VividGraph: Learning to Extract and Redesign Network Graphs from Visualization Images

Sicheng Song, Chenhui Li, Yujing Sun, and Changbo Wang

Abstract—Network graphs are common visualization charts. They often appear in the form of bitmaps in papers, web pages, magazine prints, and designer sketches. People often want to modify graphs because of their poor design, but it is difficult to obtain their underlying data. In this paper, we present VividGraph, a pipeline for automatically extracting and redesigning graphs from static images. We propose using convolutional neural networks to solve the problem of graph data extraction. Our method is robust to hand-drawn graphs, blurred graph images, and large graph images. We also present a graph classification module to make it effective for directed graphs. We propose two evaluation methods to demonstrate the effectiveness of our approach. It can be used to quickly transform designer sketches, extract underlying data from existing graphs, and interactively redesign poorly designed graphs.

Index Terms—Information visualization, Network graph, Data extraction, Chart recognition, Semantic segmentation, Redesign

1 Introduction

Network graphs (hereinafter called graphs) are popular and primary forms for information visualization that can clearly visualize various types of graph data [1, 2, 3]. The results of graph visualization usually appear in the form of static images in websites, papers, magazines, and other printed matter. We may need to obtain the original graph data in many cases, such as updating the graph data, recoloring or relaying out the graph. However, due to the lack of original code and design files, it is difficult to obtain original network data.

There are some methods for solving this problem, such as information steganography [4], pattern recognition, and statistical analysis. Information steganography needs to write the original data into the image in advance, so it is not effective for a large number of existing images. In addition, the information steganography steps are complicated, and statistical analysis performance is not satisfactory. Therefore, the common method is pattern recognition [5, 6, 7, 8]. These data extraction methods focus on simple charts, such as bar charts, line charts, radar charts, and heat maps. However, none of these methods can solve the graph data extraction task. Extracting the underlying graph data is challenging. First, it is difficult to label the graph datasets, and there is no public corpus. Second, the dimension of graph data is higher than that of bar charts or pie charts, which makes data regression difficult. Third, the graph edges are difficult to extract.

We introduce **VividGraph**, a pipeline for automatically extracting and redesigning graphs from static images. VividGraph integrates the classification of directed graphs and undirected graphs, node extraction and links, and the algorithm to calculate topological relations and interactive chart redesign. Inspired by Haehn et al. [9], we generate directed and undirected graph datasets with pixel-level labels to train the deep learning module in our pipeline. We classify a graph into a directed graph or an undirected graph through a classification neural network. Then, we obtain the node and link pixels through the semantic segmentation network. Finally, we reconstruct the topological relation of the nodes through calculation. Following the data assumptions of existing chart extraction work [8, 10], we make precise definitions on the scope of our work: VividGraph assumes that the graph has no text, and the nodes are circular. The nodes do not overlap, and the edges are straight. In our user study, 91.23% of the participants agree with the scope of our work and believe that the graph in this paper is common in the real world. Our training data overcome the difficult task of labeling real data and enable the model to have a good generalization ability. The easy morphological method [8] cannot handle the inconsistent node size. Small nodes will be eroded. Our semantic segmentation network can robustly detect many types of graphs.

We demonstrate our recognition accuracy from structural similarity and image similarity. Our method is robust enough to solve the recognition of a variety of graphs, such as directed graphs, undirected graphs, hand-drawn graphs, printed graphs, and graphs from the D3 library [11] and E-charts [12] gallery. The extracted data can be used to help designers quickly transform their ideas into interactive graphs. They can also easily redesign and modify graphs with poor design or outdated data.

Our work makes the following contributions:

(1) We first propose using convolutional neural net-

• *Sicheng Song, Chenhui Li, Yujing Sun, and Changbo Wang are with the School of Computer Science and Technology, East China Normal University, Shanghai, China. E-mail: chli@cs.ecnu.edu.cn, cbwang@cs.ecnu.edu.cn.*

Manuscript received xx xx, 2020; revised xx xx, 2020.

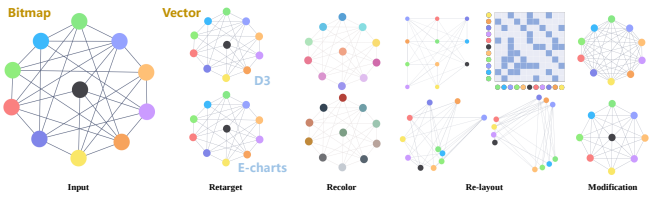


Fig. 1. VividGraph can be used in many practical applications. The input is a bitmap. Through our semantic segmentation and connection algorithm, we can obtain its underlying data. Using the extracted data, we can reconstruct the vector of the graph and redesign the chart, such as recoloring, re-layout, and data modification.

works to solve the problem of graph data extraction.
(2) We propose a pipeline with semantic segmentation to accurately identify the characteristics of graphs. The method is robust to hand-drawn graphs, blurred images, and large images.
(3) We propose two methods to evaluate the effectiveness of our methods through structure similarity and image similarity.

2 Related Work

Our work is related to three technologies: chart extraction, graph perception with CNNs (Convolutional Neural Networks), and chart interaction.

2.1 Chart Extraction

Some inverse visualization studies extract data from charts. Harper et al. [13] proposed a method for extracting underlying data from a visualization chart built with the D3 library [11]. This method depends on web page code, such as HTML and SVG. The extracted data can be used for chart redesign or color mapping redefinition. WebPlotDigitizer [14] is another graph data extraction tool based on web pages. It can extract four types of graph data, including bar charts, line charts, pole charts and ternary charts. However, the accuracy of this tool is not high, and users generally add information manually to improve accuracy. Another tool called Ycasd [15] requires the user to provide the position of all points on the line to extract the line chart data.

Static bitmaps are encountered more often. There are some studies based on image processing and machine learning to solve the problem of data extraction in bitmap charts. ReVision [6] is a data extraction framework for bitmap charts, which automatically divides charts into ten categories and focuses on data extraction for pie charts and bar charts. FigureSeer [7] focuses on extracting data from line charts. Poco et al. [8, 16] proposed a data extraction method with legends and extend the research to heat maps. They focused on the role of the legend text in the charts and added an OCR module to solve the data extraction problem for charts with legends. Zhou et al. [17] proposed a network with an attention mechanism to detect bar charts, in which deep learning techniques have been well applied.

Some researchers have focused on the method of semiautomatically extracting bitmap chart data. Jung et al. [5] introduced ChartSense, a system to increase the data extraction accuracy by manually adding information. DataThief [18] is another semiautomatic tool for extracting data from line charts. The users need to provide information such as the coordinates of the start point and endpoint of the line and the positions of the horizontal and vertical axes. iVoLVER [19] integrates data extraction of bitmap images and SVG objects on the web, providing a semiautomatic data extraction framework. These semiautomatic methods rely on a large quantity of user interaction data, such as specifying the data type (e.g., color and shape) and providing the dividing line location. While improving data extraction accuracy, it also reduces efficiency and requires considerable manual intervention. However, to the best of our knowledge, these frameworks do not support data extraction and redesign of graph bitmaps.

2.2 Graph Perception with CNNs

Haehn et al. [9] reproduced Cleveland and McGill's [20] graphic perception evaluation experiment with CNNs. They compared the recognition capabilities of four networks, MLP, LeNet [21], VGG [22], and Xception [23], on nine basic perception tasks. They presented that the graphic perception ability of VGG19 is the best among these networks. They proposed that graphs are advanced graphical coding, so the task of extracting data from graphs is challenging. Haleem et al. [24] evaluated the readability of force-directed graph layouts with CNNs. Giovannangeli et al. [25] continued this experiment and used CNNs to evaluate the image perception ability of graphs. However, their evaluation task indicators were only the number of edges, nodes and maximum degree of the graph, not the topological relations, the most critical data in the graph.

These pattern recognition methods all regress the graph numerically. The graphs are too complex to make these methods feasible. Therefore, we thought of the semantic segmentation method, a method commonly used in the field of computer vision for natural images. The fully convolutional network (FCN) [26] was the earliest method to obtain segmentation results equal to the input image size through a series of convolutional layers and deconvolution. SegNet [27] is similar to FCN. While deepening the network, it uses pooling indices to save the contour image information. PSPNet [28] uses a pyramid pooling module to simultaneously feed the feature map through four parallel pooling layers. It then obtains and upsamples four outputs of different sizes. These semantic segmentation networks are applied to natural image data, such as VOC datasets. Deep feature extraction networks cause the target to lose small features, leaving basically correct segmentation results.

