GlyphCreator: Towards Example-based Automatic Generation of Circular Glyphs

Lu Ying, Tan Tang, Yuzhe Luo, Lvkeshen Shen, Xiao Xie, Lingyun Yu, Yingcai Wu

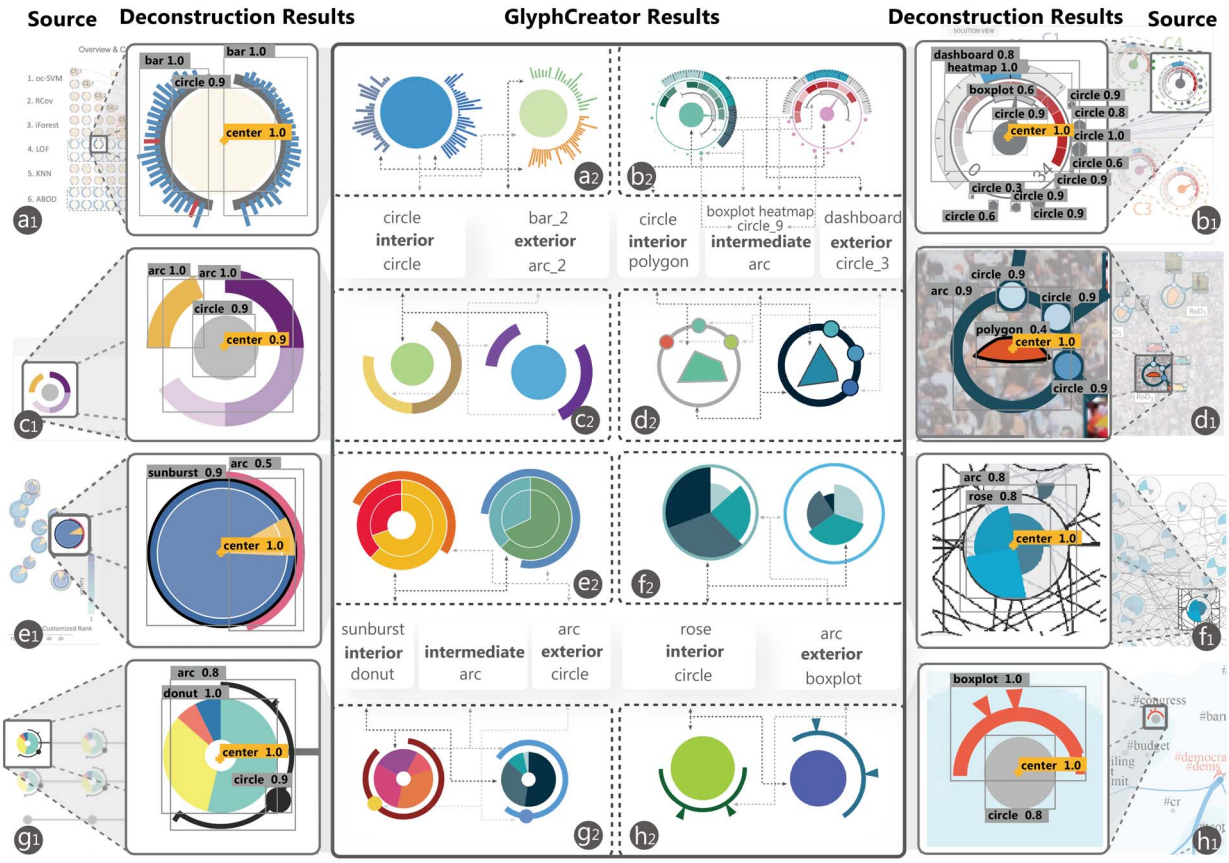

Fig. 1. Circular glyphs in (a1) EnsembleLens [68], (b1) SmartAdP [35], (c1) VisMatcher [28], (d1) VAICo [51], (e1) SeqDynamics [65], (f1) Visual IVO Editor [38], (g1) DropoutSeer [8], (h1) MutualRanker [34]. (a2)-(h2) shows the circular glyphs generated by GlyphCreator based on (a1)-(h1), respectively. Dashed arrows are used to associate glyph components with corresponding categories.

Abstract— Circular glyphs are used across disparate fields to represent multidimensional data. However, although these glyphs are extremely effective, creating them is often laborious, even for those with professional design skills. This paper presents GlyphCreator, an interactive tool for the example-based generation of circular glyphs. Given an example circular glyph and multidimensional input data, GlyphCreator promptly generates a list of design candidates, any of which can be edited to satisfy the requirements of a particular representation. To develop GlyphCreator, we first derive a design space of circular glyphs by summarizing relationships between different visual elements. With this design space, we build a circular glyph dataset and develop a deep learning model for glyph parsing. The model can deconstruct a circular glyph bitmap into a series of visual elements. Next, we introduce an interface that helps users bind the input data attributes to visual elements and customize visual styles. We evaluate the parsing model through a quantitative experiment, demonstrate the use of GlyphCreator through two use scenarios, and validate its effectiveness through user interviews.

Index Terms—Glyph-based visualization, machine learning, automatic visualization.

1 Introduction

Glyphs are widely used for multidimensional data visualization [20, 26, 55, 67]. Circular glyphs, one common layout type, are round and encode data using a polar coordinate system. Due to their artistic appearance and effectiveness at encoding in angle and radian channels [19], circular

- *L. Ying, T. Tang, Y. Luo, L. Shen, Y. Wu are with the State Key Lab of CAD & CG, Zhejiang University, Hangzhou, China. E-mail: {yingluu, tangtan, yzluo, fantast, ycwu}@zju.edu.cn. Yingcai Wu is the corresponding author.*
- *X. Xie is with Department of Sport Science, Zhejiang University, Hangzhou, China. E-mail: xxie@zju.edu.cn.*
- *L. Yu is with Department of Computing, Xi'an Jiaotong-Liverpool University, Suzhou, China. Email: Lingyun.Yu@xjtlu.edu.cn.*

Manuscript received 21 Mar. 2021; revised 13 June 2021; accepted 8 Aug. 2021. Date of publication 1 Oct. 2021; date of current version 22 Dec. 2021. Digital Object Identifier no. 10.1109/TVCG.2021.3114877

glyphs are one of the most commonly used visualizations, and make up 69% of glyphs in one large corpus that combines images from multiple visualization publications [13]. Visualization practitioners in various domains, including E-learning [8], urban applications [14, 15, 32, 60] and social media [6] prefer to use circular glyphs. For instance, TopicPanorama [57] uses circular glyphs to encode uncertainty, and StreamExplorer [62] adopts them to represent streaming data.

However, because there is such a large design space, creating circular glyphs from scratch is not easy [20]. For users with adequate visualization experience, the creation of a circular glyph involves a two-step approach: 1) design and 2) implementation. To ensure aesthetics and intelligibility, the design step should include a complete process of evaluation, refinement, and standardization, which requires artistic skills and creativity [17]. This task is quite challenging for junior visualization researchers, and even experts need to iteratively refine the design to achieve a satisfactory result. In the implementation step, users must bind data attributes to visual channels. General-purpose graphic design software (e.g., Adobe Illustrator) provides limited support for data binding. Designers usually need to customize glyph design instances for different data samples, which is tedious and time-consuming. Although computer programming can address the issue of reusability, a large group of designers might encounter difficulty when programming complex layout computations for the glyphs. To ensure design flexibility and ease of implementation, researchers have proposed several tools for creating data visualizations. However, some tools support regular charts but not circular glyphs (e.g., Lyra2 [73]), while some require users to perform complex operations, such as manually initializing all visual elements (e.g., Charticular [46]). A systematic learning process for the tools should also be considered.

With these challenges in mind, we attempt to address both aspects of the problem. For the design step, previous research paradigms (e.g., Text-to-Viz [45]) have shown that using an existing circular glyph as a reference to create another one [29] is an effective method. Making comments on a glyph is easier for users than creating one from scratch. Users can take the essence of the initial glyph and try to modify the design and create a similar one, which can shorten the creation time while maintaining quality. We improve the implementation step through automated mapping of data and visual elements. Visual elements in the reference circular glyph bitmap can be analyzed, and data can be bound into these elements automatically to simplify the creation process. To maximize convenience, we imagine whether the analysis and binding processes can be connected without human effort. Similar to data extraction, a model can help parse data from images. Overall, we envision a deep learning model that can deconstruct an online circular glyph example and use it to generate new circular glyphs. However, two critical obstacles exist:

Lack of Dataset. We currently lack a circular glyph dataset large enough to train a deep neural network. Although the percentage of circular glyphs among all glyphs is large, the overall image number in the VisImages dataset [13] is only 251, much smaller than the number required to train a model.

Model Architecture. It is difficult to automatically analyze and deconstruct a circular glyph bitmap. Previous research treated glyphs as entire units, and little work has been conducted to analyze the individual visual entities that make up a circular glyph. Moreover, due to current machine learning models' focus on natural images, no existing model allows a machine to deconstruct a circular glyph into pixels.

To address these challenges, we need to explicate the compositional elements of circular glyphs and consider the different ways in which data and visual elements are mapped.

