Sketch2Data: Recovering Data from Hand-Drawn Infographics

Anran Qi[a], Theophanis Tsandilas[b], Ariel Shamir[c], Adrien Bousseau[a]

[a]*Centre Inria d'Université Côte d'Azur*
[b]*Université Paris-Saclay, CNRS, Inria, LISN*
[c]*Reichman University, Herzliya, Israel*

Abstract

Data collection and visualization have traditionally been seen as activities reserved for experts. However, by drawing simple geometric figures — known as *glyphs* — anyone can visually record their own data. Still, the resulting *hand-drawn infographics* do not provide direct access to the underlying data, hindering digital editing of both the glyphs and their values. We introduce a method to recover data values from glyph-based hand-drawn infographics. Given a visualization in a bitmap format and a user-defined parametric template of its glyphs, we leverage deep neural networks to detect and localize the visualization glyphs, and estimate the data values they represent. We also provide a user interface to review and correct these estimates, informed by a measure of uncertainty of the neural network predictions. Our reverse-engineering procedure effectively disentangles the depicted data from its visual representation, enabling various visualization authoring applications, such as visualizing new data values or experimenting with alternative visualizations of the same data.

Keywords: Data visualization, infographics, glyphs, drawing analysis

1. Introduction

In computer graphics, data visualization is one of the principal ways to engage with data, communicate it, and understand it. A common belief is that visual depictions of data should be created by graphic designers or visualization experts. However, many people may wish to create, observe, and analyze data visually — for example, to track trends and traits in their daily lives [1, 2] or better understand scientific phenomena [3].

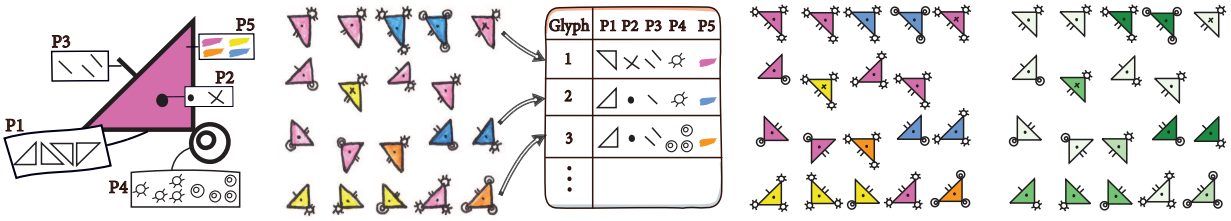
Unfortunately, tools for collecting and visualizing data can be complex and cumbersome for non-professionals. In contrast, drawing is a simple, accessible way for anyone to *record* data and immediately *see* emerging patterns. This approach was popularized by designers such as Lupi and Posavec [2], who advocate for hand-drawn visuals as a means of collecting, observing, and engaging with personal data. Such expressive representations have also been proposed as a way to popularize data sciences, including among children [4]. More broadly, drawing can be seen as a cognitive tool that makes the invisible contents of mental life visible [5].

Throughout this paper, we use the term *hand-drawn infographics* to emphasize the creative nature of these representations, which often function as a form of *personal visualizations* [6]. Figure 1b illustrates a typical hand-drawn infographic,[1] where small geometric figures, or *glyphs* [7], represent quantities of interest. Each glyph is easy to draw and encodes several data dimensions via variations in shape, color, size, or other visual parameters. This technique of using glyphs to visually represent multivariate data can cover many scenarios and is simple enough for non-professionals to use (see Figure 7). However, since such infographics are drawn by hand, the corresponding data often does not exist in digital form, making it difficult to edit, modify, reuse, and expand the visualization.

In this paper, we propose a method to recover data from hand-drawn infographics composed of individual glyphs. Our goal is to address scenarios where users do not manually enter the parameter values of each glyph. The first scenario is *reverse-engineering* hand-drawn visualizations containing tens to hundreds of glyphs, where we allow users to later edit the extracted data or the glyphs themselves (Figure 1d, e). The second

Email address: anran.qi@inria.fr (Anran Qi)

[1] Bahareh Heravi, Information Design at UCD.



(a) Parametric glyph template (b) Hand-drawn infographic (c) Recovered data table (d) Updated infographic after editing the data table (e) Recolored infographic

Figure 1: Sketch2Data is a reverse engineering system that recovers data from glyph-based hand-drawn infographics. In this example, the glyphs represent personal compliments received or given by an individual. The orientation of the triangle depicts the hour of the compliment (12-3, 3-6, 6-9, 9-12), and one or two lines on its side show whether the compliment occurred in the morning (am) or the afternoon (pm). The disks or suns in the triangle's corners indicate whether the compliment was received or given, while their number conveys the strength of the compliment. Finally, the color denotes the person who gave or received the compliment (e.g., a friend, family member, or colleague), and a dot or cross signifies whether the compliment was delivered in person or via text/email. Sketch2Data pipeline works as follows. The user defines a parametric glyph template (a), which maps the visual parameters of the glyph (P1, P2, ..., P5) to categorical data dimensions (see text for an explanation of the meaning of these dimensions). Sketch2Data recognizes the parameters of hand-drawn glyphs (b) and infers the underlying data table (c). The system supports a bi-directional workflow, where users can generate new instances of their hand-drawn infographics: by editing the data table (d), such as changing the values mapped to P4 for some entries (e.g., from ☼ to ☼☼) or by editing the parametric glyph template (e), such as changing P5 to a new color theme. The infographic is borrowed from B. Heravi's student collection ("week of compliments").

scenario, inspired by recent work in information visualization [8, 9], is casual data recording, where people quickly draw glyphs on a notebook or a simple drawing app in situ and later convert them into a data table.

In contrast to traditional charts and graphs, which are the focus of existing reverse-engineering tools [10, 11], hand-drawn glyphs are expressive depictions of data that exhibit significant diversity. Given this diversity, our strategy is to enable users to create a *specialized* machine learning model overfitting the glyph type they seek to analyze. We do not aim to develop a general, pre-trained model that supports unseen glyphs because our preliminary experiments with an open-set detector [12] reveals that it fails to detect diverse glyphs accurately even when provided with several exemplars, as illustrated in Figure 21 and 22 in the supplemental document.