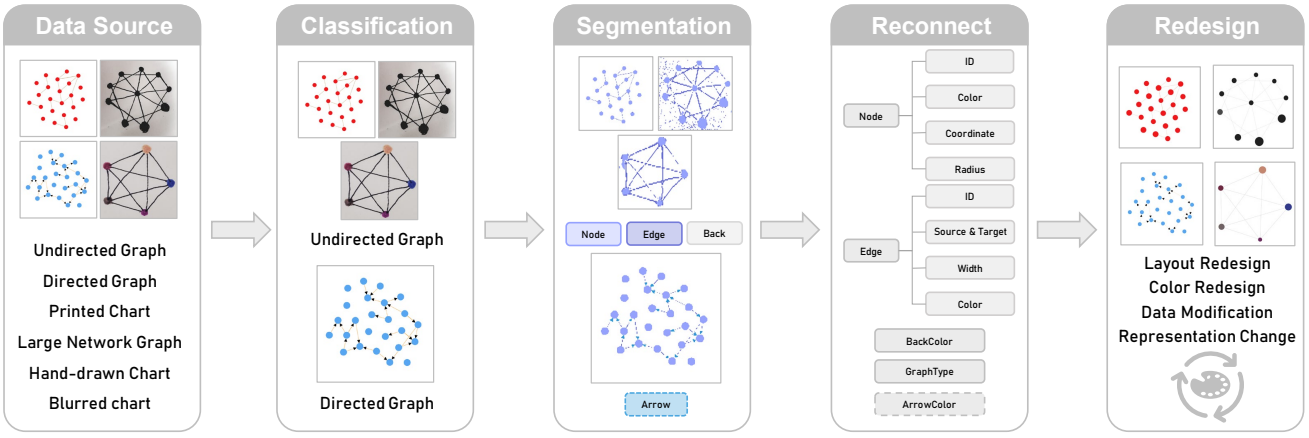


Fig. 2. The VividGraph pipeline includes five steps: (1) input a graph image, (2) classification of directed and undirected graphs, (3) semantic segmentation network, (4) algorithm to reconnect the nodes, and (5) interactive chart redesign.

However, for graph images, these small features, such as edges, are important. The network parameters need to be simplified compared to natural images.

2.3 Chart Interaction

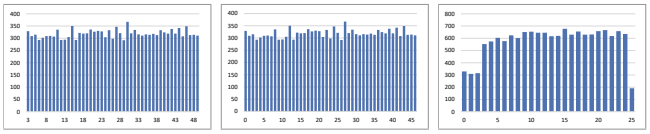
Chart redesign can maximize the value of the original data and help readers understand these data quickly and accurately. [29] Good chart visualization is made up of a clear-sighted layout, easily distinguishable color scheme and interactive application scenarios. First, the layout gives readers the first impression of a chart. Takayuki et al. [30] combined a force-directed layout with a hybrid space-filling method to simultaneously represent both connectivity and categories of multiple-category graphs. Arc diagrams [31] were proposed to display complex patterns of string data with overlapping parts. Ka-ping et al. [32] added animation techniques to radial layouts so that users can interactively explore the dynamic evolution of topological relations. Second, the similarity and comparison of color schemes help readers understand directly. Lujin Wang et al. [33] used a knowledge-based system to learn color design rules and apply them to illustrative visualization. Jorge et al. [8] extracted color encodings from a heatmap and recolored it with different color schemes to make the heatmap more comprehensible. Third, deeper information can be acquired by interactive operations. Thinkbase and Thinkpedia [34] are used to excavate semantic graphs with interactive operations of large knowledge repositories so that web content can be explored more easily. NR [35] provided an interactive database used for visual interactive analytics based on the web. Lai et al. [36] combined the chart extraction technology of bar charts and pie charts with natural language processing technology and proposed a visual interactive automatic annotation method. Kim et al. [37] proposed a pipeline that can answer questions about the chart.

3 Methods

The extraction of graphs faces three major problems: it is difficult to label the data set, the topological relation is large. If a graph has n nodes, there will be n^2 topological relations. The traditional method [25] has difficulty extracting edges. Therefore, we establish a graph data set with automatically generated pixel labels and propose VividGraph to automatically extract the data of graphs. VividGraph is a framework composed of four modules: (1) classification of directed and undirected graphs, (2) semantic segmentation network, (3) algorithm of node reconnection, and (4) interactive chart redesign. First, we classify the graph into a directed graph and an undirected graph so that the second step uses different parameters for data extraction. Second, VividGraph uses a semantic segmentation network to locate the node and edge pixels. Third, we design an algorithm to reconnect these nodes, which can calculate topological relations. Fourth, VividGraph uses the extracted data to redesign graphs according to user needs.

3.1 Training Dataset Generation

Considering that our data extraction algorithm uses a semantic segmentation neural network, we need graph datasets with pixel-level labels. We classify pixel labels into three categories, including background, nodes, and edges. When we obtain the category of each pixel, we can calculate all network attributes including relation, node size, edge width (thickness), color, etc. However, it is difficult to label the graphs generated by common visualization frameworks with pixels. Inspired by some studies [9, 10, 38] using image synthesis as datasets,



(a) Node (b) Edge (c) Maxdegree

Fig. 3. The data distribution of our training set. The horizontal axis is the number of nodes (a), the number of edges (b) and the maximum degree (c). The vertical axis is the number of images.

we provide a graph data generator for inverse visualization study of graphs, which can automatically mark the category of each pixel. This generator can control the number, color and size of nodes, the width (thickness), number and color of edges, the color and size of the background, and the type of directed graph or undirected graph. The pixels in the graph are detected as background, node, edge, and arrow (only for the directed graph).

The generator was made using Python tools and the Skimage module [39]. The image size is 320×320. The number of nodes is random between 0 and 49. The node size is random between 6 and 15 to ensure that the nodes do not overlap. To make our model robust to various layouts (e.g., force-oriented or circular), we randomly select node positions so that the model can learn various layouts. We add conditions in the process of generating nodes. If the generated node overlaps with other nodes, it will be re-randomized until it does not overlap with other nodes. The edge width (thickness) is random between 1 and 6. The colors of the background, nodes and edges are all random between (0,0,0) and (255,255,255). The random attributes can vary in a single graph.

We first generate nodes on the image and then randomly select two nodes to connect to generate edges. The graphs generated in this way can cover different graph families, but it also has some limitations. Giovannangeli et al. [25] proposed a data generation limitation. Randomly selecting the number of nodes causes the graph density distribution and the number of edges in the dataset to follow a power-law distribution. To avoid overfitting, we increase the control of the maximum degree and the number of edges in the process of random generation. As shown in Figure 3, the number of edges and graph density in our dataset are uniformly distributed.

We generate 15,000 undirected graph images as the training set and 5,000 undirected graph images as the validation set. Then, we add arrows to all images to generate a directed graph image. The width of the arrow is randomly from 1 to 5, and the length of the arrow is randomly from 1 to 5. We obtain 15,000 directed graph images as the training set and 5,000 directed graph images as the validation set. We generate a total of 40,000 visualization images, of which 30,000 are used as the training set and 10,000 as the validation set.

3.2 Graph Classification

Our pipeline can also solve the directed graph data extraction problem. To obtain the extraction results more accurately, we first classify the input images. We use a convolutional neural network (CNN) to build this module. CNN is one of the methods commonly used for image classification in the field of image processing and has been proven to have good performance [40] on the famous ImageNet dataset [41]. We tried the VGG19 model, which has been proven to have good graphical perception ability [9]. However, we found that it did not perform well on the graph classification task. We also tried the shallower network LeNet-1, and it also appeared to underfit. We compared the effects of InceptionV3, ResNet-101 [42], and Xception. The classification task results of the three networks are shown in Table 1. Finally, we choose InceptionV3 [43] among many CNN models.

The complete neural network of the system is shown in Figure 4. The classification network and the semantic segmentation network can be trained together or separately. We additionally generated 1,000 directed graphs and 1,000 undirected graphs for evaluation. The total accuracy was 99.65%. We normalized the input image to $224 \times 224 \times 3$ dimensions (sRGB space). We chose an efficient stochastic gradient descent (SGD) optimizer [44]. The initial learning rate was set to 0.001, and it decreased dynamically every 5 epochs.