Due to the aforementioned lack of data, it is difficult to extract correlations between visual elements and data. Therefore, we consider the possibility of understanding a circular glyph by analyzing the layout of visual elements. We propose a novel approach for automatically generating a new circular glyph based on imitation by deconstructing an existing circular glyph bitmap. We collect circular glyphs to explore their general patterns and the overall design space. To address the dataset shortage, we generate 11k circular glyph bitmaps based on our design space and train a neural network to interpret the bitmap images automatically. The model aims to detect all possible visual elements in the bitmap image and draw out the embedded layout. Then we implement GlyphCreator, which integrates the deconstruction model into the authoring process for circular glyphs. The major contributions of this study are as follows:

- We build a circular glyph dataset that incorporates all circular glyphs collected from VisImages [13] and other possible combinations covered by our design space. This dataset is available in `https://github.com/GlyphCreator/GlyphCreator`.
- We propose a framework for circular glyph deconstruction.
- We develop GlyphCreator, a system for automatically creating circular glyphs, and demonstrate it through a usage scenario. We also validate its usability through user interviews.

2 Related Work

Here we summarize prior studies that have covered understanding visualization with deep learning, glyph-based visualizations, and currently available authoring tools.

2.1 Understanding visualizations through deep learning

Understanding visualizations is a common task, and an increasing number of studies have applied machine learning methods to this task. Researchers have pursued several directions including data extraction, visualization redesign, and generating visualizations from data.

Researchers looking into data extraction have focused on different visualization types. Kembhavi et al. [25] devised a method based on long short-term memory architecture to parse the structure of diagrams. Siegel et al. [53] parsed figures from extant research (mostly line charts) by using a convolutional neural network (CNN)-based metric. Cliche et al. [11] introduced Scatteract, a system that extracts data from scatter plots through deep learning techniques and optical character recognition. Poco et al. [44] recovered visual encodings from chart images by using an end-to-end pipeline. Other systems, such as ChartSense [24], adopt a semi-automated approach, aiming to extract data from various chart images by combining human interaction with CNN.

With regard to redesign, iVoLVER [39] requires users to perform accurate chart interpretations through interactive annotation and builds new visualizations with the data. Poco et al. [44] recovered color mapping by providing an annotation interface and recolored a new image. Sun et al. [54] trained a dual conditional generative adversarial network (GAN) to colorize the contours of icons. Recently, Chen et al. [9] used an end-to-end deep neural network (DNN) to extract a timeline template from a bitmap image and generated new timeline infographics. Ma et al. [36] located and identified charts within an input image by using a learning-based model. Zhou et al. [71] used a neural network-based method to reverse engineer bar charts. Savva et al. [50] introduced a pioneer system called ReVision that identifies a chart type by using the SVM model, extracts visual elements and data, and automatically generates and outputs redesigned visualizations. However, the mark extraction in ReVision can only handle regular charts with a single mark type, such as bar charts, pie charts, and scatterplots. ReVision might fail to produce circular glyphs, which are usually composed of different types of visual elements and have complex layouts, sometimes with overlapped elements.

In addition to redesigning visualizations by generating them from images, redesign through data generation is also common. Wang et al. [58] proposed DataShot, which automatically creates fact sheets from tabular data. Shi et al. [52] introduced a visual story-generating system that uses tabular data. They integrated the Monte Carlo tree search algorithm into their system. Cui et al. [12] selected natural language statements as inputs and automatically generated infographics automatically. Qian et al. [45] utilized online blueprints to generate infographics by imitating examples using input text.

All existing approaches focus on how to understand visualization via deep learning methods. Our approach also belongs to this category. However, we focus on glyphs, an underexplored visualization type.

2.2 Glyph-based visualization and authoring

Presenting multivariate data [63, 69] is not easy. Glyph-based visualization is a common and effective format for presenting multivariate data [3]. However, to design and create an effective glyph is not trivial.

Therefore, the question of how to design a good glyph is an important problem for researchers. Ward [59] proposed a glyph generation pipeline that includes mapping data with visual elements and layout options. Borgo et al. [3] provided a state-of-the-art approach that focuses on glyph-based visualization and summarized design guidelines and techniques. Fuchs et al. [20] extended the guidelines from a different perspective by reviewing experimental studies, and obtained a deeper understanding of the glyph design space. Researchers have also explored glyph design in specific fields. Ropinski et al. [48,49] built a glyph taxonomy and conducted a survey on medical visualization. In the biological field, Maguire et al. [17] proposed a systematic approach for glyph design and demonstrated the approach through biological experiments.

Glyph-based visualization has been widely used in various fields, such as geo-information [16], sports [30,56], and 3D visualization [10]. Moreover, several systems [7,26,33,37] use circular glyphs to represent multidimensional data. Many authoring tools have been proposed to help users create glyphs for their data. Ribarsky et al. [47] developed Glyphmaker, a system that allows novice users to easily design a glyph. Recently, Xia et al. [64] introduced DataInk, a system that supports the creation of glyphs by adopting a pen-and-touch interactive input. Ren et al. [46] focused on layout and presented Charticular, an authoring tool that supports glyph generation. They divided the layout into chart- and glyph- levels and focused on the layouts of glyph-based visualization. However, we are also interested in the layout of glyphs, such as the visual elements that make up glyphs. Users need to create with these tools from scratch, which is a tedious task. We opt to imitate a bitmap glyph image and accomplish the data mapping process automatically.

3 The Pipeline of GlyphCreator

In this section, we first give an overview of GlyphCreator. Then we present the entire glyph creation process, including the construction of a circular glyph dataset and the model for layout deconstruction.

3.1 Overview

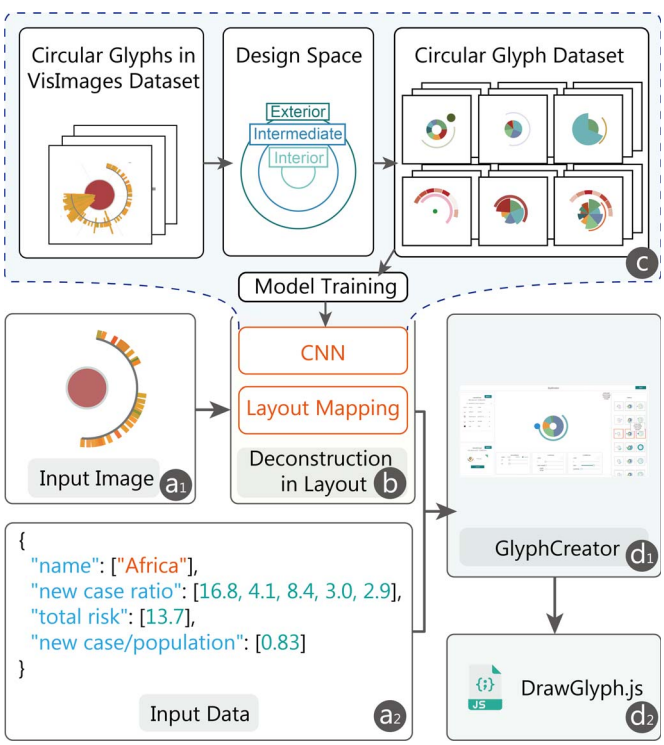

Fig. 2. The pipeline of GlyphCreator. (a1) Image and (a2) data are input. (b) The visual encodings are decoded and the layout is extracted by a CNN model. (c) The circular glyph dataset is generated for training the model. (d1) Users use GlyphCreator to combine the data and the layout and obtain the glyph renderer (d2).

We aim to help users design and create circular glyphs through a novel example-based approach. Our method identifies the structure and components of circular glyphs, and then creates a circular glyph according to the input example glyph and its structure and components. To automatically extract the structure and components, we need to train a detection model, which requires a dataset of circular glyph images. However, due to the incomplete investigation of circular glyphs, how to interpret the layout of circular glyphs remains unknown. With these considerations, we propose a working pipeline: generating a circular glyph dataset, training the machine, extracting the layout of the input example, and creating circular glyphs based on the extracted layout.

- **Circular Glyph Dataset.** Our goal is to construct a dataset that contains a diversity of circular glyphs. We initially select the VisImages dataset [13] since it is regarded as one of the complete datasets of visualization images. However, the number of circular glyphs in this dataset is still much smaller than the data size required by the training system. To solve this issue, we explore the design space of existing circular glyphs by analyzing the relationships between different visual elements. We list all possible variables that combine to compose a circular glyph, and randomly generate 10k images with annotations based on this design space to form the final dataset.
- **Layout Deconstruction.** We decide to deconstruct a circular glyph layout automatically via a two-step method (Fig. 2(b)). First, partial information is detected as visual elements. Second, the whole layout is obtained by mapping the detection results within the circular glyph design space. Based on the aforementioned VisImages dataset, we train a deep learning model to detect all the visual elements in a bitmap and to obtain the bitmap's center point. With the output label, corresponding bounding box, and center point, we calculate the distance between the center point and each element and ultimately acquire a layout by mapping the design space's detection results.
- **GlyphCreator.** We propose GlyphCreator, an example-based automatic system for circular glyph generation (Fig. 2(d1)). Users input an example circular glyph image that they like, as well as their data. Our system deconstructs the input glyph into components and binds the user's data with these components. As the output, the system generates a list of circular glyph candidates. After selecting a satisfactory glyph, users can export the result in code form (Fig. 2(d2)) and use it in their system with minimal effort.

3.2 Circular Glyph Dataset

We obtained a circular glyph dataset by collecting existing images in this category, exploring their design space, and generating various circular glyphs with annotations to form the final dataset.