We observe that many glyphs can be represented as compound objects composed of a few graphical elements, called *marks*, where each mark admits a finite set of variations across one or several visual dimensions, such as shape, color, or texture. These dimensions are mapped to the semantic meaning of the data. Hence, the first step of our method consists in letting users define a *parametric template* for the glyphs in a given hand-drawn visualization (Figure 1a). As in prior systems supporting glyph design [13, 14, 15, 16], users specify the graphical variations of individual marks and define their relative positioning through alignment constraints on their bounding boxes. This template enables the automatic generation of all possible glyph variants, facilitating the synthesis of a training dataset tailored to the specific visualization the user seeks to reverse-engineer. Using this dataset, we train deep neural networks to locate real instances of a glyph within the visualization and automatically determine their visual parameter values, thereby recovering their meaning. With this strategy, users can apply our method to new glyph types simply by creating new glyph templates.

Despite their ability to recognize complex visual patterns, neural networks may fail to predict parameters accurately when the glyph design or drawings are ambiguous (see discussion in Section 7). Our method assists the user in identifying such cases by quantifying the uncertainty of predictions and providing a simple interface for reviewing and correcting the results.

We have created two benchmarks to evaluate our approach. The first benchmark contains 10 hand-drawn visualizations collected from the internet or drawn by ourselves, and serves to evaluate the ability of our system to reverse-engineer diverse glyphs. The second benchmark contains three designs, each drawn by 12 different participants, and serves to evaluate the robustness of our method to different drawing styles. Our results show that Sketch2Data can handle a diverse set of visualizations, assist in reverse-engineering with little user effort, and allow the easy and effective creation of novel visualizations.

2. Related work

Drawing data representations. Sketching is an extremely flexible way of crafting data representations [17] and also serves as an effective tool for thinking and communication [18]. In addition, the hand-drawn nature of a visualization can enhance engagement, encourage annotation, and convey the uncertainty of the underlying data [19, 20]. Previous work has introduced a range of sketch-based systems for information visualization. Some systems use drawings to rapidly create charts and interact with them to tell stories based on data [21, 22]. Other work, inspired by the Dear Data project of Lupi and Posavec [1], focuses on how drawing can support personal and expressive data visualization. DataInk [13], in particular, enables users to draw artistic data glyphs, while DataSelfie [23] allows users to collect data from questionnaires and visualize the results using custom hand-drawn visuals. DataQuilt [24], in turn, supports hybrid glyphs that blend freehand drawing with visual elements extracted from images.

These systems require users to map existing data to the visual parameters of the glyphs to generate visualizations. In contrast, we address the inverse problem of recovering data from existing, hand-drawn visualizations. In the context of personal infographics, visualizations can serve as input substrates for data collection [8], which has motivated a separate stream of sketch-based systems that allow users to record their data by drawing on a canvas [15, 25, 9]. While these systems require users to draw in dedicated interfaces that impose specific steps in the visualization creation process, our goal is to recover data from visualizations created using a wide variety of drawing tools, including raster and vector-based software, as well as traditional mediums such as pencils, markers, and watercolors.

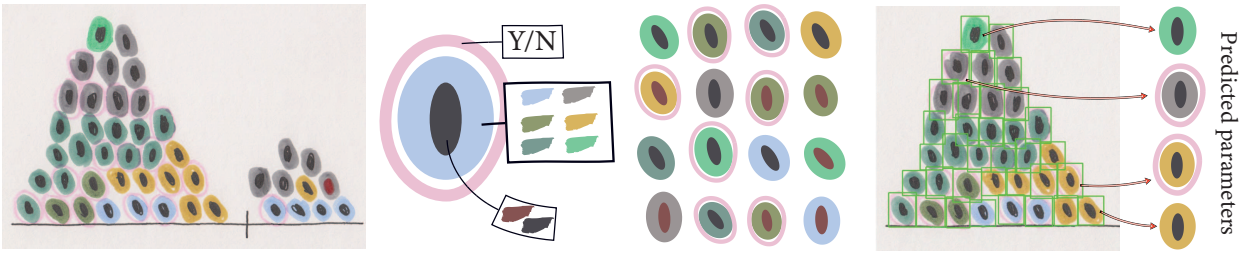
Reverse engineering data visualizations. A substantial body of work has focused on recovering data from existing visualizations. Most methods target graphs, charts and plots produced by visualization software, typically represented as clean bitmaps [10, 26, 27, 28, 29], vector graphics [30, 31], or even source code [32]. Other systems automatically extract templates from specific visualization types, such as timelines [33] and network graphs [34], and transfer them to new data. Rather than recovering data values from visuals, Lu et al. [35] focus on extracting the information flow of infographics.

Similarly to our work, most of these systems rely on machine learning to detect and classify the elements that compose a visualization. However, these machine learning modules are specialized to standardized visualizations and their constituent elements (bar charts, pie charts, scatter plots, graphs, etc.) and are not applicable to the hand-drawn glyphs we target. More closely related to our approach are systems for reverse-engineering hand-drawn graphs [11] and astronomical diagrams [36]. However, these approaches exploit domain-specific characteristics, such as the fact that graphs can be segmented into nodes and edges and astronomical diagrams are typically composed of lines, arcs and circles. In contrast, by enabling users to specialize deep neural networks for a target hand-drawn visualization, our method can recognize glyphs with significant variability in shape and style.

Sketch-based parametric design. By converting drawings to parametric objects, our work also relates to image vectorization. However, vectorization algorithms typically output low-level geometric primitives parameterized by numerous control points, such as polylines, Bézier curves, and color regions [37, 38, 39]. In contrast, we aim at recovering compound objects composed of several graphical elements whose variations are controlled by a small number of high-level parameters, such as in Figure 1b where one parameter controls whether a dot or a cross should be drawn. This makes our problem more closely related to sketch-based interfaces for procedural modeling [40, 41, 42, 43], where deep neural networks have been used to recognize pre-defined parametric shapes and predict their parameters. We follow a similar strategy and adapt it to the domain of data visualization. Specifically, since hand-drawn infographics contain multiple glyphs, we split the problem into two sub-tasks — glyph *localization* and glyph *parameter estimation*. Furthermore, since we aim at supporting diverse, expressive visualizations, we let users compose their own glyphs by specifying their constituent graphical elements and alignments. We then use synthetic drawings to train custom neural networks to localize and estimate the parameters of these glyphs.

