.7	42.8
GCN-CNN-TRI (Ours)	**51.6**	**46.5**	**45.1**	**57.6**	**51.1**	**49.4**

Fig. 5. GoogleUI auto-annotated UX designs searched by our RICO-trained model. Left: Representative top-5 search results for 3 queries. Right: Sample UI layouts.

comprising a graph convolutional network (GCN) and convolutional decoder (CNN). Our directed graph representation encodes the relative positioning and types of user interface control, including component nesting. The graph is encoded via GCN with self-attention to pool learned GCN features to a vector embedding suitable for similarity search. In order to encourage metric properties in the embedding, a siamese network is formed from the GCN-CNN backbone and training supervised with dual reconstruction and triplet loss. We showed that both unsupervised, and supervised training of the latter, yield performance gain over raster autoencoder networks previously used to search UI layout. We demonstrate the benefits of our approach over both the public dataset of crowd-mined annotated mobile UX designs (RICO [7]) and a new automatically annotated corpus of UX designs from the web (GoogleUI).

Future work could incorporate appearance properties of the user interface components (for example the actual text or pixels in a label or image control) as additional features on the nodes. Currently our work has focused upon matching only the structural similarity of the UI layout. It would be interesting to exploit the learned representation for alternative tasks such as a structural prior on detecting UI components within screenshots or even as an user-assistive tool for partial design completion, for example where a graph decoder added to the architecture to make the model generative.

References

1. Ashual, O., Wolf, L.: Specifying object attributes and relations in interactive scene generation. In: Proc. ICCV (2019)
2. Beltramelli, T.: pix2code: Generating code from a graphical user interface screenshot. arXiV 1705.07962v2 (2017)
3. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)
4. Bui, T., Ribeiro, L., Ponti, M., Collomosse, J.: Compact descriptors for sketch-based image retrieval using a triplet loss convolutional neural network. Computer Vision and Image Understanding (CVIU) (2017)
5. Bylinskii, Z., Kim, N., O'Donovan, P., Alsheikh, S., Madan, S., Pfister, H., Durand, F., Russell, B., Hertzmann, A.: Learning visual importance for graphic designs and data visualizations. In: Proc. ACM UIST (2017)
6. Chen, J., Ma, T., Xiao, C.: FastGCN: fast learning with graph convolutional networks via importance sampling. In: Proc. Intl. Conf. Learning Representations (ICLR) (2018)
7. Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: Proceedings of the 30th Annual Symposium on User Interface Software and Technology. UIST '17 (2017)
8. Geigel, J., Loui, A.: Automatic page layout using genetic algorithms for electronic albuming. In: Proc. Electronic Imaging (2001)
9. Goldenbert, E.: Automatic layout of variable-content print data. Master's thesis, School of Cognitive & Computing Sciences, University of Sussex, UK (2000)
10. Gordo, A., Almazán, J., Revaud, J., Larlus, D.: Deep image retrieval: Learning global representations for image search. In: Proc. ECCV. pp. 241–257 (2016)
11. Gu, J., Joty, S., Cai, J., Zhao, H., Yang, X., Wang, G.: Unpaired image captioning via scene graph alignments. In: Proc. ICCV (2019)
12. Guo, L., Liu, J., Tang, J., Li, J., Luo, W., Lu, H.: Aligning linguistic words and visual semantic units for image captioning. In: ACM Multimedia (2019)
13. Guo, M., Chou, E., Huang, D., Song, S., Yeung, S., Fei-Fei, L.: Neural graph matching networks for few shot 3D action recognition. In: Proc. ECCV (2018)
14. Harrington, S., Naveda, J., Jones, R., Roetling, P., Thakkar, N.: Aesthetic measures for automated document layout. In: Proc. ACM Document Eng. (2004)
15. Huang, C., Loy, C.C., Tang, X.: Local similarity-aware deep feature embedding. In: Advances in neural information processing systems (2016)
16. Hurst, N., Li, W., Marriott, K.: Review of automatic document formatting. In: Proc. ACM Document Eng. (2009)
17. Khan, N., Chaudhuri, U., Banerjee, B., Chaudhuri, S.: Graph convolutional network for multilabel remote sensing scene recognition. J. Neurocomputing **357**, 36–46 (2019)
18. Kuen, J., Wang, Z., Wang, G.: Recurrent attentional networks for saliency detection. In: Proc. CVPR (2016)
19. Li, J., Yang, J., Hertzmann, A., Zhang, J., Xu, T.: LayoutGAN: Generating graphic layouts with wireframe discriminators. In: Proc. Intl. Conf. Learning Representations (ICLR) (2019)
20. Liu, T.F., Craft, M., Situ, J., Yumer, E., Mech, R., Kumar, R.: Learning design semantics for mobile apps. In: The 31st Annual ACM Symposium on User Interface Software and Technology. pp. 569–579. UIST '18,

ACM, New York, NY, USA (2018). https://doi.org/10.1145/3242587.3242650, http://doi.acm.org/10.1145/3242587.3242650
21. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proc. CVPR (2015)
22. O'Donovan, P., Agarwala, A., Hertzmann, A.: Learning layouts for single-page graphic designs. IEEE Transactions on Visualization and Computer Graphics (2014)
23. O'Donovan, P., Agarwala, A., Hertzmann, A.: Designscape: Design with interactive layout suggestions. In: Proc. ACM Conf. Human Factors in Comp. Sys. pp. 1221–1224 (2015)
24. Oh Song, H., Xiang, Y., Jegelka, S., Savarese, S.: Deep metric learning via lifted structured feature embedding. In: Proc. CVPR (2016)
25. Radenović, F., Tolias, G., Chum, O.: CNN image retrieval learns from BoW: Unsupervised fine-tuning with hard examples. In: Proc. ECCV. pp. 3–20 (2016)
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Proc. NIPS (2015)
27. Sangkloy, P., Burnell, N., Ham, C., Hays, J.: The sketchy database: Learning to retrieve badly drawn bunnies. In: Proc. ACM SIGGRAPH (2016)
28. Schlichtkrull, M., Kipf, T., Bloem, P., Berg, R.: Modeling relational data with graph convolutional networks. In: Proc. European Semantic Web Conf. (2018)
29. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proc. CVPR (2015)
30. Swearngin, A., Dontcheva, M., Li, W., Brandt, J., Dixon, M., Ko, A.: Rewire: Interface design assistance from examples. In: Proc. ACM CHI (2018)
31. Tripathi, S., Sridhar, S., Sundaresan, S., Tang, H.: Compact scene graphs for layout composition and patch retrieval. In: Proc. CVPR (2019)
32. Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., Wu, Y.: Learning fine-grained image similarity with deep ranking. In: Proc. CVPR. pp. 1386–1393 (2014)
33. Wang, R., Yan, J., Yang, X.: Learning combinatorial embedding networks for deep graph matching. In: Proc. ICCV (2019)
34. X. Pang, Y. Cao, R.L., Chan, A.: Directing user attention via visual flow on web designs. In: Proc. ACM SIGGRAPH (2016)
35. Yang, X., Yumer, E., Asente, P., Kraley, M., Kifer, D., Giles, C.: Learning to extract semantic structure from documents using multimodal fully convolutional neural networks. In: Proc. CVPR. pp. 5315–5324 (2017)
36. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. arXiV 1812.04202v2 (2019)