3.2.1 Circular glyphs in VisImages dataset

To collect circular glyphs, we started with existing image datasets in visualization. Several datasets, such as MassVis [4], Beagle [2], and VisImages [13], collect numerous visualization images and classify them by design type. We selected the VisImages dataset [13] as our data source because it collects images from top visualization conferences and journals, such as IEEE VAST and InfoVis. Moreover, it specifically contains a "glyph" category, with 251 images. We filtered the circular glyphs according to two criteria:

- **C1** *images that have at least one visual element with radial contour whether intact or not;*
- **C2** *images that have at least two kinds of visual elements.*

Based on these criteria, three co-authors of this paper reviewed all 251 images and identified circular glyphs individually. The co-authors reached a consensus on 201 (80%) of the images and ended up with 162 circular glyphs. Within the remaining 50 images, some glyphs were difficult to recognize because they were small or blended into the background. After a thorough discussion of these images, the co-authors identified 16 circular glyphs. Finally, a total of 178 images were selected as circular glyphs that met the aforementioned criteria.

3.2.2 Analysis of circular glyphs

To explore the design space for state-of-the-art circular glyphs, we started with the template identified by Maguire et al. [17], which divides

a glyph into three regions; namely, mainbody, exterior, and interior. Although this template focuses on a particular design, this breakdown can be extended to the circular glyph layout in general. Four visualization researchers (co-authors of this work) with rich experience in designing and using glyphs participated in this entire process. Because having too many pictures or iterative rounds might keep them from reaching a conclusive design space, we selected a subset of 20 glyphs and checked whether it could work as well as the full set of 178 glyphs. Based on the selected images, we refined the design space over several iterative rounds, each consisting of three steps: deconstruction and classification, discussion, and refinement.

- **Deconstruction and classification.** We analyzed all selected images based on the template proposed by Maguire [17] and the current design space, which includes deconstructing the circular glyph into visual elements and classifying all visual elements as belonging to one of these types.
- **Discussion.** Through the results of the previous stage, we identified difficulties with the current design space. The segmentation degree of visual elements was unified, such as individual sectors in pie charts. If visual elements could not be classified within the current design space, we extended the space.
- **Refinement.** To solve the difficulties of each step, we refined the definition of visual elements and improved the design space. All participants reached a consensus about all visual elements, and identified a design space that can be applied to all 20 circular glyph images.

We divided all visual elements into four categories; namely, chart, shape, label, and icon. *Chart* uses graphic figures for data visualization, such as pie charts or line charts [23]. When acting as a visual element in a glyph, it is not necessary for a chart to show all the components it normally would, such as axes or a legend. *Shape* refers to a basic geometric object, such as a line, a circle, or a polygon. *Icon* refers to a small pictogram or ideogram. *Label* refers to a text label. After examining all circular glyphs in the dataset, we further discovered diverse patterns in the dominant chart category and categorized them into two sub-categories: circular and variant charts. Circular charts, like donut charts and pie charts, use a polar coordinate system. Others, such as heatmaps and boxplots, need to be transformed in order to achieve a circular shape and be placed in a polar coordinate system.

Without knowing the data underlying the image, we distinguished the visual elements based on their relative positions. If **C1** is satisfied, the entire circular glyph can be placed within a polar coordinate system. Therefore, we focused on parameters R and θ to locate each visual element. If several visual elements have the same sub-category and their R values are almost the same, they can be regarded as the same layer. For instance, in Fig. 1(d1), the three peripheral circles are regarded as one layer. Then, we divided a circular glyph into several layers. After examining all the collected circular glyph images, we classified all layers into three regions: interior, intermediate, and exterior. As a result, we have obtained a design space that covers the full collected circular glyph dataset, as shown in Fig. 4.

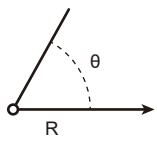

Fig. 3. A polar coordinate system

- **Interior**. The interior represents the closest layer of the circular glyph, which is indispensable. The visual element of the interior layer has three parts: chart, icon, and shape. Designers prefer pie charts (11/17) and donut charts (3/17) in the circle chart category. For the shape category, they use the circles (76/95) most. In Fig. 1(d1), the interior layer is denoted by the red polygon. In Fig. 1(a1)(b1)(c1)(h1), the center circle represents the interior, and in Fig. 1(g1), the layer denoted by the donut chart is the interior layer.
- **Exterior**. The exterior represents the outside layer of one circular glyph. The exterior layer can be classified into three categories: shape, chart, and icon. In the shape type, circles (90/120) are mostly used, whereas variant charts (12/21), such as boxplots and bar charts, are often used in the chart type. For example, in Fig. 1(d1), the three circles are included in the exterior as the same layer. The modified dashboard in Fig. 1(b1), the circle in Fig. 1(g1) and the variant bar chart in Fig. 1(a1) also denote exterior layers.
- **Intermediate**. The intermediate represents layers between the interior and exterior. The number of layers is greater than or equal to zero. An intermediate layer includes two categories of visual elements: chart and shape. No intermediate layer exists in Fig. 1(a1)(c1)(e1)(f1)(h1). The circular glyph in Fig. 1(b1) has three intermediate layers: an arc, a variant heatmap, and a circle set.

3.2.3 Generating training dataset

DNN requires a large amount of high-quality training data to ensure the accuracy and effectiveness of the detection model. We need a large and diverse collection of circular glyphs. To build our training dataset, we used D3 [5] to generate circular glyphs in two ways: by finding all possible layouts of circular glyphs within the design space, and then iterating various styles of a particular layout.

- **Layout**. We analyzed existing circular glyphs found within our design space. We found that the most common number of layers are two, three, and four, corresponding to zero, one, or two intermediate layers, one interior layer, and one exterior layer. To obtain all possible circular glyph layouts, we enumerated all visual representations for each layer. Therefore, the number of combinations was: $C_{n_i}^1 \times (C_{n_m}^0 + C_{n_m}^1 + C_{n_m}^2) \times C_{n_e}^1$, where n_i, n_m, and n_e are the numbers of visual representations of the interior, intermediate, and exterior layers. We manually checked the reasonableness of each combination.
- **Style**. We examined the size, position, and color of all visual elements that make up a circular glyph. For each element, we determined the number of parameters, as shown in Fig. 4(b). We changed red, green, and blue channels based on RGB color space to ensure color randomness. Based on a polar coordinate system (Fig. 3), we used R to indicate the radial coordinate and θ to indicate the angular coordinate. We combined two parameters and located the visual element's position. Notably, the elements in the interior layer have fixed positions. The same applies to all circle charts in other layers. For several variant charts (e.g., bar chart) and shapes (e.g., arc), two parameters are needed to indicate the angle: the starting angle and the angle range. When the visual element has only one parameter in the

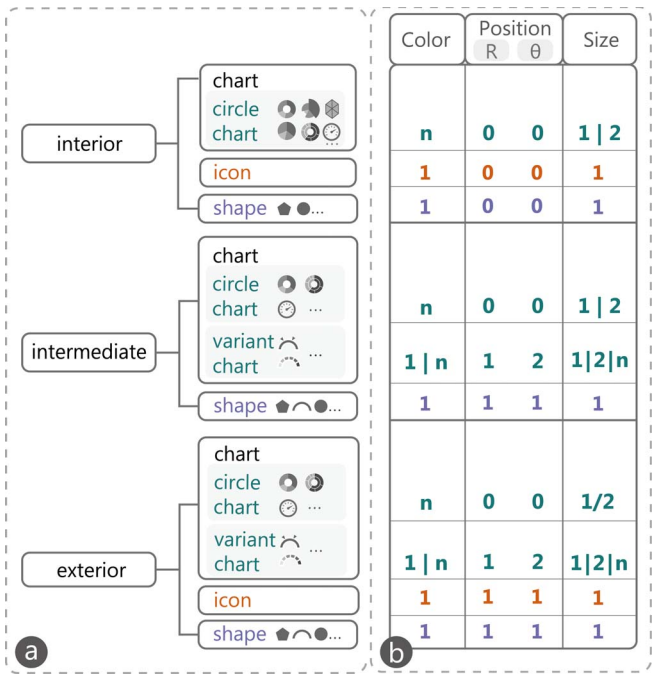

Fig. 4. (a) The design space of circular glyphs. A circular glyph can be divided into three regions: interior, intermediate and exterior. The second column indicates the contained categories of visual elements for each region. We used four different colors to represent four different categories. (b) The number of parameters for each visual element is shown in three dimensions: color, position and size. One number indicates a fixed number of parameters. "a | b" means "a" or "b".

size dimension, like pie charts or circles in the interior layer, then only R is required. For elements that have two size parameters (e.g., donut charts), $R1$ and $R2$ are essential. For bar charts, $\Delta R = (R1 - R2)$ is regarded as the parameter, and for arc, ΔR indicates the thickness.

We randomly generated data and assigned them as the parameters of corresponding visual elements (Fig. 4(b)) for data encoding based on all styles. We used this process to create 10k circular glyphs.

Annotation. All detection tasks require an annotated dataset. We used the same metrics as in the Microsoft Common Objects in COntext (MS COCO) dataset [31], which is cost-efficient and of high quality, to unify the format of annotations. At the labeling stage, we outlined the bounding box of each visual element. When drawn by D3 [5], labeling all visual elements through direct calculation was not time-consuming. Finally, each circular glyph was converted from SVG to PNG format and annotated with its representation and layout.

3.3 Layout Deconstruction

Due to the absence of fixed rules governing the styles and layouts of circular glyphs, parsing one to extract layout information is difficult . We achieve this goal by following two steps: first, obtaining the partial visual element information, second, elucidating the overall layout. The first step involves the detection of visual elements in a circular glyph. The second means obtaining the entire layout by locating the polar coordinate system and mapping the elements with the design space.