3. Overview

Figure 2 illustrates the main components of our method for recovering data from hand-drawn visualizations. Given an input visualization, users first define a parametric template that captures all possible variations of the glyphs it contains (Figure 2b). We describe a simple procedure for this task, where the users specify each mark (graphical element) individually and define alignment constraints to assemble them. The resulting template allows us to generate a large dataset of synthetic



(a) Hand-drawn infographic (b) Parametric glyph template (c) Synthesized glyph dataset (d) Detected glyphs in green boxes (e)

Figure 2: Overview of our method: given an input hand-drawn infographic (a), the user defines a parametric glyph template by specifying all elements that compose the glyph and their visual variations (b). Based on this template, we synthesize an annotated dataset of glyph drawings (c) and use it to train a glyph detector (d) and a parameter predictor (e). A simple interface (see accompanying video) allows users to review the recovered data, make corrections, and even refine the template to achieve higher accuracy.

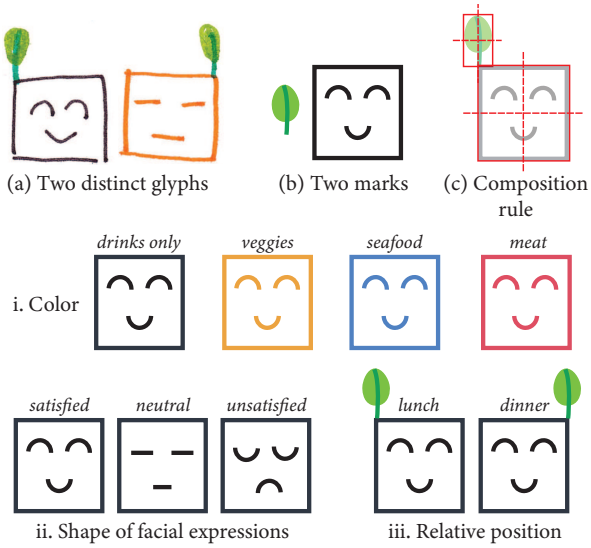


Figure 3: (a) Two examples of a hand drawn parametric glyph. (b) The glyph's template is defined using two marks (graphical elements): a leaf and a face. (c) The glyph is created by anchoring the bottom of the leaf to the top border of the face (either to the left or right). (d) Each visual parameter of a mark (color and shape of the face, position of the leaf) is mapped to a data dimension.

drawings of the specific glyph in the visualization (Figure 2c), where each drawing is associated with ground truth parameter values. We use this dataset to train two neural networks: one for localizing glyph instances in the visualization (Figure 2d) and another for estimating their parameter values (Figure 2e). To help users identify potential ambiguities in the template, our method reports uncertainty in its predictions. It also provides a user interface for comparing each glyph to its reconstructed representation and correcting potential errors (see examples in accompanying video).

4. Parametric glyph templates

In data visualization, glyphs are visual objects that depict data dimensions by varying in position, shape, color, texture, or other visual attributes [7]. A glyph can be compound, grouping multiple visual elements together, which are often referred to as *marks* [44]. A number of visualization authoring tools [13, 14, 15, 16] provide interfaces for creating custom parametric glyphs. Building on these works, we define a glyph as a group of graphical elements, or *marks*, organized using simple composition rules. Each mark can vary according to one or several *visual parameters* that encode a data dimension. We focus in our work on categorical data, where the visual parameters are limited to a finite set of values, and our goal is to recover these values for each drawing of a glyph.

Figure 3a illustrates two variations of a typical hand-drawn glyph, which, in our example, captures a client's experience in a restaurant. This glyph is made of two marks, the squared face and the leaf (Figure 3b), composed by anchoring the bottom-center of the leaf to the top left or right corner of the face (Figure 3c). This glyph can vary according to three visual parameters, where each parameter is mapped (bound) to a categorical data dimension (see Figure 3d for details). Our system concentrates on visual recognition and outputs a data table where each visual parameter can take different values, as defined by the glyph template. Users are free to map these values to any semantic meaning, such as mapping the four possible values of the color parameter to four different types of meal in Figure 3.

We refer to the parametric definition of a glyph — i.e., the definition of its marks, and their variations and composition — as the *glyph template*. In our prototype, the user first creates all possible variations of each mark using a vector drawing software and then specifies how

marks are composed together. We fix the relative position of two marks through a parent-child relationship, where we anchor a reference point of the child mark (e.g., the center, top or bottom of its bounding box) to a reference point of the parent mark. In Figure 3c, for example, the bottom-center point of the leaf (child) is anchored to the top-left point of the face (parent). Once defined, the template allows the automatic generation of all possible variations of the glyph. We provide a visual summary of all parametric templates used in our results in Figure 4 in the supplemental document.

5. Recognizing hand-drawn glyphs

While hand-drawn glyphs often depict a small number of data dimensions, each admitting a finite set of values, combinatorial explosion quickly yields a large number of glyph variations even in simple cases (the illustrative example in Figure 3 covers 24 variations, while the more complex examples in Figure 7 include hundreds of variations). This complicates a direct lookup comparison between each hand-drawn glyph and all possible variations of its template. In addition, a direct comparison would require an image metric robust to misalignment and other defects inherent to hand drawings. Instead, we tackle this challenge by adopting a machine learning approach to glyph recognition, using deep neural networks to both locate glyphs in an image and to estimate the parameter values of each glyph instance.

5.1. Synthesizing training data

Training neural networks to detect and recognize a specific type of glyph requires a dataset that covers all possible variations of that glyph, including common defects found in hand-drawn examples. Since drawing all these images by hand would be impractical, we generate synthetic drawings to obtain sufficient data.