TABLE 1
Performance comparison of different models in graph classification tasks

Model	Parameters	Accuracy Directed	Accuracy Undirected	Accuracy Total
VGG19	144 M	50.3%	50.1%	50.2%
Xception	22.8 M	99.3%	99.5%	98.9%
InceptionV3	23.6 M	**99.5%**	**99.8%**	**99.65%**
ResNet-101	44.7 M	88.4%	99.2%	93.8%

After graph classification, the output graph has three types of labels (for undirected graphs) or four types of

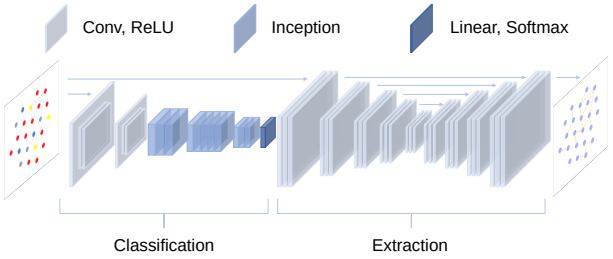


Fig. 4. VividGraph contains two neural network models: a classification network and a semantic segmentation network.

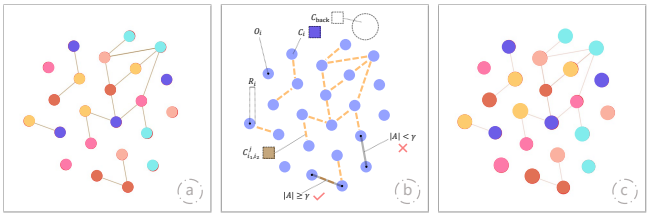


Fig. 5. Illustration of the semantic segmentation and reconnection algorithm. In (b), white pixels indicate that the pixels are detected as background, blue pixels are nodes, and orange pixels are edges. There are noise pixels detected as edges near the nodes, but it does not reduce the efficiency of our algorithm.

labels (for directed graphs).

3.3 Semantic Segmentation Network

Giovannangeli et al. [25] noted that it is difficult to recognize edges with traditional image processing methods and evaluate the possibility of using CNNs to perceive the network. However, their evaluation task indicators were only the number of edges, nodes and maximum degree of the graph. Therefore, they could not extract the topological relation in the images. Many inverse visualizations in the past were based on simple visual coding, and their attributes were often relatively simple. For example, the attribute of bar charts is the length of the bar, the attribute of scatter graphs is the coordinate value of a point, and the attribute of point cloud graphs is the number of points. As a high-level visual coding, the attributes of networks are not only the size, location, and number of its nodes but also the relations of these nodes. If using the adjacency matrix of the graph as the annotations for training, the data extraction task becomes a deep learning regression task. This method is unrealistic, and the regression data dimension is too large. Therefore, we split the data extraction task into two parts. The first part uses the semantic segmentation network to locate the nodes and edges, and the second part reconnects the nodes through our algorithm.

In traditional convolutional neural networks, an input image often has only one output label. However, in a semantic segmentation network, each pixel has an output label. In this task, we use the elements of the graph as pixel-level labels (background, node, edge). We choose U-Net [45], a semantic segmentation convolutional network applied to medical images, as our semantic segmentation network model. We normalize the image to a dimension of $320 \times 320 \times 3$ (sRGB space). We choose VGG16 as the U-Net backbone network. We use the popular deep learning framework Keras [46] to quickly implement our model.

3.4 Reconnection Algorithm

When we obtain the pixel-level label output by the semantic segmentation network, we design an algorithm to reconnect nodes to calculate the topological relation of the graph. First, we extract the connected components labeled as nodes in the image. We perform an erosion operation on these connected components to remove noise, where the size of the kernel is determined by the area of the connected components. Erosion and dilation are two fundamental morphological operations of image processing [47]. Then, we use the same size kernel to perform a dilation operation on the connected components to maintain the node radius. These connected components are the nodes of the graph.

- W, H: image width and height
- $C_{x,y}$: input image color, storing 3D data (R, G, B).
- $Label_{x,y} = \begin{cases} 0, back \\ 1, node \\ 2, edge \end{cases}$:semantic segmentation results of pixels located at (x, y)
- O_i, R_i: center coordinate and radius of Node i
- $C^j_{i_1,i_2}$: color of Edge j
- CC: Connected Component
- $Area_i$: rectangular area surrounding CC i
- k: erosion or dilation operation core size
- γ: threshold of edge pixels between two nodes

Algorithm 1 Node Reconnect Algorithm

Input: $\{C_{x,y} | x \in [0, W], y \in [0, H]\}$, $\{Label_{x,y} | x \in [0, W], y \in [0, H]\}$
Output: O_i, R_i, $C^j_{i_1,i_2}$
Extract the CC of $Label_{x,y} = 1$
for all CC **do**
  $k = \frac{1}{3} \times \sqrt{Area_i}$
  Use (k, k) size kernel to perform morphological opening on CC
end for
Extract CC again
for all CC **do**
  O_i =The coordinates of the center pixel of CC
  $R_i = \frac{1}{2} \times \sqrt{Area_i}$
end for
for each Node i_1 and Node i_2,$i_1 \neq i_2$ **do**
  Draw an line connecting Node i_1 and Node i_2
  Check $A = \{(x, y) | Label_{x,y} = 2, (x, y) \in line\}$
  Perform dilation operation
  Set $\gamma \propto length_{line}$
  if $|A| > \gamma$ **then**
    Node i_1 and Node i_2 are connected
    $C^j_{i_1,i_2} = \bar{C_{x,y}}, (x, y) \in A$
  end if
end for
return O_i, R_i, $C^j_{i_1,i_2}$;

We draw a line between every two nodes to check the number of pixels detected as edges on this line. If the number of such pixels exceeds the threshold, the two nodes are considered to be connected. The size of the threshold is proportional to the length of the line. The

result of semantic segmentation and the process of the reconnection algorithm are shown in Figure 5.

With the advantages of deep learning methods, our method can detect edges with various widths. After reconnecting all nodes, we calculate the width of each edge. We calculate the average distance from the pixels detected as the edge category near an edge to the edge, and the width of this edge is twice this distance.

We also generate a set of data with edges of different widths (thicknesses) to evaluate our algorithm for extracting edge width. The dataset is generated by the D3 library, containing 100 images with the resolution of 480×480. The correct rate of edge width is 70.76%.

The graph colors are important. The color of edges $C^j_{i_1,i_2}$ is mentioned in Algorithm 1. We take the color of the center pixel of each node as the node color. The average color value of the pixels recognized as the background category is taken as the background color C_{back}.

Many real graphs do not have a good colormap, which prevents users from clearly observing the results. We also add the contrast comparison between the foreground and background colors and the background color recommendation in the color extraction to comply with W3C standards [48]. Many real-world graphs have poor designs, making readers unable to see the nodes or edges clearly. We calculate node color averages and edge color averages. We then calculate their average as the foreground color C_{fore} and give them the same importance. The number of edge pixels is generally less than the number of node pixels, but when calculating color contrast, the edge color is as important as the node color. We use C_{back} as the background color. First, their brightness is calculated and then contrast is calculated. If the contrast is lower than 7:1, the chart does not comply with the W3C standard. We choose the color with the highest contrast to replace the background color from the safe colors preset by W3C. In the sRGB space, the definitions of color brightness and contrast are shown in Equation 1. C_i is the color of node i, and $C^j_{i_1,i_2}$ is the color of the edge connecting node i_1 and node i_2. We assume that there are n nodes and m edges here. C_w is the contrast color of the foreground and background. l is the luminance of the color. l_1 represents the brighter of the foreground and background colors, and l_2 is the other.

$$\begin{cases} C_{fore} = \frac{1}{2n}\sum_{i=1}^{n} C_i + \frac{1}{2m}\sum_{j=1}^{m} C^j_{i_1,i_2} \\ l = 0.2126R + 0.7152G + 0.0722B \\ C_w = \frac{l_1+0.05}{l_2+0.05}, l_1 > l_2 \end{cases} \quad (1)$$

3.5 Directed Graph

For directed graphs, their label categories increase from three to four (background, node, edge and arrow). The arrow category is used to determine the direction of the edge.

We take the coordinates of the midpoint of the two nodes and draw a segment with each of the two nodes.

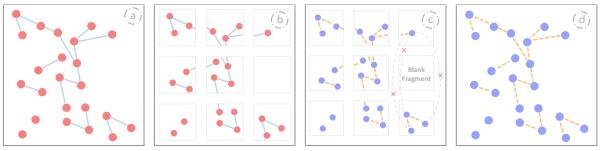


Fig. 6. Illustration of large graph data extraction. In (b), we crop the input image into several equal pieces. In (c), we extract data from each piece separately. In (d), we reconnect the nodes in a large matrix.