3.3.1 Detecting Visual Elements

Object detection is a popular and well-developed task within the computer vision domain. It is similar to ours: identify objects in an image with their associated category and outline the profile. However, object detection focus on natural images, whereas we are interested in circular glyph visualization specifically. The detection task is simpler for abstract images compared to natural ones due to clear boundaries. We defined 12 types of target visual elements (Fig. 4) that may be present in a circular glyph. The 12 categories include: donut charts, rose charts, radar charts, pie charts, sunburst charts, gauge charts in the circle chart category (**6**); icon (**1**); polygons, circles and arcs in the shape category (**3**); boxplots and heatmaps in the variant chart category (**2**). Our main goal is to deconstruct the layouts of circular glyphs by detecting all visual elements – in other words, what appears and where.

A possible solution is to train an existing object detection model using our dataset. State-of-the-art detectors can be broadly divided into one-stage detectors, which have high speed, and two-stage detectors, which have high accuracy. As a one-stage anchor-free detector without post-processing, CenterNet [72] can identify each object with a category and outline its bounding box in real time. It outperforms a range of state-of-the-art algorithms with a speed-accuracy tradeoff. However, for the second task of locating the polar coordinate system, the network needs modification for center localization. Moreover, CenterNet [72] cannot handle two objects with the same center because they will be regarded as the same object.

Inspired by Zhou et al. [72], we built a detection model by regarding visual elements as points. With the model, we could detect all visual elements accurately in real time and simultaneously locate the origin of the polar coordinate system.

Training Dataset. Given the dataset generated in Sec. 3.2.3, we provided extra information about the origin of the polar coordinate system. To guarantee the accuracy and efficiency of the detection model, we preserved the annotation metrics used in the MS COCO dataset [31]. Specifically, for each circular glyph image in the dataset, instead of giving the x, y value of the center point, we added a bounding box in the annotations whose center is the same as the origin and labeled it as *Center*. To avoid center point collision – two objects with the same center – two conditions regarding the size of the center bounding box in one circular glyph image must be met.

- If the image has one visual element whose center point is the same as the origin (e.g., the center circle in Fig. 1(a1)), the corresponding element is annotated with another label *Center*.
- If all visual elements' centers do not collide with the origin (e.g., the circular glyph in Fig. 1(d1)), we define a new, small bounding box containing the center and annotate it with the label *Center*.

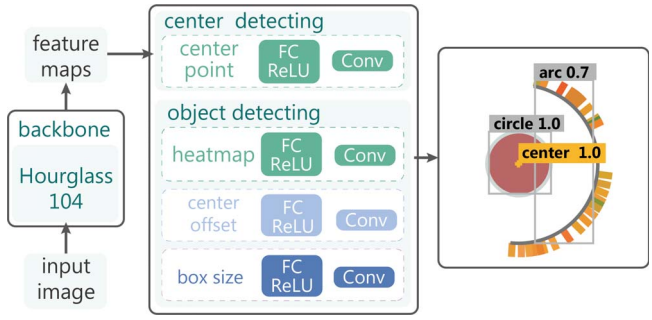

Fig. 5. Model architecture to detect visual elements and the center point. The backbone network extracts the feature map from the input image. Four neural networks then predict the bounding box and center location separately.

Model Architecture. Given an RGB circular glyph bitmap $I_c \in R^{W \times H \times 3}$, where R refers to the bitmap, W, H refers to the width and height, 3 refers to three color channels, the model aims to predict the center position of glyph (x_c, y_c) and bounding boxes $\{B_{vs}\}$ of all visual elements. We used a fully-convolutional network to obtain the feature map F from the input image I. The output prediction was downsampled by an output stride r. We selected the stacked hourglass network, up-convolutional residual networks (ResNet [22, 66]), and deep layer aggregation (DLA [70]) as candidates. After the experiment in Sec. 5.3, we employed the *hourglass* network as the backbone.

CenterNet [72], a network for object detection, approaches this goal through center localization and size regression by modeling an object as a single point. With feature map F extracted by the backbone, CenterNet produces a heatmap for each category using a Gaussian kernel for localization. For regression, CenterNet predicts the height and width of the object and the offset to recover the error due to output stride r. Our task includes object detection and center detection. We used a similar strategy for object detection for all categories in Fig. 4(a) and considered the label *Center* for center detection. Focused on the position of the center, we aim to obtain the heatmap of *Center*. The size prediction and the offset are not necessary. Therefore, for each object K with category c_k, the objective loss is

$$L_k = \begin{cases} L_h, & c_k = Center \\ L_h + \lambda_{size} L_{size} + \lambda_{off} L_{off}, & otherwise \end{cases} \quad (1)$$

where L_h is the loss of the heatmap, L_{size} is the loss at the center point, and L_{off} is the offset loss. These definitions are similar to those in CenterNet [72], and we refer readers to the description of CenterNet [72] for additional details. We scaled the loss by two constants λ_{size} and λ_{off}, and set $\lambda_{size} = 0.1$ and $\lambda_{size} = 1$ in all our experiments. Total loss L is composed of individual loss L_k for each object.

We used a single network to predict heatmap Y, offset O, and size S. Sharing a common fully-convolutional backbone network, all outputs are obtained, and then made to pass through a separate 3×3 convolution ReLU and another 1×1 convolution. An overview of the network input and output is shown in Fig. 5.

3.3.2 Mapping Elements with Layout

After understanding the content using the network, the next step was to map the detection elements with the design space. Given the origin of the polar coordinate system, we calculated the distance of different visual elements and compared their values for mapping.

We define the distance of each visual element using two conditions. For the first condition, as shown in Fig. 6(a), neither the x range nor y range of the bounding box goes through the corresponding value of the origin. The distance of this visual element is close to the linear distance of origin and the center point of the bounding box. Given the bounding box and element type, predicting the region's actual shape is difficult. Therefore, we opted to use the bounding box for calculation. If one value, whether x or y, is in the bounding box's range, as shown

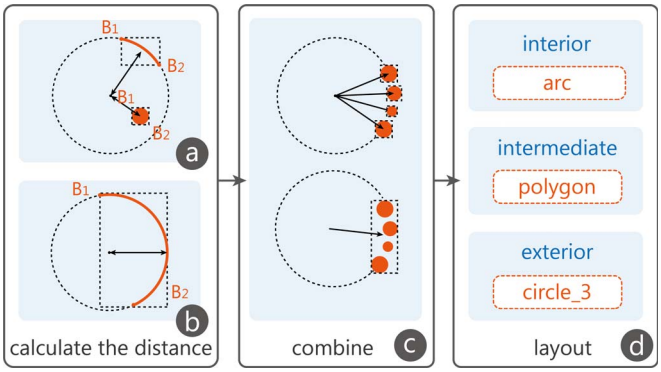

Fig. 6. Process of mapping each visual element with the layout. The first step is to calculate the distance for each visual element based on all bounding boxes and a center point. (a) The bounding box without the center point and (b) the bounding box containing the center point correspond to the two conditions in Equation. 2. (c) The elements are combined. (d) The final layout is obtained.

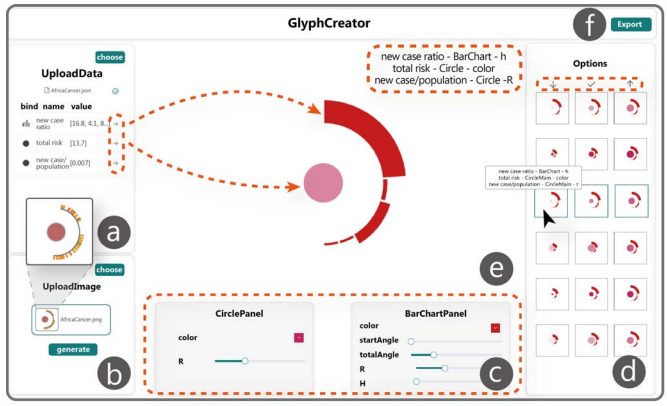

Fig. 7. GlyphCreator: (a) Data and (b) image can be upload. (c) Users can manipulate each visual element in individual editing panels. (d) Options for different encodings are shown to users. (e) Users can preview the generated circular glyph design and (f) export it.

in Fig. 6(b), then the distance is represented with the true value R. The concrete definition is:

$$D_v = \begin{cases} \max\limits_{i=1,2}(|x_{B_i} - x_O|, |y_{B_i} - y_O|), & x_O \in [x_{B_1}, x_{B_2}], \\ & y_O \in [y_{B_1}, y_{B_2}] \\ d(B_{center}, O), & otherwise \end{cases} \quad (2)$$

where O is the origin of the polar coordinate system; B_1, B_2 is marked in Fig. 6(a)(b); B_{center} is the center point of the bounding box; x_p, y_p represent the x, y value of the corresponding point p, which can be O, B_1, B_2; and $d(p)$ is the linear distance of the origin and the point.

When the distance between two same-type visual elements was too small, we merged them into the same layer. We discovered that they share the same encoding mode due to falling within the same category and close position for elements of this category. Therefore, they should be bound with the same data attribute. After a thorough investigation of existing circular glyphs, we found that 5% of the radius of a circular glyph (the distance of the most distant visual element) performed well in all situations. Then, we define the distance of each layer as:

$$Dist_l = \arg\min_{v} d(v) \quad v \in l \quad (3)$$

Based on the definition above, we combined visual elements with distances within a small gap into the same layer. We also marked the number of elements for further generation (e.g., the *circle_3* in the exterior layer in Fig. 1(d1)).

Finally, we obtained all layers, together with their category and distance. We sorted all distances by value and assigned the smallest to the interior, the largest to the exterior, and the middle to the intermediate.