The parametric template described in Section 4 allows us to generate all variations of the glyph. We augment this data by generating drawings subject to global and local deformations that frequently occur in real drawings. We adjusted the range of these deformations such that the synthesized glyphs visually resemble the original designs. Specifically, global deformations include random rotation (within $[-1, 1]$ degrees) and random scaling (within $[0.9, 1.1]$ horizontally and vertically), except for designs where the orientation or scale of the glyphs varies significantly. In such cases, we adjust the deformation ranges accordingly, as with designs (1) and (3) in Figure 7). Local deformations include random rotation, translation and scaling of individual

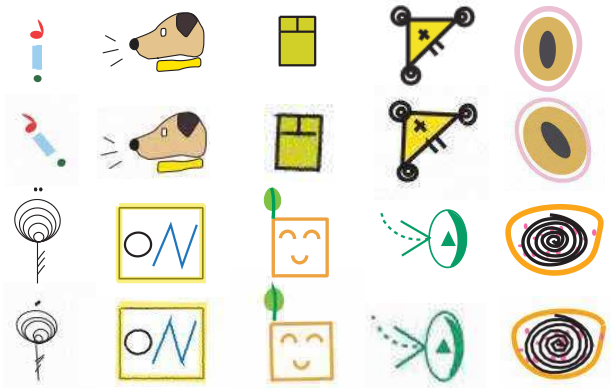


Figure 4: We augment our training data by applying various deformations as well as image noise and compression on the synthetic glyphs to mimic hand-drawn ones. For each example, we show the original synthetic glyph (top) and its deformed version (bottom).

lines that compose the graphical marks, by an amount that users can adjust depending on variations observed in the input drawings. Furthermore, we mimic drawing inaccuracies by shortening or extending the lines by a random amount, and applying random shifts to points regularly placed along the lines. Finally, we apply JPEG compression with a mean quality of $75(\pm 20)$ and Gaussian image noise with a mean of $0(\pm 3)$ to simulate imaging artifacts. Figure 4 illustrates representative glyphs before and after deformations. Table 2 in the supplemental document provides an ablation study showing that this data augmentation increases the average accuracy of parameter estimation from 0.83 to 0.94 on our benchmark. Note however that additional data augmentation might be needed for glyphs that exhibit more variation than the ones we experimented with, for instance using the rendering library proposed by Wood et al. [19].

For the localization dataset, we generate $2,000$ images of randomly placed glyphs, ensuring that glyphs do not overlap. We adjust the number of glyphs within each image to approximately match the number of original glyphs in the input visualization. For the parameter estimation dataset, we generate between $10,000$ and $20,000$ images of individual glyphs, depending on the total number of parameter combinations that the glyph template admits.

5.2. Glyph detection

The task of detecting and localizing glyphs in the input visualization relates to object detection, with two specific difficulties. First, hand-drawn glyphs are abstract graphical objects for which no pre-trained model exists.

Second, the glyph detector must be robust to the significant visual variations encoded by the glyph, which are further combined with pen stroke variations inherent to hand-drawing. We address these difficulties by training a custom object detector for each glyph template, leveraging the synthetic drawings in our training data.

We rely on the Yolo-v8n architecture [45], a state-of-the-art object detector that outputs labeled axis-aligned or object-oriented bounding boxes of detected objects. We initialize the model with weights pre-trained on the COCO segmentation dataset [46], and fine-tune on our localization dataset. Note that we fine-tune the model for each glyph template, such that the model specializes in that glyph. We train the model to output axis-aligned bounding boxes, except for cases that contain glyphs in different orientations, such as design (3) in Figure 7. Fine-tuning the object detector only takes 10 minutes on average on a single NVIDIA RTX A6000 GPU with 48GB memory.

5.3. Estimating parameters from cropped glyphs

Once a glyph has been detected, the next step consists in recovering the value of its visual parameters. Since we target categorical data, we cast parameter estimation as a classification task, where each possible parameter value corresponds to one label. Extending to continuous parameters would require training the network to perform a regression task, as demonstrated by related work in procedural modeling [42], but we leave this for future work.

We build our parameter estimation network on the ResNet-18 architecture [47], initialized with weights pre-trained on the ImageNet dataset [48]. For each glyph template, we replace the final layer of the ResNet-18 architecture by N branches, where N is the number of parameters of a glyph. Each branch is formed of three fully-connected layers of dimensions 256, 128 and M respectively, where M is the number of values that the parameter can take. We train the resulting neural network on the synthesized dataset for that glyph, using a cross-entropy loss to measure agreement between the predicted parameter values and the ground truth. In practice, we freeze all parameters of the ResNet-18 backbone, except for the last ResNet block, which we fine-tune along with the branches we have added. Training this architecture takes approximately 5 minutes on average.

5.4. User quality refinement

Despite the high-level accuracy of the pipeline we have described (see Section 6), it remains prone to errors in the presence of drawing inaccuracies or ambiguities in the parameter values that need to be distinguished. Our method provides an efficient procedure for users to assess and correct errors in the generated results. We propose two mechanisms to support this. First, we report the uncertainty of the neural network's predictions, which can help identify difficult cases and diagnose ambiguities in the parametric glyph template. Second, we provide a user interface that allows users to review the predictions side-by-side with the original glyphs and correct any erroneous parameter values.

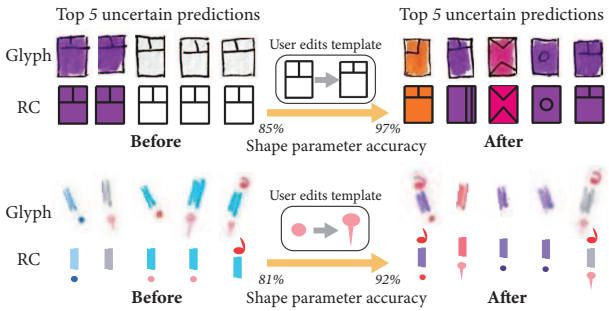
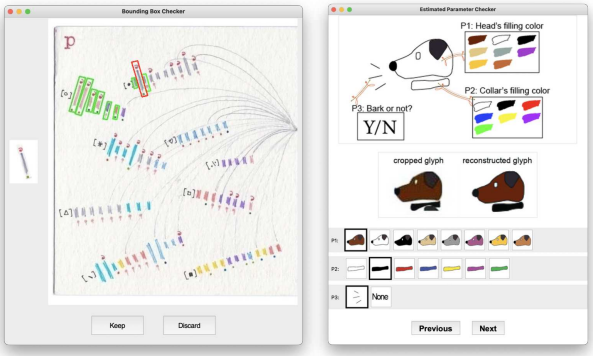


Figure 5: In these two examples, visualizing the top 5 most uncertain predictions of a parameter (RC) reveals that they correlate with specific values of that parameter (window-shaped glyph for the top example, pink circle for the bottom example), even though the values are actually predicted accurately in most cases. The user edits the parametric template to better depict these values, such that they do not yield high uncertainty anymore. These corrections also improve the accuracy of parameter estimation overall.