Then, we check the pixels detected as arrow categories on the two segments. The segment with more pixels connects the target node. This method is suitable for directed graphs where most arrows are near the target nodes.

3.6 Large Network Graph

VividGraph is also suitable for large graphs with high resolution. When the resolution of a picture is high (more than 4096×4096), we can crop the picture and extract data from each part separately. We put each part of the extracted data into a large matrix and then use Algorithm 1 to reconnect the nodes. To improve the algorithm efficiency, we do not calculate blank image fragments that do not contain node-type and edge-type pixels. If the line connecting two nodes passes through these blank fragments, then the two nodes are definitely not connected, and there is no need to perform the connection determination. This method can effectively solve the error caused by convolution when the semantic segmentation network has only 320×320 dimensional input for large graphs with dense nodes. The steps in this method are shown in Figure 6.

We generate 100 large graph images by D3 library. The number of nodes in each graph is around 100, and the resolution is 8000×8000. The average time consumption without the algorithm is 6.05 seconds. The average time consumption with this algorithm is 2.37 seconds. Time efficiency increased by 60.83%.

4 Application

VividGraph can be applied to different kinds of scenarios, such as converting sketches into electronic charts, obtaining original data and visualization retargeting. The specific examples generated by D3, E-charts and AntV are implemented on a PC with an Intel Core i5-10400F CPU and 32 GB of memory. We use an NVIDIA GTX1080Ti, 11 GB memory to train our model.

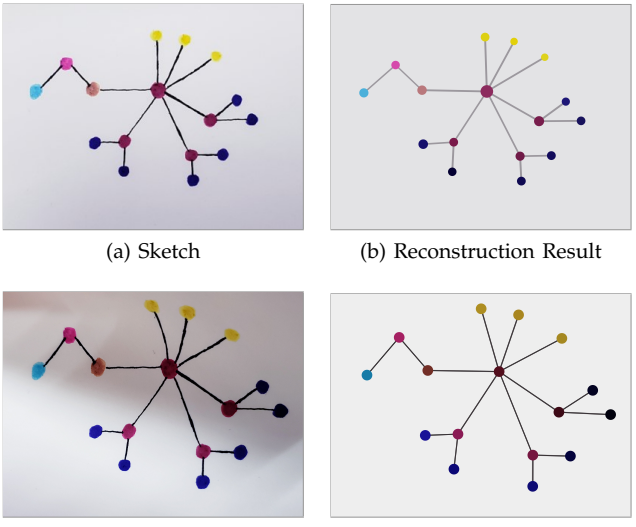
4.1 Quick Realization of Sketches

Network graphs are widely used in our daily work. Enterprise staff need to use graphs to clearly present the complex relations of different areas. Teachers who teach computer science need to vividly explain complicated graph algorithms with graphs. Designers usually add

graphs in their works to enhance aesthetic feelings. However, most of these people are not familiar with visualization tools such as E-charts, D3 and AntV, so it is difficult for them to generate electronic graphs.

To solve these problems, VividGraph provides a powerful function that converts graph sketches into electronic charts accurately and quickly. Users only need to draw a graph on a piece of paper with colored pens, take a picture of the sketch and upload this picture to VividGraph. After these steps, VividGraph extracts the sketch accurately and shows users a graph on the web that can be saved as a clear image.

A company's network administrator, who is responsible for the design, management, and maintenance of computers and networks, often needs to draw a sketch of the company's network connections. She or he can simply draw the network structure connecting the computers on paper and then use VividGraph to turn it into an interactive graph for modification. As shown in Figure 7(a), the designer uses blue nodes to represent the internet, pink nodes to represent routers, orange nodes to represent firewalls, red nodes to represent switches, yellow nodes to represent servers, and purple nodes to represent personal computers. Our method is robust enough to deal with various types of hand-drawn graphs, even if noise is accidentally introduced when taking pictures. It is common for the designer to introduce shadows when taking pictures, as shown in Figure 7(c). The shadow noise present more challenges for semantic segmentation. Our semantic segmentation model and Algorithm 1 can still obtain the correct result of relations. However, since the node color in the original image is a shaded color, our algorithm also extracts a shaded color instead of the real color of the node. Our evaluation and user study both include this type of sketches.



(a) Sketch (b) Reconstruction Result

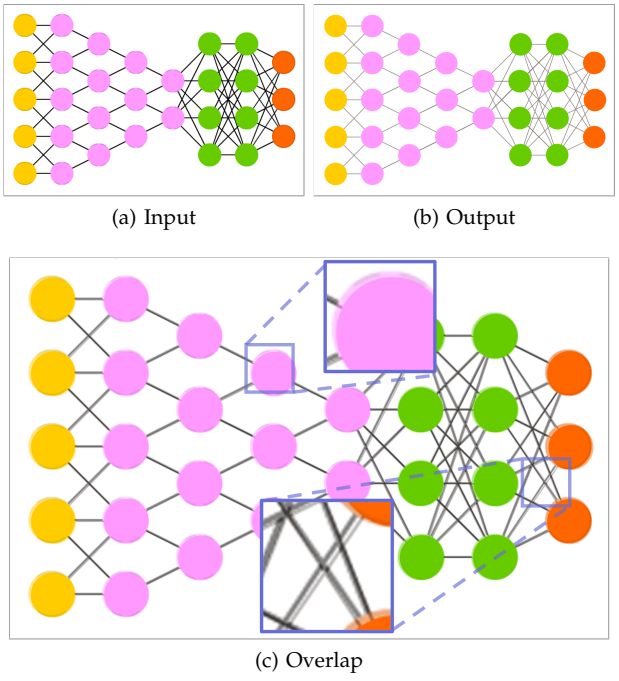
(c) Sketch with Shadow (d) Reconstruction Result

Fig. 7. The reconstruction result of computer network sketches. The reconstruction results have some bias in color, but the topological relation and position are robust.

4.2 Extraction of Underlying Data

In scientific research, the reuse of previous research results facilitate reproducibility and open science. Graphs published in books or papers always contain complex relations, such as social networks and knowledge maps. To make full use of data embedded in these existing graphs, VividGraph can identify the topological relation of the graph and generate the corresponding JSON file. Additionally, it shows a new graph on the web and allows users to operate on it.

To obtain underlying bitmap data, users take a picture or a screenshot of a graph on papers and upload the image to VividGraph. After that, on the one hand, users can download the generated JSON file directly to obtain the underlying data of the graph. On the other hand, users can obtain the vector of the graph by E-charts, D3 or AntV. Based on the vector, users can perform some operations to change the network, such as adding new nodes and links, deleting useless nodes, and changing their position.



(a) Input (b) Output

(c) Overlap

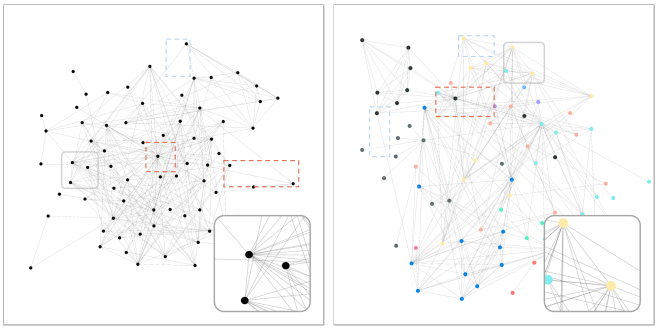
Fig. 8. The result of extracting the underlying data of the graph in the paper [49].

Neural network zoo [49] uses graphs to clearly and vividly summarize neural networks in recent years. We notice that many readers requested graph SVGs on the author's blog to help with their research, but the author only has bitmap data. We input one of the network pictures in this paper into VividGraph. The output are shown in Figure 8(b). The output image is almost the same as the input image (SSIM [50] = 0.8414), which proves that our extraction is accurate. We overlap the

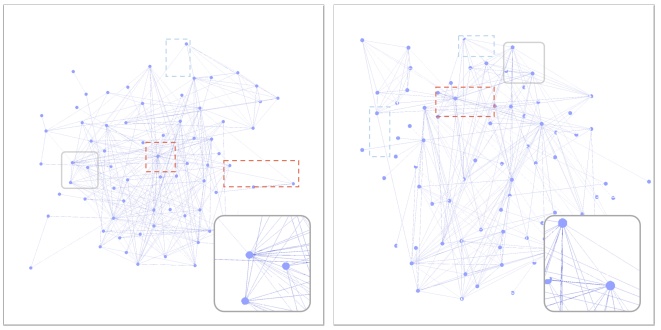
input image and output image to show the difference more clearly. As shown in Figure 8(c), there is a slight offset in the positions of the node and the edge, the color of the edge has little bias, and other results are completely correct.