4 The GlyphCreator System

This section presents GlyphCreator, an automatic example-based system that facilitates the easy generation of circular glyphs. Users can use GlyphCreator to produce a satisfactory circular glyph nearly effortlessly, simply by uploading relevant data and a reference image designed by experts. The reference image ensures quality, while the tool reduces the high effort usually necessary to produce circular glyphs.

4.1 Design Considerations

We iteratively improved our design considerations through a careful review of relevant literature and discussion with two researchers. The three primary design considerations of GlyphCreator are as follows:

DC1. Support an easy-to-learn [40] and easy-to-use generating workflow. The current workflow for generating circular glyphs is time-consuming, which inspired us to simplify the process. GlyphCreator targets all users of data visualizations, including those who excel in design and those without extensive design experience. The learning cost of a complex tool is high for general users [21]. Moreover, users are not willing to spend a long time editing after they find a satisfactory reference image. It is necessary to create a tool with a low learning cost [40] and easy-to-use interactions. Therefore, we designed our glyph-generating tool with a straightforward workflow and a simple editing interface.

DC2. Generate custom and appropriate glyphs quickly and easily. Different users have different style preferences. Although it is relatively easy for inexperienced users to create a glyph based on an existing design, the final result will lack creativity. Therefore, we aim to support end users in creating their own uniquely styled glyphs easily and intuitively. This requires balancing two considerations: On the one hand, users should be supported in learning layouts from existing circular glyphs. On the other hand, they should have the freedom to design their own circular glyphs in accordance with their preferences.

DC3. Support a reusable [40] and editable circular glyph output. In many visualization applications, multiple circular glyphs are combined to make up a larger glyph-based visualization. Thus, a tool that makes it easy to reuse [40] and edit a circular glyph is attractive. For editability, users of GlyphCreator can modify data and produce a new glyph version. For reusability, users can easily use the output circular glyph in their visualization system to see the real-time effect of any edits, thus saving time that might have been spent on multiple design iterations.

4.2 System Workflow

To support automatic example-based generation of a circular glyph, our system needs two inputs: data and an example circular glyph image. By allowing users to upload an example image, we have transformed a labor-intensive and time-consuming workflow into an automatic and time-saving one, in which our system easily understands the reference image and binds data with the visual elements.

Following **DC1**, users first upload the multivariate data and circular glyph bitmap to the system. Next, our deconstruction model extracts the layout and creates an initial circular glyph for the input data. The layout of a circular glyph rather than the specific style (such as the color of a visual element) is extracted following **DC2**.

We used a heuristic approach to pre-process the input data. For each data attribute, we first determine the number of parameters required for visual elements. Specifically, singular value (e.g., average value), paired values (e.g., min-max range), and other values require 1, 2, and n parameters, respectively. Second, we match the data attributes to visual elements according to the number of parameters (Fig. 4(b)). A successful match is shown in Fig. 8. For example, "the risk of developing cancer" with one parameter was matched to the color of the

circle, which also has one parameter. By default, we recommend the circular glyph whose encoding is commonly used by experts. Users can then skim through all the images generated by our system and choose one for further editing.

4.3 System Interface

Following **DC1**, we placed one function into one view. As shown in Fig. 7, our system has four views, namely, Upload, Preview, Edit, and Options. In the Upload view, a user can upload data (in JSON format) and an image. Next, multivariate data are shown in detail as *name* and *value* attributes in the UploadData panel, and the bitmap image is presented in a small preview window. Then, the user can click the *generate* button to go to the next step. Many generated circular glyphs are shown in the Preview view and Options view. The image in the preview windows is the one with the most common encoding from the Options view. In the Options view (Fig. 7(d)), only the center column (the tick column) corresponds to the valid input data. The left and right columns (with up and down arrows) highlight the encoding attribute for each visual element by making the value smaller or larger. We also present the encoding information in a text format to help users ensure the correct encoding. When hovering over the glyphs in the Options view, the encoding information can be seen as shown in Fig. 7(d). By dragging data (using the corresponding arrows in Fig. 7(a)) to the visual elements in the Preview view (Fig. 7(e))), users can bind their data with the glyph. After dragging, some unmatched encoding in the Options view disappears. The user can choose one circular glyph in the Options view by selecting the suitable attribute that encodes the data. After selecting one circular glyph from the alternatives, the Preview view is replaced by the new circular glyph and corresponding details, including the layout and encoding information near the upper right-hand corner (Fig. 7(e)). We developed one panel for each visual element in the Edit view (Fig. 7(c)) to support small changes, such as those in color and size. Notably, we encoded the attribute with unavailable data to ensure the correctness of data expression. For charts that use multiple colors, such as pie charts, we provided users with several popular color schemes for an improved result. For the other numerical attributes, users can drag the button on the slider to change the value. Then, the well-designed circular glyph can be exported in *JavaScript*, following **DC3**. With the exported file, the user can easily create new circular glyphs with new data by calling on the *DrawGlyph* function.

4.4 Implementation

We employ a client-server architecture to develop GlyphCreator, which comprises a backend that runs the deconstructing model to understand a circular glyph bitmap and a web interface that allows users to choose and edit glyphs. The web interface is implemented using JavaSrcipt and Vue framework, which supports uploading data and images and the editing of circular glyphs. The server side is built in python and PyTorch [43], the popular machine learning library. We also use a well-established graphic library, namely D3 [50], to render all visual elements in circular glyphs.

5 EVALUATION

This section presents two usage scenarios, user interviews, and a quantitative experiment to demonstrate the effectiveness of GlyphCreator.

5.1 Usage Scenarios

This section presents two usage scenarios using different datasets. The first scenario demonstrates the entire process of creating tailored glyphs with GlyphCreator as performed by a junior visualization researcher, Lucy. Lucy's job is to implement a system for analyzing the worldwide distribution of cancer. The dataset [42] that she is using comprises the incidence (new cases), mortality (deaths), prevalence, and region of each type of cancer in 2020. To see how GlyphCreator helps Lucy design and use circular glyphs, we lay out her workflow step by step.

Because Lucy was unfamiliar with glyph visualization, she looked for inspiration on the Internet. Cao et al. [6] proposed an interesting glyph to visualize the overall behaviors of an online learner (Fig. 2(a1)). The size of the inner circle of this glyph encodes an important piece of data about the learner, while the bar chart wrapped in the glyph encodes more detailed information. Lucy's data can easily be encoded by such a glyph design. Each of the world's regions can be represented by a glyph. The bar chart can be used to show the percentages of new cases represented by different cancers (e.g., breast cancer accounts for 16.8% of all cancers in Africa). Lucy chose to include the top five most prevalent cancers in the world. The ratio of new cancer cases to overall population is the most important attribute, and is thus encoded by the circle size. Moreover, Lucy decided to use colors to encode the risk of developing cancer before the age of 75. In this manner, regions with a large ratio and a high risk are highlighted.

Lucy prepared the data and wrote a JSON file containing all data dimensions for a particular region, Africa. After uploading the data, our system showed the information in *bind* (initially an empty box), *name*, and *value* (Fig. 7). Next, Lucy saved one individual glyph image from the Cao et al. paper, then uploaded it to the GlyphCreator system.

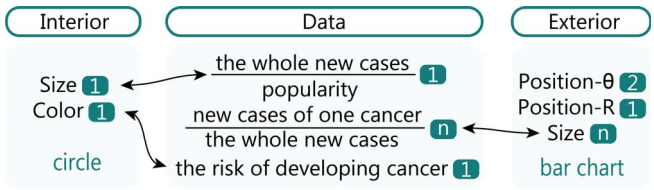

Fig. 8. A mapping example between visual elements and input data. The numbers in dark green boxes show the parameter number for each attribute or each data dimension. The arrow indicates the mapping.

After Lucy clicked the "generate" button, she immediately noticed the circular glyph in the Preview view in the center of the webpage, the two bottom editing panels (*CirclePanel* and *BarchartPanel*), and the various images in the corresponding Options view. To create her desired encoding, she dragged the arrow of the "new case ratio" row to the bar chart in the center. Then, she noticed the options shown in three columns (Fig. 7(d)). She compared the left and right images of each row to see what the glyph would look like if the value of the data increased or decreased. Moreover, when she hovered over the circular glyph image in the Options view, text appeared describing how the data were encoded. For example, *total risk - circle - color* indicates that the total risk data will be bound with the color of the inner circle.

Given the previous considerations, Lucy decided to check the Options view to find a glyph version that she likes. She chose the encoding *total risk - circle - color, new case ratio - bar chart - size, new case/population - circle - size*. The mappings are shown in Fig. 8.

The size and the color of the circle encode the data labeled "new case/population" and "total risk," respectively. The size of the bar chart represents the proportion of new cases of different types of cancer to the overall number of new cancer cases in Africa.

After seeing the default glyph in the Preview view, Lucy decided to make several changes. Satisfied with the inner circle, she chose to adjust the bar chart. Excluding the size encoded with data, she changed the bar chart's starting angle by dragging the small button in the corresponding slider (Fig. 7(e)).

Ultimately, she exported the circular glyph in code form, which can be easily read by her system. With the export file in JavaScript format, Lucy called on the function *DrawGlyph* in her system by providing the specified variables: the center point of the expected circular glyph location, the glyph size, and, most crucially, the data in a JSON format. Moreover, by iterating the functions using the data of different regions, such as Asia and Europe, Lucy drew six circular glyphs corresponding to six continents in one map with ease. The result is shown in Fig. 9. Now familiar with the system workflow, Lucy went on to design other versions by making different choices about the inner circle.