Uncertainty estimation. We compute uncertainty of the parameter estimation by running an ensemble of neural networks [49], each trained on a different subset of the dataset. In our implementation, we train 10 such networks on subsets of our training data by drawing random samples that cover 70% of the data (see Section 3 in the supplemental document for the influence of the number of networks in the ensemble). We express the uncertainty of a given parameter as the average pairwise distance between the predictions given by all networks, and keep the most frequent value as the final prediction. In the case of a tie, we keep the value with the highest average probability in the prediction vector.

Figure 5 illustrates two examples of using uncertainty to iterate on the parametric template of a glyph. In the first example, the top 5 most uncertain predictions of the glyph's first parameter correspond to the same parameter value, depicted as a rectangle crossed by a horizontal and a vertical bar. By inspecting the original glyphs depicting that value, the user notices that in the parametric template, the horizontal bar is lower than it should



(a) Glyph detector R&C UI (b) Glyph parameter R&C UI

Figure 6: Screenshots of our interface for reviewing and correcting glyph detection and parameter estimation. For glyph detection (a), the interface displays each detected bounding box one by one, allowing the user to either keep or discard it. For parameter estimation (b), the interface provides a visual summary of the parametric template (top), a comparison of the detected and reconstructed glyphs (center), and the estimated value of each glyph parameter (bottom). The user can choose to keep the estimated value or correct it by selecting the appropriate parameter value.

be. Similarly, in the second example, visualizing the top 5 most uncertain predictions reveals that three of them contain a pink circle with a spike, while the parametric template did not include that spike. This examination provides important information, as correcting the template and retraining the method fixes these issues and improves accuracy in both cases.

User interface for data review. Figure 6 shows two screenshots of the user interface we developed for data review and correction. We provide as Appendix a screen recording of this interface in action. For glyph detection and localization, the interface displays each detected bounding box and asks the user to either accept or reject it. Most glyphs are well detected, but false positives can occur in the presence of background strokes. Some glyphs can also be detected multiple times, yielding duplicated data if not rejected. For parameter estimation, the interface displays the detected glyphs one by one along with their visual reconstructions and the corresponding parameter values. This allows users to quickly scan the recovered data and correct erroneous values.

6. Evaluation and applications

In this section we evaluate the accuracy of our glyph detection and parameter estimation on two usage scenarios. The first scenario corresponds to the case where the user wants to recover data from an existing visualization. The second scenario corresponds to the case where the user wants to collect new data by drawing instances of a prescribed glyph template. We end this section by demonstrating various applications enabled by reverse-engineering hand-drawn infographics.

6.1. Reverse engineering existing infographics

We first evaluate whether our system can recover accurate data from diverse, existing hand-drawn infographics.

Evaluation protocol. We have created a benchmark of 10 hand-drawn infographics collected from diverse sources, including from data designers who advocate drawing for visualization, as well as from coursework of design schools. For copyright reasons, we redesigned 4 of these infographics, using the original ones as visual inspiration. We also include one image created with a previous sketch-based data visualization system [13]. Figure 7 provides an overview of this benchmark.

These infographics are composed of diverse glyphs that depict data dimensions through variations in shape, color, orientation, location, size and texture, and were drawn with pencils, markers, watercolor, and digital pen. The number of glyphs within each image varies from 23 to 315 (73 on average), the number of glyph parameters from 2 to 5, and the number of values per parameter from 2 to 16. Refer to the supplemental document for detailed statistics.

For each image, we have used our system and its correction interface (Subsection 5.4) to annotate ground truth parameter values for each detected glyph.

Results. Table 1 reports the accuracy of our glyph detector for each of the 10 infographics in our benchmark, and we provide visualization of the detected bounding boxes in Figure 5 to 20 in the supplemental document. The detector is very accurate overall, the most challenging case being the *Boyfriend* visualization that contains many small glyphs, as well as background curves and annotations that do not appear in the synthetic training data.

Furthermore, Table 2 reports the accuracy of the parameter estimation for each parameter of each type of glyph, along with the Pearson correlation between parameter accuracy and neural network uncertainty. The accuracy is high overall (0.94 on average) and exhibits a negative correlation with uncertainty for most parameters (-0.4 on average), which indicates that reporting uncertainty can help users identify erroneous cases. Figure 8 shows typical results for each type of glyph, and

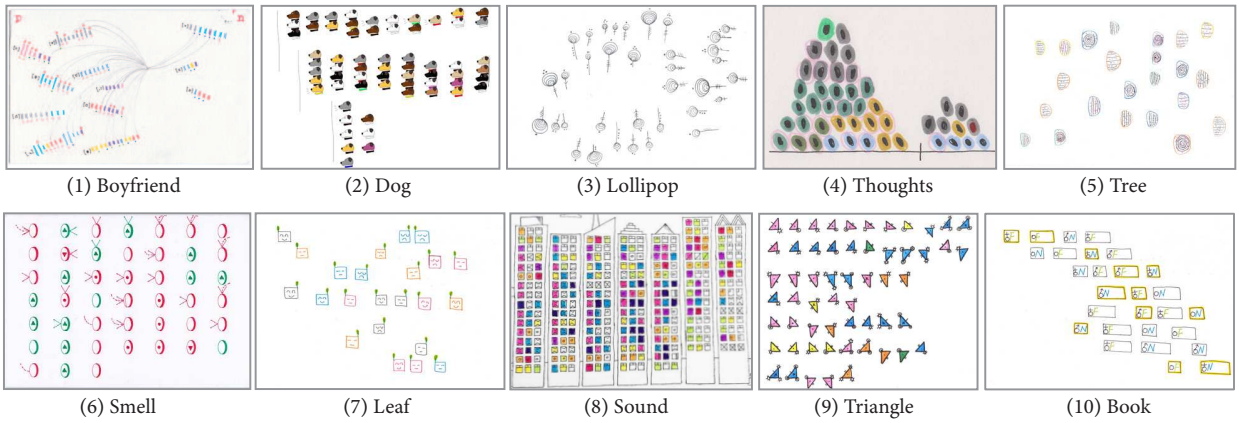


Figure 7: Our benchmark of ten hand-drawn infographics. (1) Boyfriend: Dear Data Week 29 ©Giorgia Lupi. (2) Dog: DataInk ©Haijun Xia. (3) Lollipop: redesigned, inspired by Dear Data Week 36. (4) Thoughts: Dear Data Week 38 ©Stefanie Posavec, (5) Tree: redesigned, inspired by course work on data visualization from Université Paris Nanterre. (6) Smell: Dear Data2 Week 47 ©Andy Kriebel. (7) Leaf: redesigned, inspired by post on medium. (8) Sound: post on Facebook ©Anita Boeira, (9) Triangle: course work of Information Design, UCD iSchool, instructor: Bahareh Heravi ©Baker Kerrigan. (10) Book: redesigned, inspired by post on X.