Shutterstock is a global picture market open to artists and creators. We select some design pictures related to the graph and put them into VividGraph, as shown in Figure 9(a). Our system can also restore some blurred graphs. Figure 9(b) is the result of data extraction and reconstruction of a chart with a resolution of 160×160, which is an obviously blurred graph from the Google image search system. When a directed graph appears, VividGraph automatically classifies the graph, uses the parameters of the directed graph to perform semantic segmentation, and obtains accurate results, whose results are shown in Figure 9(c).

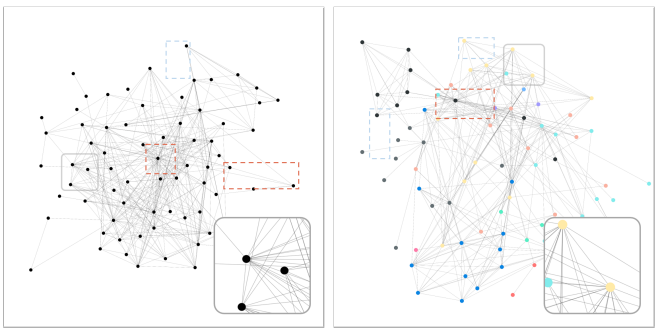
VividGraph can also extract the underlying data of large graphs. Figure 10 shows large graphs from the D3 gallery, which have a large number of nodes and dense edges. In the red box, because the edges are dense, the output has more edges than the input. In the blue box, due to insufficient pixels in the segmentation result, the output has fewer edges than the input. Despite some minor differences, the topological relations of most networks were accurately extracted. We also conduct a user study in Sec. 5, and most of the users indicate that these minor errors can be accepted in large-scale network extraction. Users gave the topological relations of the large network a score of 85.12 (out of 100). The real world corpus in Figure 14 includes large network. In addition, we provide the interaction of adding and deleting nodes and edges in the system, and users can also eliminate errors through this interactive operation.



(a) Input



(b) Segmentation Result



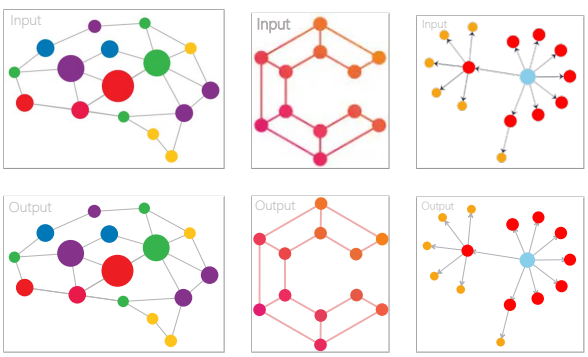
(c) Output

Fig. 10. The reconstruction results of large graphs. VividGraph can extract complex networks with dense nodes and edges. We show some details in the lower right corner of each picture. We also label some errors in the figure.

4.3 Visualization Redesign

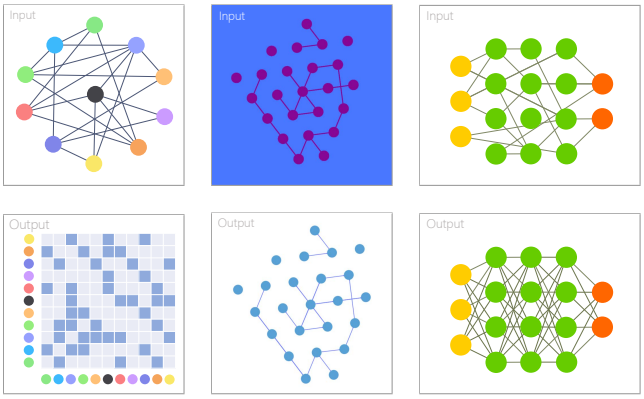
Many existing graphs have poor designs because they are not designed by professional visualization workers. VividGraph provides automatic and interactive chart redesign functions, including recoloring, re-layout and data modification. Color plays a crucial role in visualization. Some poor color schemes make people feel unpleasant and even make it impossible for people with color weakness or color blindness to obtain the chart characteristics. We show the recoloring results of low-contrast graphs in Figure 11(b). We can see that VividGraph also has good performance on graphs with similar nodes, edges and background colors.

To make it easier for users to access the graph information, the graph has many different kinds of representations, such as a tree, force-directed layout, radial layout, circle layout, and grid. The force-directed layout is one of the most basic graph layouts. The layout algorithm exerts a repulsive force between any two



(a) Shutterstock (b) Blurred (c) Directed

Fig. 9. The results of extracting the underlying data of the graphs on the internet. The first row is the input bitmap, and the second row is the output vector reconstructed from the extracted result.



(a) Re-layout (b) Recolor (c) Modification

Fig. 11. Redesign of graphs. In (a), we convert a graph into adjacency matrix respectively. In (b), we recolor a graph with poor color design. In (c), we modify a neural graph node connection relation.

nodes and an attractive force between two linked nodes. All nodes are evenly distributed across the screen to achieve an "aesthetically pleasing" presentation [51].

The adjacency matrix performs better than the network layout in terms of weight variation and connectivity tasks [52, 53]. In particular, compared to network representation, the adjacency matrix shows the relations between nodes clearer, especially when the number of links in the graph is large [54], which is called a dense graph. For example, there is a technical department that consists of employees in different positions, such as product managers, software development engineers, and test engineers. As shown in Figure 11(a), each node in the graph represents an employee, and each link represents a cooperation project between employees. The department manager needs to count the projects that every employee has participated in to assign new project tasks. The department manager has to count the links one by one with the network in the node-link graph, while he or she only needs to count nonempty cells row by row to obtain the degree of each node in the adjacency matrix.

To adapt to changeable application scenarios, VividGraph also provides some node and link operations for users. Users can add a new node with a specified radius and color and put it anywhere on the canvas or delete any existing node. Additionally, it allows users to add links between nodes with any color or delete any existing link. Through these operations, the designer or researcher can modify the bitmap of the existing graph instead of drawing the graph from scratch, which improves efficiency. In Figure 11(c), we show the modification of an existing neural graph to another neural network.

5 User Study

To improve the user experience of VividGraph, we conducted some user studies. We obtained feedback through user interviews and questionnaires after using the system in actual scenarios. We interviewed three types of people: professional designers, people who are not proficient in computers, and scientific researchers.

Procedure: Our investigation is divided into five steps: (1) user screening and informed consent, (2) system introduction, (3) questionnaire survey, (4) actual use of the system, and (5) interview. We first obtained informed consent from the participants and classified the participants by occupation. After introducing the VividGraph functions in detail, we conducted a questionnaire survey to evaluate the effectiveness of our method through user points. After that, users tried our system to complete some work tasks related to their careers, and we obtained user feedback through interviews.

Recruitment: We recruited 60 participants, including professional designers (more than 2 years in the industry), scientific researchers in the computer field, and ordinary workers who are not majors in computer science. We excluded the data of 3 participants because they did not understand the system. We analyzed the data of the remaining 57 participants (μ_{age}=28.7 years of age, 33 computer professionals, 24 noncomputer professionals) and obtained feedback through interviews.

Questionnaire: We designed 20 questions to evaluate the effectiveness of our method. We designed 14 questions for participants to score the extraction results. The results were extracted from hand-drawn graphs, images from Shutterstock, images from the D3 library, images from academic papers and large graphs. Participants scored from four aspects: color, node location, node radius, and topological relation by a rating slider. This slider was initially set to 0, and the user could move the slider to the maximum value of 100, which represents the degree of satisfaction with the reconstruction results in this aspect. The scoring result is shown in Figure 12. Our average score was 89.09, indicating that participants were generally satisfied with our extraction performance. Some graphs in bitmap format saved online were blurred due to compression or transmission. We present two sets of blurred graph results extracted and reconstructed by the system; 80.7% of participants thought the output results were clearer. The next 4 questions were the results of re-layout or recoloring graphs with poor design, allowing participants to judge whether the graphs helped them more rapidly obtain the characteristics of graphs, 92.99% of the participants thought it was easier to obtain the characteristics of the recolored graph. 59.65% of the participants felt that the relayout of the adjacency matrix obtained features visually faster. When the graph was close to the fully connected graph, this ratio increased to 73.68%. The results indicated that 91.23% of participants thought the images given in this questionnaire were common graphs in the real world, and 96.49% thought VividGraph could help them with their daily study or work. The ques-

tionnaire data have been included in the appendix for further study.