To demonstrate how a more comprehensive glyph can be designed by GlyphCreator, we present another usage scenario. Andrew is a business intelligence analyst with basic data visualization knowledge. He wanted to analyze car sales over the past 12 years using Skoda's UK Used Car Dataset [1]. He followed Lucy's approach to create a more complex tailored glyph using GlyphCreator, which he subsequently adapted into a glyph-based visualization (Fig. 9(b)). In the visualization,

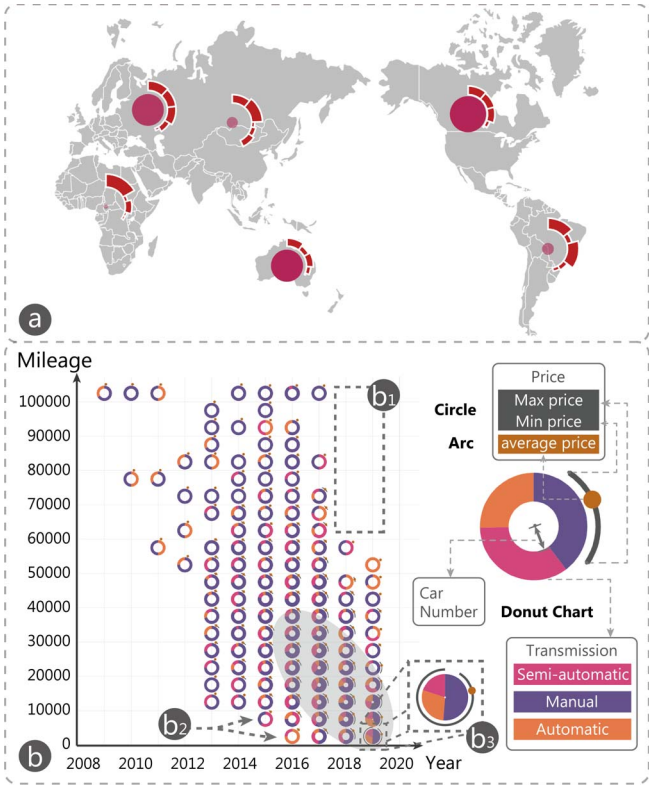

Fig. 9. Two visualizations based on the tailored glyphs created by GlyphCreator using the worldwide cancer dataset (a) and the UK Used Car Dataset of Skoda (b).

each glyph represents information about used cars sold over the past 12 years. The visual elements of the circular glyphs encode multivariate information about the cars. For instance, the inner donut chart shows three different types of cars (pink: semi-automatic cars, purple: manual cars and yellow: automatic cars). The width of the inner donut chart ($R_{outer} - R_{inner}$) shows the number of sold cars (a thicker band indicates that more cars were sold). The arc in the intermediate layer illustrates the price range, and the circle within the arc presents the average price. Glyphs are placed in a Cartesian coordinate system, where the x-axis represents the year and the y-axis represents the mileage.

Andrew discovered some interesting patterns with this visualization. In general, glyphs at lower positions (oval shadow in Fig. 9(b)) had a larger donut width, indicating that cars with lower mileage (between 5,000 and 40,000) sold slightly better than those with higher mileage. In the two most recent years (2018 and 2019), no car was sold with a mileage higher than 60,000 (Fig. 9(b1)). However, the lowest-mileage cars in a given year do not generally have the highest sales numbers (Fig. 9(b2)), likely because many of them are new cars and thus unlikely to be resold. An interesting finding is that sales of low-mileage cars increased drastically in 2019. This may be related to the fact that Skoda recalled thousands of cars in the UK that year [41]. As a result, people might have lost confidence in the brand and sold their new cars, even though the models they owned were not recalled. Andrew also noticed that the glyph at the bottom right (Fig. 9(b3)) has an outer arc that is significantly longer than those of the other glyphs, but the bullet position is similar to the neighboring glyphs. By exploring the data, he found that one car of a common model was sold at an extremely high price for an unknown reason.

5.2 User Interview

To evaluate the effectiveness and usability of GlyphCreator, we conducted semi-structured interviews with four end users who were familiar with data visualization. The first user (**U1**) was a senior researcher who graduated from a professional design school. He also studied visualization and visual analysis for four years. The second user (**U2**) worked as a senior visualization researcher for three years. However, he did not have systematic design training. The third user (**U3**) was a student majoring in *computer science* and had little prior experience in visualization, all of which was obtained from an *Information Visualization course* (eight weeks, four lectures per week). These three users (**U1, U2, U3**) evaluated the workflow of GlyphCreator and discussed its potential applications. The fourth user (**U4**) was a professional UI/UX designer who worked for a visualization group. Based on her visualization knowledge and design experience, she evaluated the output circular glyphs generated by GlyphCreator. She also compared the system with various commercial design tools (e.g., Adobe AI/PS).

Each 60-minute interview began with a five-minute introduction to the system workflow. Afterward, we demonstrated GlyphCreator by going through a three-minute example case. We let users become thoroughly familiar with our system through free exploration. They were then asked to generate two circular glyphs using a given reference image and data. Afterwards, a semi-structured interview was conducted to obtain opinions on the usability and quality of our system. In the interview, the users were asked to demonstrate how to use the system to create circular glyphs. We asked about their previous experience with glyph creation. We also asked them to freely share their thoughts and suggestions about the glyphs and the system as a whole, and to point out any causes of confusion.

Overall, the users were impressed with our system's convenience and intelligence. For example, **U3** said, "The process of editing a circular glyph by GlyphCreator is easy and straightforward." **U2** said, " It does a great job of shortening the time for creating a glyph." **U4** focused on system design, including the workflow, design of interactions, and user interface. She commented that "the interface is easy to follow and the interactions are intuitive."

The users liked the entire workflow design. Regarding the glyph generation process, **U1** and **U2** began from the data, whereas **U3** started with the reference images. All three users (**U1, U2**, and **U3**) said they read papers to find glyph designs that could meet their needs. **U1**, who has solid design skills, regularly collects good glyph designs in case they come in handy. The three users are used to designing and implementing glyphs through programming. Based on their previous design experience, they expressed the need for a system for generating circular glyphs because it could reduce the iteration time. Due to an indispensable reference image, they agreed with our input, including data and image. **U4** focused on the workflow of GlyphCreator. Comparing GlyphCreator with professional design tools, she commented, "GlyphCreator is more intelligent and efficient. I only needed several minutes to learn how to use the system." She also pointed out several possible improvements to the interface, such as adding hints on three columns in the Options view. We further improved our interface by providing it with a user-friendly, built-in user guide.

All users thought the system was easy to understand. We asked users to score the system on the generated circular glyphs' quality based on a 5-point Likert Scale, where 5 is the best and 1 is the worst. **U1** gave a 5, and **U2, U3, U4** gave a 4, which shows their high appraisal of the glyph aesthetic and its similarity to the original reference image. All users thought that the generated glyphs properly adopted the layouts of the original images. **U1** commented, "The glyphs look similar to the original ones, and I can freely edit other attributes." **U4**, a designer, said, "From the perspective of data encoding, I think the generated picture is reasonable. Calculating the size of each component when using a professional design tool is tedious."

We also received suggestions for further improvements. First, the users suggested we consider the data range to avoid blocking each visual element. For example, when using a circle as an interior layer and a boxplot as an exterior layer (Fig. 1(h1)), the circle's radius should not be larger than the boxplot's, even if users encode data on this attribute. To ensure correctness, we used a relative size for the inner element to replace the previous size. Moreover, the users suggested uploading two referred images or more to compare other possibilities.

5.3 Model Experiments

Our model was implemented using PyTorch [43] with four types of CNN backbone, namely, ResNet-18 [22], ResNet-101 [66], DLA-34 [70] and Hourglass-104 [27]. We used the Hourglass-104 network as the standard and modified both the ResNets and DLA-34 by using deformable convolution layers. To evaluate the performance of parsing a circular glyph, we adopted the average precision (AP) [18] value from the precision/recall curve to access two tasks, namely, what and where. Each detection was considered true or false based on the area that overlapped with the ground truth bounding boxes. Overlap area was calculated by the formula: $IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$, where B_p is the predicted bounding box and B_{gt} is the ground truth bounding box. If IoU exceeded a threshold, then we considered the detection as a true one. Without test augmentation, we evaluated our object detection performance on our circular glyph dataset, which contained 8k training images and 2k validation images. We reported the average precision over all IoU thresholds (AP) and AP at thresholds of 0.5 (AP_{50}) and 0.75 (AP_{75}). Table 1 shows results with different backbones. According to the definition, a high AP value denotes a good detection model.

Table 1. Average precision of detecting visual elements using different backbones in the circular glyph dataset.

Backbone	AP	AP_{50}	AP_{75}	Time(ms)
Hourglass-104	82.6	98.4	94.1	92
DLA-34	82.2	98.2	93.2	45
ResNet-101	80.8	98.1	92.7	36
ResNet-18	79.0	97.6	91.4	20

Table 1 shows the results with different backbones. The running time was tested on a local machine, with Intel(R) Xeon(R) Platinum 8260 CPU, Tesla V100 GPU, PyTorch 1.5.0, and CUDA 10.2. Considering that all backbones take less than 0.1 seconds to deal with one image, we selected the best one. Among all backbones, Hourglass-104 had the highest accuracy with a relatively good speed. Therefore, we chose the Hourglass-104 backbone for model training.

6 Discussion

This section discusses the implications and limitations of GlyphCreator.