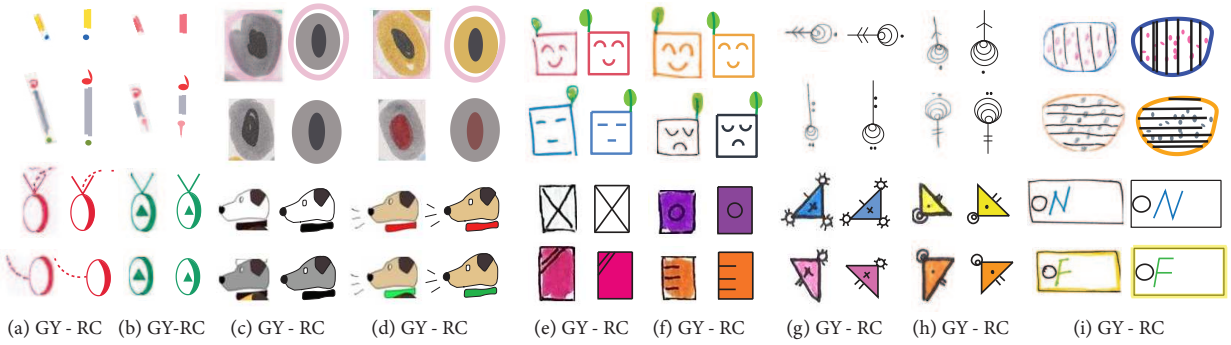
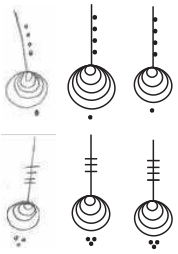


Figure 8: Examples of reconstructed glyphs. For each, we show the original glyph cropped from the input image (GY, left) and the one reconstructed from the estimated parameters (RC, right).

we provide full reconstruction of each visualization in Figure 5 to 20 in the supplemental document.

Three of the 10 infographics appear more challenging — *Lollipop*, *Sound*, and *Triangle* — we next discuss them in more details by showing examples of the input drawing (GY), the recovered glyph (RC) and ground truth glyph (GT).

Several parameters of the *Lollipop* are depicted by a varying number of elements forming a mark, such as the number of inscribed circles, or the number of dots or ticks along the line. As shown as inset, while the neural network distinguishes dots from ticks (top vs. bottom), it miscounts the number of circles (top) or ticks (bottom) as the corresponding visual patterns are very similar. These failure cases are consistent with previous studies on the performance of convolutional neural networks for graphical perception tasks [50], and would require more advanced architectures that include attention mechanisms for counting [51]. Furthermore, the *Lollipop* template admits 2160 different combinations of parameter values, making the estimation of precise parameter values much more challenging than for the other designs.



The *Triangle* glyph is illustrative of another challenging case, which is the use of small marks to depict data values. One parameter is depicted by the presence of a small dot or cross, and another parameter is depicted by the presence of one or two small ticks on the side. When drawn by hand, such small details are difficult to distinguish, especially in low-resolution images.

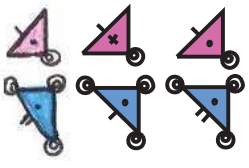
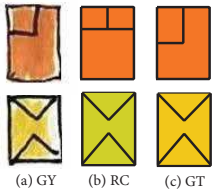


Table 1: Evaluation of glyph detector for our benchmark. For each design, we report the number of glyphs to be detected (Num), the number of true positive detection (TP), false positive detection (FP), and false negative detection (FN), from which we compute precision and recall. The *Boyfriend* design exhibits a large number of false positives due to the presence of background pen strokes.

Design	Num	TP	FP	FN	Precision	Recall
Book	28	28	1	0	0.97	1.00
Boyfriend	94	88	118	6	0.43	0.94
Dog	63	63	0	0	1.00	1.00
Leaf	23	23	0	0	1.00	1.00
Lollipop	35	35	0	0	1.00	1.00
Smell	45	45	0	0	1.00	1.00
Sound	315	315	19	0	0.95	1.00
Thoughts	45	45	0	0	1.00	1.00
Tree	23	23	0	0	1.00	1.00
Triangle	57	57	1	0	0.98	1.00
Avg.					0.93	0.99

The *Sound* glyph encodes only two parameters, but each parameter can take around 10 different values, making the difference between some of the values subtle, especially when hand-drawn. Such subtle variations can be difficult to perceive, even for humans. In the inset, the glyph template shows subtle variations in shape (top) and color (bottom), which confuses parameter estimation.



6.2. Glyphs drawn by different people

We conducted a study to evaluate the second usage scenario where users record data by drawing glyphs according to a prescribed template. We have collected drawings of the same glyphs from multiple participants and evaluate the sensitivity of our approach to variations in individual drawing styles.

Participants. We recruited 12 volunteers from two different research laboratories (six from each). The participants were local Ph.D. students or researchers. All participants signed a consent form and agreed that their drawings would be distributed under a CC0 license. The study was approved by the ethical committee of our institution, approval code Inria COERLE 2024-54.

Protocol. After a brief training task using the *Sound* example (see Figure 7), we asked participants to draw infographics following three glyph templates: *Smell*, *Leaf*, and *Book*. For each template, we introduced a hypothetical data-collection scenario and provided participants with three A4 sheets: (i) a data sheet containing data values that they should encode visually in their glyphs, (ii) a template specification sheet outlining the mappings between data values and glyph visual parameters, and (iii) a drawing sheet. In addition, we provided marker pens of different colors and specified which colors to use. For each scenario, participants were asked to draw 10 different glyphs (30 glyphs in total), with each glyph corresponding to a unique row in the data table. Data values were randomly generated, and each pair of participants (one from each laboratory) tested a different combination of values.