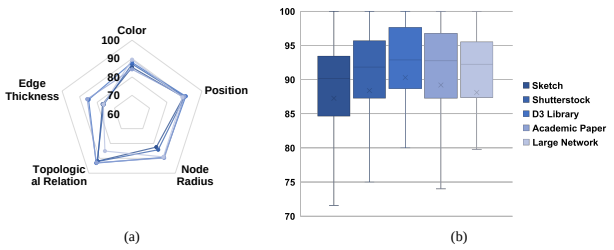


Fig. 12. The questionnaire scoring results. The maximum score is 100. In (a), Our topological relation and location extraction results are more satisfactory. The extraction of node size, edge thickness and color is basically accurate, but still needs improvement. In (b), the graphs from D3 library have the most satisfactory performance. The sketch has room for improvement due to some color bias.

Interview: After the survey, we invited participants to use our system to solve some actual problems in their work. We obtain their feedback through interviews to improve the system. We divide the feedback into three categories according to the user's occupation. Ordinary people who are not computer majors think it is helpful to convert hand-drawn graphs into images. In the interview, a middle school teacher who teaches natural sciences and a primary school teacher who teaches mathematics both stated that graphs often appear in elementary education. When they need to make courseware and examination papers, their electronic drawing ability is so weak that they spend considerable time drawing graphs. They proposed enhancing the accuracy and robustness of hand-drawn graph extraction.

Professional designers feel that the function of converting hand-drawn graphs into images is not practical. They can use drawing software proficiently, and the time consumption of hand drawing is greater than drawing directly on the computer. However, they find it helpful to convert existing graphs into vectors. They need to use materials found on the internet in their design work. The quality of these materials is uneven, many of which are bitmaps. They proposed the need for more professional automatic color schemes and more customized data modification options, such as node size, node color, edge color, and edge width. After adding some customization options to the system, we again invite these professional designers to use the system. They also proposed adding some functions to improve the human-computer interaction experience, such as color-picking pens, auxiliary design lines, and canvas changes. Embedding our system into drawing software on iPads or professional drawing software can also be considered. In the future, we will continue to explore this direction.

The last group of people is scientific researchers. They are as proficient in computer software as designers. However, their demand lies more in quoting and modifying graphs in papers. Most of the existing graphs in the paper are bitmaps or sketches. In these interviews, more than half of the studies were related to deep learning. When modifying other neural network structures(e.g. change Inception V1 to Inception V2), they can save drawing time by using VividGraph. In addition, many researchers are not specialized in visualization. Our system can also help them generate graphs with reasonable layout and color matching, making the charts pleasant and easy to obtain features. This type of user feedback was the most positive feedback among the three types of users. They also hoped that some auxiliary interactive functions can be added in the future.

6 Perceptual Evaluation

We propose two methods to prove the effectiveness of our method. The first method is NetSimile [55], which is proven to be a scalable and effective method for measuring the structural similarity between two networks. NetSimile features integrate the degree of nodes, clustering coefficient, two-hop away neighbors, ego net, etc. It generates a 35-dimensional signature vector from the average, median, and standard deviation of these features. It defines the similarity of two networks by the Canberra distance of two signature vectors. Given a graph signature vector x_i and the other graph signature vector y_i, where $i = 1, 2, 3..., 35$, the Canberra distance between x_i and y_i is defined as:

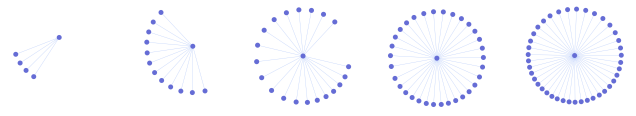
$$\sum_{i=1}^{35} \frac{\|x_i - y_i\|}{\|x_i\| + \|y_i\|} \tag{2}$$

The lower the Canberra distance is, the more similar the two networks. This method does not require a one-to-one correspondence of nodes between the two networks. Therefore, when there are missing nodes in our evaluation experiment, we can still output evaluation indicators. NetSimile is a indicator that varies with the number of nodes. To show the range of NetSimile, we traverse the NetSimile between each graph and the graph structure with the same number of nodes. We choose the maximum value from them as the maximum value of NetSimile. When NetSimile reaches this value, the two graphs are significantly different. In our datasets, this value is approximately 24, and we choose it as the maximum value of the Y-axis.

The second method is SSIM [50], which is widely used to evaluate the similarity of two images. We use the extracted data (topological relation; node coordinates, color and size; background color; edge color) to reconstruct the image of the graph. We measure the similarity of the two graphs by calculating the SSIM of the reconstructed image and the original image. Given the images x and y of two graphs, where μ_x is the average of x, σ_x^2 is the variance in x, σ_{xy} is the covariance of x and y, and c is a constant used to maintain stability,

(a) Chain ($SSIM = 0.98, NetSimile = 3.13$)

(b) Loop ($SSIM = 0.98, NetSimile = 2.53$)



(c) Egocentric ($SSIM = 0.97, NetSimile = 0.06$)

(d) Clique ($SSIM = 0.93, NetSimile = 0$)

Fig. 13. We evaluate four patterns of graphs. The results of the correct recognition are filled with blue. In the gray box, we show the missing edge.

SSIM is defined as:

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (3)$$

The higher the 1-SSIM is, the more different the two graphs. When 1-SSIM is 0, the two images are exactly the same.

In addition, to ensure the accuracy of the similarity evaluation, we turn off the contrast calculation of the foreground and background colors and the recoloring function module of the chart with poor design in the evaluation experiment.

6.1 Pattern Corpus

VividGraph is robust to different network structures. Some studies [56] related to graphs summarize four patterns of networks. There are chain type, loop type, egocentric type, and clique type. We select these four patterns that often appear in the network for evaluation experiments.

For each pattern of the network, we select ten images with a resolution of 960×960 for evaluation. Some experimental results are shown in Figure 13. The blue images are reconstructed from the extraction results. The network structures of these four modes were extracted correctly. The average SSIM was 0.97, and the average NetSimile was 1.43. High SSIM and low NetSimile show that VividGraph can accurately extract networks of various patterns. This is because our training set covers nodes of various sizes, edges of different lengths, and various distribution relations.

6.2 Real-World Corpus

We collect 100 pictures from Google, Shutterstock, E-charts Gallery, D3 Gallery and user hand-drawn sketches as our evaluation dataset. Examples of pictures from each source are given in Figure 14. The average SSIM obtained by extracting the underlying data in the bitmap by our method and reconstructing the image was 0.95. The average NetSimile was 7.67. The error mainly comes from some complex graphs and unclear pictures in the real world. However, VividGraph can still extract most of the network, especially when some hand-drawn graphs have background noise. This is not possible for traditional image processing methods.

In addition, we also find that the image extracted and reconstructed by our method is clearer and easier for obtaining chart features than the original image. To verify this point of view, we also use the evaluation model of the attention mechanism [57], an automatic model that has been proven to predict the importance of different elements in the design of data visualization. This method can give an attention score between 0 and 255. The higher the score, the more attractive the data visualization elements in the image are. The average score of the input images was 96.44. The average score of the output images was 97.99. The score increased by 1.55. The small change in score proves the accuracy of extraction, while the positive change shows that reconstructing real-world bitmaps through our method can help users better obtain chart features.

6.3 Additional Corpus

To further verify that our method is robust to graphs with different resolutions and numbers of nodes, we build four additional corpora. They include Simple I corpus from the D3 library, Simple II corpus from E-charts, complex corpus from Skimage, and directed graph corpus from D3 library. The graphs of Simple I and Simple II corpora are undirected graphs with random topological relations, random colors, and force-directed layouts. We choose three resolutions of 480×480, 640×640, and 800×800 and three types of node numbers around 10, 25, and 50. The results of the evaluation experiment are shown in Figure 15(a) and (b).

The graphs of a complex corpus do not follow any layout, so there are many layouts that have more node-edge overlap, edge-edge overlap, or nodes that are closer but still non-overlapping. Moreover, their colors are all random, their edges are thin, and there are many images where human eyes cannot distinguish between the background and the edges. The results shown in Figure 15(c) demonstrate that although the indicators decline, our method is still effective.