6.1 Implications

Loop of Visualization and Artificial Intelligence (AI). Recently, using AI to aid in data visualization has become more common [61]. We use AI to simplify the visualization task, and use visualization to help in the machine learning task, creating a loop of visualization and AI. To deconstruct a circular glyph for the former arrow automatically, we trained a detection model that satisfied our scenario. The lack of a training dataset led us to draw the second arrow. We used *D3.js* [5] to draw circular glyphs, and saved the PNGs and annotations as the training dataset. Moreover, we utilized simple calculation as a substitute for the labor-intensive annotation process. For an improved authoring tool benefiting the community, we need a better model with high accuracy that is fast enough for quick editing. For a perfect model, a comprehensive understanding of the design space of circular glyphs for data visualization must be obtained. Through multiple iterative rounds, we achieved a well-behaved model and a tool, GlyphCreator, with high efficiency. GlyphCreator is also inspiring, providing a successful work example rooted in two fields of knowledge and thus inspiring the development of future visualization tools.

Example-based Circular Glyph Generation. As combinations of data-driven visual entities, glyphs use different visual channels to encode multiple informational dimensions. Although glyphs are effective, their complex layouts and multiple encodings can make the design and realization process difficult. Therefore, automating the generation of glyphs is a meaningful task. Compared with creating from scratch, example-based generation is more efficient. In real scenarios, users with different design skills in the visualization field can use GlyphCreator. Users with design expertise usually have a clear goal for their glyph, and want to see the final result as fast as they can for iteration. Unlike when this process required tedious drawing and binding, they can see the image only after a few minutes of operation. Moreover, they can use the glyph in their system, previewing it quickly with our tool. Users with little design experience have a clear understanding of glyphs but find the design process difficult. They want to use an existing glyph as a reference for their own data. With our tool, they can obtain their desired glyph even when they have little understanding of the original image. These scenarios lead us to believe that our example-based glyph generation method provides improved efficiency to users who need to design glyphs, even without a complete understanding of the form. Such a useful and effective means of representing multivariate data visualization could have a wide range of applications.

Application of the Dataset. Our dataset of circular glyphs can be used for other tasks, such as predicting visualization types and searching for a glyph image using keywords. For each glyph, we labeled each visual element with its category. A new model for chart prediction can be feasibly trained for all visual elements in circular glyphs. Moreover, people can use the labels in annotations to search for a desired circular glyph by inputting several keywords about the corresponding visual elements. During the design process, people aim to find a glyph with a particular visual element that is suitable for one data type, like a pie chart for proportional data. However, obtaining content results by inputting related keywords is difficult with popular search engines, such as Google. With our dataset that contains numerous images, users can acquire related images for inspiration or other possible usages.

6.2 Limitations

Our approach has several limitations. First, GlyphCreator has limited support for glyph style customization. There is a default style for all users, and users must edit it manually. A possible solution is improving the model to learn the style parameters of input images, such as color scheme, and automatically configure the generated glyphs. Second, we applied our approach only to circular glyphs, but it can be potentially generalized to other glyph types. By analyzing and exploring the design space of other glyphs, we can extend our dataset and build a model for parsing all kinds of glyphs. Third, the deconstruction model is not diverse enough to support circular glyphs in other fields. We collected circular glyphs designed by experts in the visualization field. Other fields, such as journalism, could use glyphs for data representation, and these glyphs could also be collected for diversity. The deconstruction model could be utilized for many other images with high accuracy by adopting a dataset with a broadened range.

7 Conclusion

In this paper, we introduce GlyphCreator, an automatic system for generating circular glyphs based on an example-based method. We collected existing circular glyphs and created a design space of circular glyphs to analyze the relationship among different visual elements. With this design space, we built a circular glyph dataset and proposed a framework that includes a deconstruction model to obtain the layout of circular glyphs. By utilizing the uploaded circular glyph reference image and multi-dimensional data in GlyphCreator, users can bind the data with the layout in a straightforward manner and generate a new circular glyph after revising the style. We evaluated the deconstruction model through a quantitative experiment. We also demonstrated the expressiveness and usability of our approach through a usage scenario and user interviews. We believe that our work provides a new idea for automatically generating glyphs in all visualizations via a two-way method, a loop of visualization and AI. In the future, we plan to extend this pipeline by analyzing other glyphs and expand the authoring tools of GlyphCreator to support other creative designs.

Acknowledgments

This work was supported by NSFC (62072400), Zhejiang Provincial Natural Science Foundation (LR18F020001), and the Collaborative Innovation Center of Artificial Intelligence by MOE and Zhejiang Provincial Government (ZJU).

REFERENCES

[1] 100,000 UK Used Car Data set. https://www.kaggle.com/adityadesai13/used-car-dataset-ford-and-mercedes. Accessed: 2021-6-30.

[2] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, and M. Stonebraker. Beagle: Automated Extraction and Interpretation of Visualizations from the Web. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, p. 594, 2018.

[3] R. Borgo, J. Kehrer, D. H. S. Chung, E. Maguire, R. Laramee, H. Hauser, M. Ward, and M. Chen. Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications. In *Proceedings of Eurographics*, pp. 39–63, 2013.

[4] M. Borkin, A. A. Vo, Z. Bylinskii, P. Isola, S. Sunkavalli, A. Oliva, and H. Pfister. What Makes a Visualization Memorable? *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2306–2315, 2013.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[6] N. Cao, C. Shi, W. S. Lin, J. Lu, Y. Lin, and C. Lin. TargetVue: Visual Analysis of Anomalous User Behaviors in Online Communication Systems. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):280–289, 2016.

[7] K. Chen, Y. Wang, M. Yu, H. Shen, X. Yu, and G. Shan. ConfVisExplorer: A Literature-based Visual Analysis System for Conference Comparison. *Journal of Visualization*, 24(2):381–395, 2021.

[8] Y. Chen, Q. Chen, M. Zhao, S. Boyer, K. Veeramachaneni, and H. Qu. DropoutSeer: Visualizing Learning Patterns in Massive Open Online Courses for Dropout Reasoning and Prediction. In *Proceedings of IEEE Conference on Visual Analytics Science and Technology*, pp. 111–120, 2016.

[9] Z. Chen, Y. Wang, Q. Wang, Y. Wang, and H. Qu. Towards Automated Infographic Design: Deep Learning-based Auto-Extraction of Extensible Timeline. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):917–926, 2020.

[10] E. B. Chlan and P. Rheingans. Multivariate Glyphs for Multi-Object Clusters. In *Proceedings of IEEE Symposium on Information Visualization*, pp. 141–148, 2005.

[11] M. Cliche, D. S. Rosenberg, D. Madeka, and C. Yee. Scatteract: Automated Extraction of Data from Scatter Plots. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases*, vol. 10534, pp. 135–150, 2017.

[12] W. Cui, X. Zhang, Y. Wang, H. Huang, B. Chen, L. Fang, H. Zhang, J. Lou, and D. Zhang. Text-to-Viz: Automatic Generation of Infographics from Proportion-Related Natural Language Statements. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):906–916, 2020.

[13] D. Deng, Y. Wu, X. Shu, M. Xu, J. Wu, S. Fu, and Y. Wu. VisImages: A Large-scale, High-quality Image Corpus in Visualization Publications. *CoRR*, 2020.

[14] Z. Deng, D. Weng, J. Chen, R. Liu, Z. Wang, J. Bao, Y. Zheng, and Y. Wu. AirVis: Visual Analytics of Air Pollution Propagation. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):800–810, 2020.

[15] Z. Deng, D. Weng, Y. Liang, J. Bao, Y. Zheng, T. Schreck, M. Xu, and Y. Wu. Visual Cascade Analytics of Large-scale Spatiotemporal Data. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[16] Y. Drocourt, R. Borgo, K. Scharrer, T. Murray, S. Bevan, and M. Chen. Temporal Visualization of Boundary-based Geo-information Using Radial Projection. *Computer Graphics Forum*, 30(3):981–990, 2011.

[17] Eamonn Maguire and Philippe Rocca-Serra and Susanna-Assunta Sansone and Jim Davies and Min Chen. Taxonomy-Based Glyph Design - with a Case Study on Visualizing Workflows of Biological Experiments. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2603–2612, 2012.

[18] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[19] V. A. Filipov, V. Schetinger, K. Raminger, N. Soursos, S. Zapke, and S. Miksch. Gone full circle: A Radial Approach to Visualize Event-based Networks in Digital Humanities. *Visual Informatics*, 5(1):45–60, 2021.

[20] J. Fuchs, P. Isenberg, A. Bezerianos, and D. Keim. A Systematic Review of Experimental Studies on Data Glyphs. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1863–1879, 2017.

[21] S. R. Haynes and T. G. Kannampallil. Learning, Performance, and Analysis Support for Complex Software Applications. *SIGHCI Proceedings*, p. 15, 2004.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778. IEEE Computer Society, 2016.

[23] C. Jensen and L. Anderson. *Harvard Graphics 3: the Complete Reference*. McGraw-Hill Osborne Media, 1992.

[24] D. Jung, W. Kim, H. Song, J. Hwang, B. Lee, B. H. Kim, and J. Seo. ChartSense: Interactive Data Extraction from Chart Images. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 6706–6717, 2017.

[25] A. Kembhavi, M. Salvato, E. Kolve, M. J. Seo, H. Hajishirzi, and A. Farhadi. A Diagram is Worth a Dozen Images. In *Proceedings of European Conference on Computer Vision*, vol. 9908, pp. 235–251. Springer, 2016.