Participants were free to draw the glyphs and their individual strokes in any order. Some participants drew the glyphs sequentially (i.e., working row by row in the data table), while others employed alternative strategies, such as drawing one visual parameter at a time (i.e., column by column) or drawing with one color pen at a time. We instructed participants to scale their drawings similarly to the 2-3 examples printed at the top of their drawing sheet. We re-scaled their drawings if they significantly violated this instruction.

Results. We applied our approach to scans of the participants' drawings (using a TOSHIBA e-STUDIO3515AC printer at 600 dpi in JPEG format). Figure 9 presents examples of glyphs drawn by 6 of the participants, illustrating the variation in their drawing styles. Despite this variation, our glyph detector and parameter estimator performed well, with an average precision of 0.99, recall of 0.99, and parameter estimation accuracy of 0.96 (see Table 4 to 6 in the supplemental document). This demonstrates that our system adapts to diverse drawing styles.

The *Leaf* design achieved near-perfect scores across all participants, with an average parameter estimation accuracy of 0.99. Only two parameters were incorrectly estimated from the drawings of Participant 11 (inset), due to the ambiguous depiction of facial expressions. The *Smell* design resulted in more errors, with an average accuracy of 0.90, as it requires distinguishing small details, such as triangles pointing up or down versus a disk. Participant 10's drawings exhibited the lowest performance across all metrics — precision (0.87), recall (0.93), and parameter estimation accuracy (0.87). This low performance aligns with our estimation of uncertainty, which is higher for this participant than for the others on average. As shown in Figure 9, this participant struggled to respect the spatial relationships and relative sizes of the graphical elements that form the glyphs.

Table 2: Estimation accuracy for each parameter (P) and its correlation (Pearson correlation coefficient ranging from -1 to 1) with uncertainty (C). Note that correlation is undefined (N/A) for parameters with perfect accuracy.

Design	P1	C1	P2	C2	P3	C3	P4	C4	P5	C5	Avg. P	Avg. C
Book	1.00	N/A	1.00	N/A	1.00	N/A	1.00	N/A			1.00	N/A
Boyfriend	1.00	N/A	1.00	N/A	0.97	-0.43	1.00	N/A			0.99	-0.42
Dog	1.00	N/A	0.94	-0.61	1.00	N/A					0.98	-0.61
Leaf	1.00	N/A	1.00	N/A	1.00	N/A					1.00	N/A
Lollipop	0.71	-0.48	0.71	-0.19	0.74	0.10	0.46	-0.23	1.00	N/A	0.73	-0.20
Smell	1.00	N/A	1.00	N/A	1.00	N/A	1.00	-0.46			1.00	-0.46
Sound	0.92	-0.24	0.88	-0.41							0.90	-0.33
Thoughts	1.00	N/A	0.98	-0.65	0.91	-0.35					0.96	-0.50
Tree	0.91	-0.34	1.00	N/A	1.00	N/A					0.97	-0.34
Triangle	1.00	N/A	0.70	-0.32	0.82	-0.30	0.93	-0.58	1.00	N/A	0.89	-0.40
Avg.	—	—	—	—	—	—	—	—	—	—	0.94	-0.41

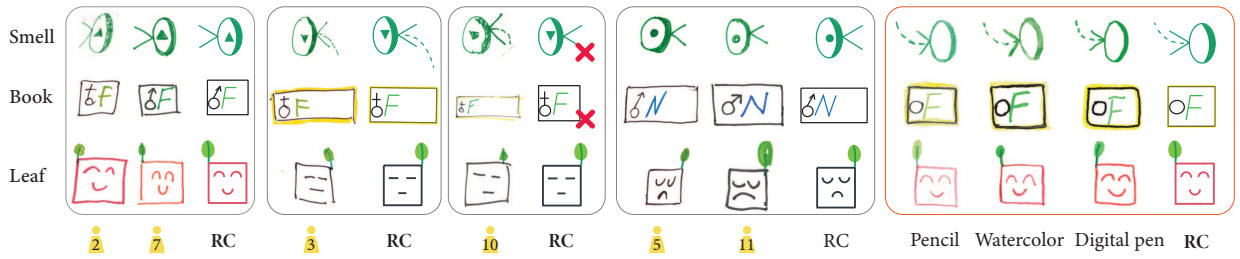


Figure 9: The first four boxes show glyphs drawn by six participants (N, where N is the participant's number) alongside their corresponding reconstructions (RC). The last box displays glyphs created using pencil, watercolor, and a digital pen, along with their reconstructions (RC). Red crosses indicate erroneous estimations for two glyphs of Participant 10, where our reconstruction misses the arc of the arrow (top) or is not elongated and not framed in yellow (center).

6.3. Glyphs drawn with different materials

To evaluate the robustness to different drawing materials, the first author of the paper replicated the task and created three additional infographics from the same three pre-defined glyph templates using three different drawing tools (10 drawings each): pencil, watercolor, and digital pen.

Results. We provide, in Table 8 in the supplemental document, detailed statistics on the accuracy of our glyph detector and parameter estimation for each of the three infographics and each drawing tool. These materials also include visualizations of the detected bounding boxes and the reconstructed glyphs based on the estimated parameters. The detector is very accurate, with only two false positives in the *Smell* design. The parameter estimation is also nearly perfect, despite the visual differences introduced by each tool, as illustrated in Figure 9 (right).

6.4. Editing hand-drawn infographics

We illustrate various applications enabled by our ability to recover data from hand-drawn infographics.

Visualization editing. Figure 10 shows how our method can be used to turn a set of hand-drawn glyphs into clean vector graphics (a → b) that can be further edited and enriched in any graphics design tool. For example, by modifying the shape (a → d), or the color (a → e) of the glyph template, users can quickly experiment with variants of the visualization. Similarly, editing the data table itself allows users to visualize partially updated data values while preserving unchanged ones (a → c). Finally, users can easily explore alternative glyph layouts (a → f).

Figure 11 showcases a more advanced editing scenario where a hand-drawn infographic is first reversed-engineered using our method (a → b) and then regenerated with different glyph templates (c), resulting in new visualizations of the same data in drastically different styles (d, e, f).