We test three sets of directed graphs with a resolution of 640×640 and the number of nodes around

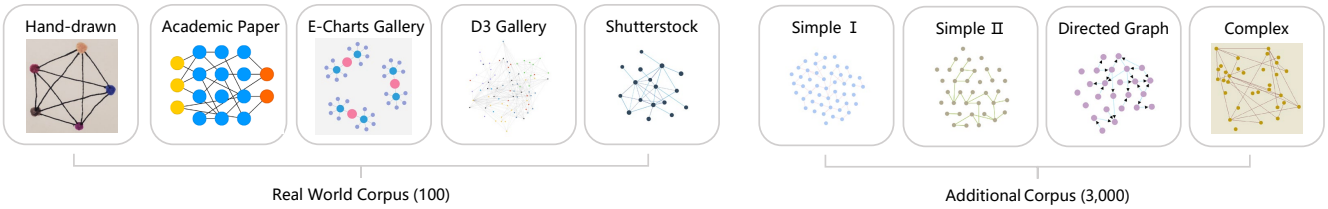


Fig. 14. Our perceptual evaluation dataset examples.

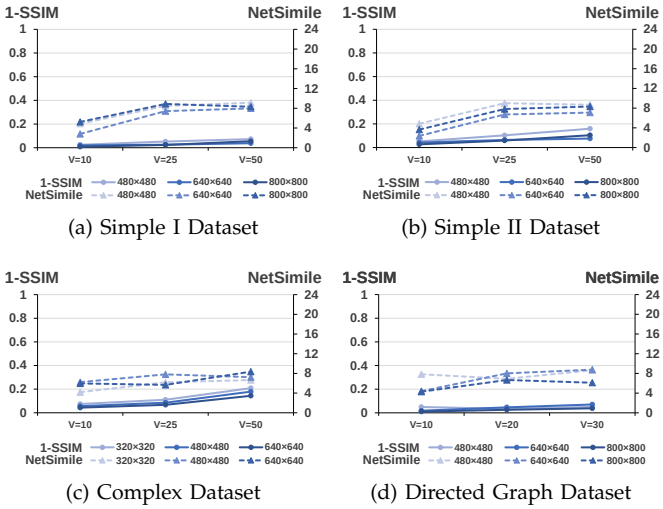


Fig. 15. Evaluation experiment results. In (a), the results demonstrate that VividGraph can deal with graphs of various image resolutions and different scale nodes. In (b), the results are similar to those of Simple I. In (c), the results demonstrate that although 1-SSIM raises, VividGraph is still effective in extreme cases. In (d), the results demonstrate that VividGraph is also suitable for data extraction of directed graphs.

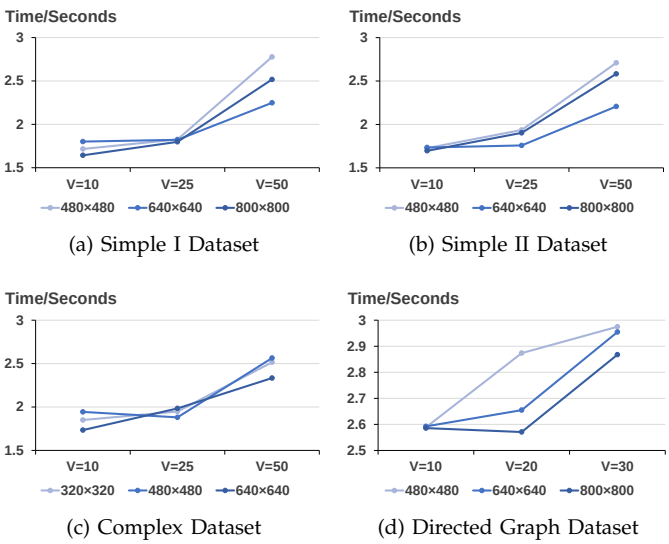


Fig. 16. Time performance comparisons of evaluation experiments.

10, 20, and 30. The results are shown in Figure 15(d). Since our pipeline has an efficient classification network and model parameters trained separately for directed graphs, our method is also effective for nondense, clear directed graphs.

6.4 Time Performance

We also evaluate the time performance of VividGraph on these datasets in Figure 16. In the figure, the X-axis represents the number of nodes, the Y-axis represents the number of seconds used, and the lines of different colors represent images of different resolutions.

The resolution of the image has little effect on the time efficiency of VividGraph. The time used will increase significantly as the number of nodes increases. This is because the time spent on VividGraph mainly comes from Algorithm 1. The time complexity of Algorithm 1 is $O(n^2)$, where n is the number of nodes. Therefore, when processing large-scale graphs, the calculation time of VividGraph will become longer. VividGraph has a good time performance for graphs with fifty nodes in general.

7 Limitation and Discussion

The current version of VividGraph also has some limitations in extracting graph data. First, our method is data-driven, so the model is based on our training set. We have tried to make our training dataset cover various graph styles, including the scale of graph, the graph density, the graph layout, the arrow size, the image resolution, etc. However, we cannot cover the space for all visualizations. When the number of nodes in the picture exceeds 50, the accuracy of the model will decrease. we cannot deal with the overlapping nodes. When the nodes overlap, more errors will occur. As shown in Figure 15, the error of complex dataset is higher than that of simple dataset. The reason is that complex dataset includes images with overlapping nodes. The error also increased when $V = 50$. We propose an extraction algorithm of large graphs to solve those high-resolution images with more than 50 nodes. In the process of cutting the picture, if the node is cut into a non-circular shape, there may be problems with semantic segmentation or errors in the size of the nodes.

As shown in Figure 10(b), the segmentation result of a node in the detail is smaller than the ground truth.

Second, the reconnection algorithm can be further improved. The segmentation results are not completely accurate. When the pixels of the edge category are identified as the background category, the edge width will be smaller than the original image, as shown in Figure 8(c). When the pixels of the background category are identified as the edge category, the color of the edges is closer to back ground, as shown in Figure 8(b). Besides, these conditions will result in the offset of the edge, as shown in Figure 8(c). For the sketches with shadow as shown in Figure 7(c,d), our algorithm cannot infer the original color of shadowed nodes. We plan to increase the accuracy of the segmentation model or optimize the heuristic rules to improve this in the future.

Third, when the three nodes are collinear, if we have no prior knowledge and cannot distinguish with our eyes, our algorithm will think that any two of these nodes are connected. As shown in Figure 11(c), the collinear nodes are considered as connected with each other.

8 Conclusion

We proposed a method to extract the underlying data of the graph image. We also proposed a pipeline called VividGraph, which combines a semantic segmentation model and a node connection algorithm. This framework is suitable for undirected graphs, directed graphs, blurred graph images, hand-drawn graphs, large graph images, and other graphs. VividGraph can be used to quickly transform designer sketches, restore blurred graph pictures, obtain the underlying data of bitmaps to generate vectors, modify graph data, redesign graphs, etc.

In the future, we plan to improve the time efficiency and accuracy of pipelines for large-scale networks by optimizing networks and algorithms. We will combine the model of this paper with OCR technology to improve our system. Cooperating with designers to improve the human-computer interaction experience of the system is also under our consideration.

Acknowledgments

We would like to acknowledge the support from NSFC under Grant No. 61802128 and 62072183.

References

[1] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, "Evaluating bag-of-visual-words representations in scene classification," in *Proceedings of the international workshop on Workshop on multimedia information retrieval*, 2007, pp. 197–206.

[2] Y. Liu, X. Lu, Y. Qin, Z. Tang, and J. Xu, "Review of chart recognition in document images," in *Visualization and Data Analysis 2013*, vol. 8654. International Society for Optics and Photonics, 2013, pp. 384–391.

[3] E. Brynjolfsson and K. McElheran, "The rapid adoption of data-driven decision-making," *American Economic Review*, vol. 106, no. 5, pp. 133–39, 2016.

[4] P. Zhang, C. Li, and C. Wang, "Viscode: Embedding information in visualization images using encoder-decoder network," *IEEE TVCG*, 2020.

[5] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo, "Chartsense: Interactive data extraction from chart images," in *Proceedings of the 2017 chi conference on human factors in computing systems*, 2017, pp. 6706–6717.

[6] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer, "Revision: Automated classification, analysis and redesign of chart images," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 393–402.