[26] X. Kui, H. Lv, Z. Tang, H. Zhou, W. Yang, J. Li, J. Guo, and J. Xia. TVseer: A Visual Analytics System for Television Ratings. *Visual Informatics*, 4(3):1–11, 2020.

[27] H. Law and J. Deng. CornerNet: Detecting Objects as Paired Keypoints. In *Proceedings of European Conference on Computer Vision*, vol. 11218, pp. 765–781, 2018.

[28] P. Law, W. Wu, Y. Zheng, and H. Qu. VisMatchmaker: Cooperation of the User and the Computer in Centralized Matching Adjustment. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):231–240, 2017.

[29] B. Lee, S. Srivastava, R. Kumar, R. I. Brafman, and S. R. Klemmer. Designing with Interactive Example Galleries. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, pp. 2257–2266, 2010.

[30] P. A. Legg, D. H. S. Chung, M. L. Parry, M. W. Jones, R. Long, I. W. Griffiths, and M. Chen. MatchPad: Interactive Glyph-Based Visualization for Real-Time Sports Performance Analysis. *Computer Graphics Forum*, 31(3):1255–1264, 2012.

[31] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In *Proceedings of European Conference on Computer Vision*, vol. 8693, pp. 740–755. Springer, 2014.

[32] D. Liu, D. Weng, Y. Li, J. Bao, Y. Zheng, H. Qu, and Y. Wu. SmartAdP: Visual Analytics of Large-scale Taxi Trajectories for Selecting Billboard Locations. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):1–10, 2017.

[33] L. Liu, H. Zhang, J. Liu, S. Liu, W. Chen, and J. Man. Visual Exploration of Urban Functional Zones Based on Augmented Nonnegative Tensor Factorization. *Journal of Visualization*, 24(2):331–347, 2021.

[34] M. Liu, S. Liu, X. Zhu, Q. Liao, F. Wei, and S. Pan. An Uncertainty-Aware Approach for Exploratory Microblog Retrieval. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):250–259, 2016.

[35] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. T. Stasko. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of ACM Conference on Human Factors in Computing Systems*, p. 123, 2018.

[36] R. Ma, H. Mei, H. Guan, W. Huang, F. Zhang, C. Xin, W. Dai, X. Wen, and W. Chen. LADV: Deep Learning Assisted Authoring of Dashboard Visualizations from Images and Sketches. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2020.

[37] H. Mansoor, W. Gerych, A. Alajaji, L. Buquicchio, K. Chandrasekaran, E. Agu, and E. Rundensteiner. ARGUS: Interactive Visual Analysis of Disruptions in Smartphone-detected Bio-Behavioral Rhythms. *Visual Informatics*, 2021.

[38] B. McDonnel and N. Elmqvist. Towards Utilizing GPUs in Information Visualization: A Model and Implementation of Image-Space Operations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1105–1112, 2009.

[39] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. iVoLVER: Interactive Visual Language for Visualization Extraction and Reconstruction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 4073–4085, 2016.

[40] T. Murray. Theory-based Authoring Tool Design: Considering the Complexity of Tasks and Mental Models. *Design Recommendations for Intelligent Tutoring Systems. Authoring Tools and Expert Modeling Techniques*, 3:9–29, 2015.

[41] L. O'Callaghan. Volkswagen, audi & skoda recall: Thousands of cars suddenly lose power in dangerous fault. https://www.express.co.uk/life-style/cars/1190969/Volkswagen-Audi-Skoda-recall-news-affected-

model-list-news. Accessed: 2021-6-30.
[42] W. H. Organization. Population Fact Sheets about Cancer. https://gco.iarc.fr/today/fact-sheets-populations. Accessed: 2021-3-20.
[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, and et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. 2019.
[44] J. Poco, A. Mayhua, and J. Heer. Extracting and Retargeting Color Mappings from Bitmap Images of Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):637–646, 2018.
[45] C. Qian, S. Sun, W. Cui, J. Lou, H. Zhang, and D. Zhang. Retrieve-Then-Adapt: Example-based Automatic Generation for Proportion-related Infographics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):443–452, 2021.
[46] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive Construction of Bespoke Chart Layouts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):789–799, 2019.
[47] W. Ribarsky, E. Z. Ayers, J. Eble, and S. Mukherjea. Glyphmaker: Creating Customized Visualizations fo Complex Data. *Computer*, 27(7):57–64, 1994.
[48] T. Ropinski, S. Oeltze, and B. Preim. Survey of Glyph-based Visualization Techniques for Spatial Multivariate Medical Data. *Computers & Graphics*, 35(2):392–401, 2011.
[49] T. Ropinski and B. Preim. Taxonomy and Usage Guidelines for Glyph-based Medical Visualization. In *Proceedings of SimVis*, vol. 522, pp. 121–138, 2008.
[50] 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 ACM Symposium on User Interface Software and Technology*, p. 393–402, 2011.
[51] J. Schmidt, M. E. Gröller, and S. Bruckner. VAICo: Visual Analysis for Image Comparison. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2090–2099, 2013.
[52] D. Shi, X. Xu, F. Sun, Y. Shi, and N. Cao. Calliope: Automatic Visual Data Story Generation from a Spreadsheet. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):453–463, 2021.
[53] N. Siegel, Z. Horvitz, R. Levin, S. K. Divvala, and A. Farhadi. FigureSeer: Parsing Result-Figures in Research Papers. In *Proceedings of European Conference on Computer Vision*, vol. 9911, pp. 664–680, 2016.
[54] T.-H. Sun, C.-H. Lai, S.-K. Wong, and Y.-S. Wang. Adversarial Colorization of Icons Based on Contour and Color Conditions. In *Proceedings of the 27th ACM International Conference on Multimedia*, pp. 683–691, 2019.
[55] K. Umbleja, M. Ichino, and H. Yaguchi. Improving Symbolic Data Visualization for Pattern Recognition and Knowledge Discovery. *Visual Informatics*, 4(1):23–31, 2020.
[56] J. Wang, J. Wu, A. Cao, Z. Zhou, H. Zhang, and Y. Wu. Tac-Miner: Visual Tactic Mining for Multiple Table Tennis Matches. *IEEE Transactions on Visualization and Computer Graphics*, 27(6):2770–2782, 2021.
[57] X. Wang, S. Liu, J. Liu, J. Chen, J. Zhu, and B. Guo. TopicPanorama: A Full Picture of Relevant Topics. *IEEE Transactions on Visualization and Computer Graphics*, 22(12):2508–2521, 2016.
[58] Y. Wang, Z. Sun, H. Zhang, W. Cui, K. Xu, X. Ma, and D. Zhang. DataShot: Automatic Generation of Fact Sheets from Tabular Data. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):895–905, 2020.
[59] M. O. Ward. Multivariate Data Glyphs: Principles and Practice. In *Handbook of Data Visualization*, pp. 179–198. 2008.
[60] D. Weng, C. Zheng, Z. Deng, M. Ma, J. Bao, Y. Zheng, M. Xu, and Y. Wu. Towards Better Bus Networks: A Visual Analytics Approach. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):817–827, 2021.
[61] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu. AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2021.
[62] Y. Wu, Z. Chen, G. Sun, X. Xie, N. Cao, S. Liu, and W. Cui. StreamExplorer: A Multi-Stage System for Visually Exploring Events in Social Streams. *IEEE Transactions on Visualization and Computer Graphics*, 24(10):2758–2772, 2018.
[63] Y. Wu, D. Weng, Z. Deng, J. Bao, M. Xu, Z. Wang, Y. Zheng, Z. Ding, and W. Chen. Towards Better Detection and Analysis of Massive Spatiotemporal Co-Occurrence Patterns. *IEEE Transactions on Intelligent Transportation Systems*, 22(6):3387–3402, 2021.
[64] H. Xia, N. H. Riche, F. Chevalier, B. R. D. Araújo, and D. Wigdor. DataInk: Direct and Creative Data-Oriented Drawing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, p. 223, 2018.
[65] M. Xia, M. Xu, C. Lin, T. Y. Cheng, H. Qu, and X. Ma. SeqDynamics: Visual Analytics for Evaluating Online Problem-solving Dynamics. *Computer Graphics Forum*, 39(3):511–522, 2020.
[66] B. Xiao, H. Wu, and Y. Wei. Simple Baselines for Human Pose Estimation and Tracking. In *Proceedings of European Conference on Computer Vision*, vol. 11210, pp. 472–487, 2018.
[67] X. Xie, J. Wang, H. Liang, D. Deng, S. Cheng, H. Zhang, W. Chen, and Y. Wu. PassVizor: Toward Better Understanding of the Dynamics of Soccer Passes. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1322–1331, 2021.
[68] K. Xu, M. Xia, X. Mu, Y. Wang, and N. Cao. EnsembleLens: Ensemble-based Visual Exploration of Anomaly Detection Algorithms with Multidimensional Data. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):109–119, 2019.
[69] S. Ye, Z. Chen, X. Chu, Y. Wang, S. Fu, L. Shen, K. Zhou, and Y. Wu. ShuttleSpace: Exploring and Analyzing Movement Trajectory in Immersive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):860–869, 2021.
[70] F. Yu, D. Wang, E. Shelhamer, and T. Darrell. Deep Layer Aggregation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2403–2412, 2018.
[71] 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*, 24(2):419–435, 2021.
[72] X. Zhou, D. Wang, and P. Krähenbühl. Objects as Points. *CoRR*, abs/1904.07850, 2019.
[73] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan. Lyra 2: Designing Interactive Visualizations by Demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):304–314, 2021.