[7] N. Siegel, Z. Horvitz, R. Levin, S. Divvala, and A. Farhadi, "Figureseer: Parsing result-figures in research papers," in *European Conference on Computer Vision*. Springer, 2016, pp. 664–680.

[8] J. Poco, A. Mayhua, and J. Heer, "Extracting and retargeting color mappings from bitmap images of visualizations," *IEEE TVCG*, vol. 24, no. 1, pp. 637–646, 2017.

[9] D. Haehn, J. Tompkin, and H. Pfister, "Evaluating 'graphical perception'with cnns," *IEEE TVCG*, vol. 25, no. 1, pp. 641–650, 2018.

[10] L. Yuan, W. Zeng, S. Fu, Z. Zeng, H. Li, C.-W. Fu, and H. Qu, "Deep colormap extraction from visualizations," *IEEE TVCG*, 2021.

[11] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE TVCG*, vol. 17, no. 12, pp. 2301–2309, 2011.

[12] D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, and W. Chen, "Echarts: a declarative framework for rapid construction of web-based visualization," *Visual Informatics*, vol. 2, no. 2, pp. 136–146, 2018.

[13] J. Harper and M. Agrawala, "Deconstructing and restyling d3 visualizations," in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, 2014, pp. 253–262.

[14] A. Rohatgi, "Webplotdigitizer," 2017.

[15] A. Gross, S. Schirm, and M. Scholz, "Ycasd–a tool for capturing and scaling data from graphical representations," *BMC bioinformatics*, vol. 15, no. 1, p. 219, 2014.

[16] J. Poco and J. Heer, "Reverse-engineering visualizations: Recovering visual encodings from chart images," in *Computer Graphics Forum*, vol. 36, no. 3. Wiley Online Library, 2017, pp. 353–363.

[17] F. Zhou, Y. Zhao, W. Chen, Y. Tan, Y. Xu, Y. Chen, C. Liu, and Y. Zhao, "Reverse-engineering bar charts using neural networks," *Journal of Visualization*, pp. 491–435, 2021.

[18] A. Flower, J. W. McKenna, and G. Upreti, "Validity and reliability of graphclick and datathief iii for data extraction," *Behavior modification*, vol. 40, no. 3, pp. 396–413, 2016.

[19] G. G. Méndez, M. A. Nacenta, and S. Vandenheste, "ivolver: Interactive visual language for visualization extraction and reconstruction," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 4073–4085.

[20] W. S. Cleveland and R. McGill, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *Journal of the American statistical association*, vol. 79, no. 387, pp. 531–554, 1984.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," pp. 1–14, 2015.

[23] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE CVPR*, 2017, pp. 1251–1258.

[24] H. Haleem, Y. Wang, A. Puri, S. Wadhwa, and H. Qu, "Evaluating the readability of force directed graph layouts: A deep learning approach," *Computer Graphics and Applications, IEEE*, 2019.

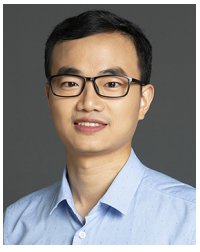
[25] L. Giovannangeli, R. Bourqui, R. Giot, and D. Auber, "Toward automatic comparison of visualization techniques: Application to graph visualization," *Visual Informatics*, 2020.

[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE CVPR*, 2015, pp. 3431–3440.

[27] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[28] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE CVPR*, 2017, pp. 2881–2890.
[29] N. Kong and M. Agrawala, "Graphical overlays: Using layered elements to aid chart reading," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 12, pp. 2631–2638, 2012.
[30] T. Itoh, C. Muelder, K.-L. Ma, and J. Sese, "A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs," in *2009 IEEE Pacific Visualization Symposium*. IEEE, 2009, pp. 121–128.
[31] M. Wattenberg, "Arc diagrams: Visualizing structure in strings," in *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.* IEEE, 2002, pp. 110–116.
[32] K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst, "Animated exploration of graphs with radial layout," in *Proc. IEEE InfoVis 2001*, 2001, pp. 43–50.
[33] L. Wang, J. Giesen, K. T. McDonnell, P. Zolliker, and K. Mueller, "Color design for illustrative visualization," *IEEE TVCG*, vol. 14, no. 6, pp. 1739–1754, 2008.
[34] C. Hirsch, J. Hosking, and J. Grundy, "Interactive visualization tools for exploring the semantic graph of large knowledge spaces," in *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009)*, vol. 443, 2009, pp. 11–16.
[35] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
[36] C. Lai, Z. Lin, R. Jiang, Y. Han, C. Liu, and X. Yuan, "Automatic annotation synchronizing with textual description for visualization," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
[37] D. H. Kim, E. Hoque, and M. Agrawala, "Answering questions about charts and generating visual explanations," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
[38] Y. Ma, A. K. Tung, W. Wang, X. Gao, Z. Pan, and W. Chen, "Scatternet: A deep subjective similarity model for visual analysis of scatterplots," *IEEE TVCG*, vol. 26, no. 3, pp. 1562–1576, 2018.
[39] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.
[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE CVPR*. Ieee, 2009, pp. 248–255.
[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE CVPR*, 2016, pp. 770–778.
[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE CVPR*, 2016, pp. 2818–2826.
[44] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.
[45] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.
[46] F. Chollet *et al.*, "Keras," https://github.com/keras-team/keras, 2015.
[47] J. Serra, "Image analysis and mathematical morphol-ogy," 1982.
[48] B. Caldwell, M. Cooper, L. G. Reid, G. Vanderheiden, W. Chisholm, J. Slatin, and J. White, "Web content accessibility guidelines (wcag) 2.0," *WWW Consortium (W3C)*, 2008.
[49] S. Leijnen and F. v. Veen, "The neural network zoo," in *Multidisciplinary Digital Publishing Institute Proceedings*, vol. 47, no. 1, 2020, p. 9.
[50] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
[51] S. E. Palmer, K. B. Schloss, and J. Sammartino, "Visual aesthetics and human preference," *Annual review of psychology*, vol. 64, pp. 77–107, 2013.
[52] B. Alper, B. Bach, N. Henry Riche, T. Isenberg, and J.-D. Fekete, "Weighted graph comparison techniques for brain connectivity analysis," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2013, pp. 483–492.
[53] M. Okoe, R. Jianu, and S. Kobourov, "Node-link or adjacency matrices: Old question, new insights," *IEEE TVCG*, vol. 25, no. 10, pp. 2940–2952, 2018.
[54] M. Ghoniem, J.-D. Fekete, and P. Castagliola, "On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis," *Information Visualization*, vol. 4, no. 2, pp. 114–135, 2005.
[55] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Netsimile: A scalable approach to size-independent network similarity," *arXiv preprint arXiv:1209.2684*, 2012.
[56] Y. Wang, G. Baciu, and C. Li, "Smooth animation of structure evolution in time-varying graphs with pattern matching," in *SIGGRAPH Asia 2017 Symposium on Visualization*, ser. SA '17, New York, NY, USA, 2017.
[57] Z. Bylinskii, N. W. Kim, P. O'Donovan, S. Alsheikh, S. Madan, H. Pfister, F. Durand, B. Russell, and A. Hertzmann, "Learning visual importance for graphic designs and data visualizations," in *Proceedings of the 30th Annual ACM symposium on user interface software and technology*, 2017, pp. 57–69.

Sicheng Song received his B.Eng. from Hangzhou Dianzi University, China, in 2019. He is working toward the Ph.D. degree with East China Normal University, Shanghai, China. His main research interests include information visualization and visual analysis.



Chenhui Li received Ph.D. from the Department of Computing at Hong Kong Polytechnic University, in 2018. He is an associate professor with the School of Computer Science and Technology at East China Normal University. He received ICCI*CC Best Paper Award (2015) and SIGGRAPH Asia Sym. Vis. Best Paper Award (2017). He has served as a local chair in VINCI2019. He works on the research of information visualization and computer graphics.

Yujing Sun received her B.Eng. from East China Normal University, in 2020. She is working toward the Master degree with East China Normal University, Shanghai, China. Her main research interests include information visualization and visual analysis.

Changbo Wang is a professor with the School of Computer Science and Technology, East China Normal University. He received his Ph.D. degree at the State Key Lab of CADCG of Zhejiang University in 2006. He was a post-doctor of the State University of New York in 2010. His research interests mainly include computer graphics, information visualization, visual Analytics, etc. He is serving as the Young AE of Frontiers of Computer Science, and PC member for several international conferences.