Style transfer. While vector graphics offer a clean and precise look, users may sometimes prefer to preserve the original hand-drawn aesthetics of their infographics. To achieve this, we used the glyphs reconstructed by our method, along with their original hand-drawn representation, to train a Pix2Pix model [52] capable of transfer-

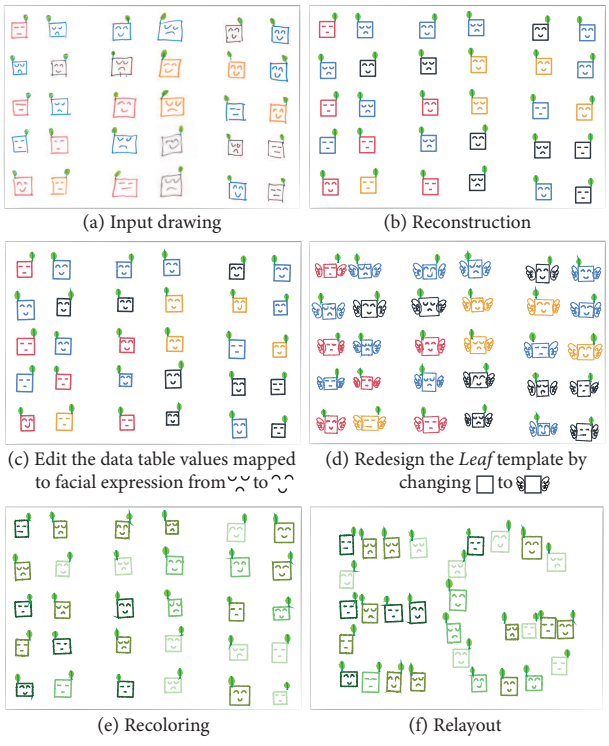


Figure 10: Reverse engineering the input hand-drawn illustration (a) drawn by Participants 1, 2, and 3 to a clean vector infographic (b). This allows various operations such as editing the data table itself (c), redesigning (d) and recoloring (e) the marks of Leaf template and creating a new layout for the glyphs (f).

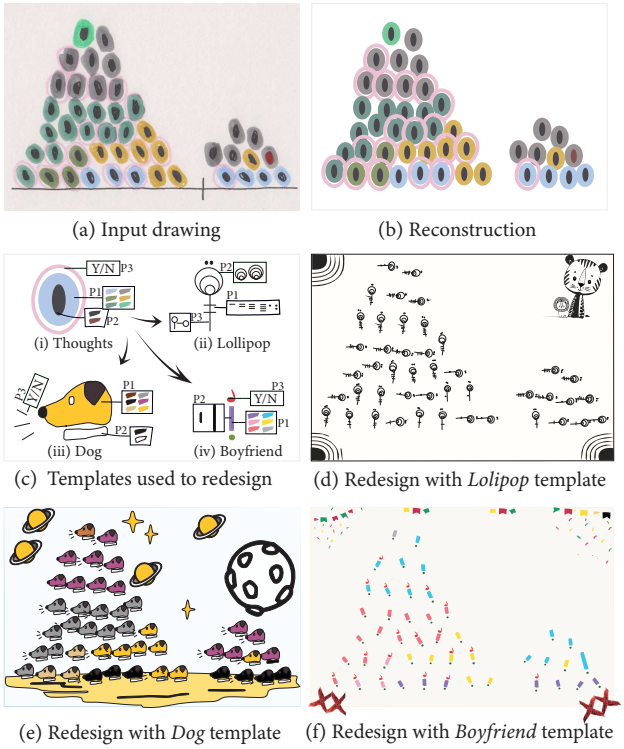


Figure 11: Reverse engineering the original hand-drawn illustration of Thoughts (a) into a clean vector infographic (b) allows to visualize the same data using different templates (c). In this example using Lollipop template (d), Dog template (e), and Boyfriend template (f).

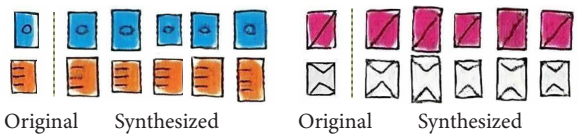


Figure 12: Using style transfer [52], we can synthesize glyph variants that retain the hand-drawn style of the original image.

ring the hand-drawn style onto synthesized glyphs.

We demonstrate this application on the *Sound* design from Figure 7 because it contains a large number of glyphs that can serve as training samples. The design consists of 315 glyphs with two parameters, creating a parametric space of 11 shapes and 10 colors. While there are 110 possible glyph variations, only 47 appear in the original design, with varying frequencies. To balance the dataset, we ensured that each shape appeared 50 times in as many colors as possible. This was achieved by duplicating instances of under-represented shapes and removing instances of over-represented shapes. Furthermore, each hand-drawn glyph was aligned with its reconstructed version based on their respective bounding boxes.

Figure 12 shows four original glyphs alongside multiple variants synthesized with this approach. The parametric template dictates the overall shape and color of the glyph, while style transfer gives it a hand-drawn look. Figure 13 illustrates how this approach generalizes to unseen parameter combinations, with the 47 original glyphs highlighted by dashed frames. The approach enables us to synthesize all 110 possible glyph variants. However, style transfer requires a sufficient number of hand-drawn glyph examples to be trained, and as such cannot be applied to small designs that only contain a few glyphs. An interesting direction for future work would be to use the stylized glyphs as additional training samples to refine our model.

7. Conclusion and discussion

The act of drawing has long been advocated as a means for individuals to closely observe objects of interest, whether the human figure, landscapes, or buildings [53, 54]. Hand-drawn infographics play a similar role for data: by drawing glyphs, people can visually represent quantities that matter to them, record their val-

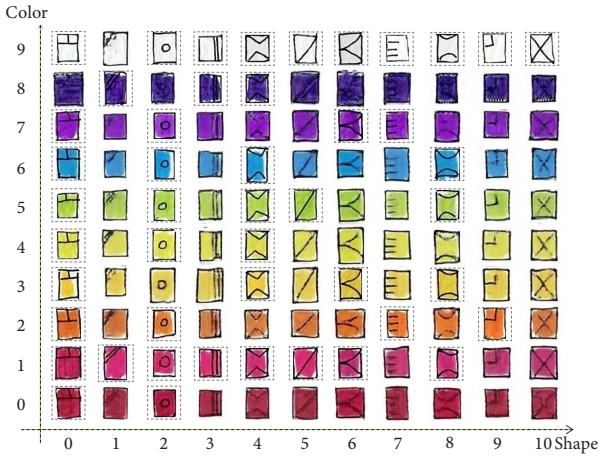


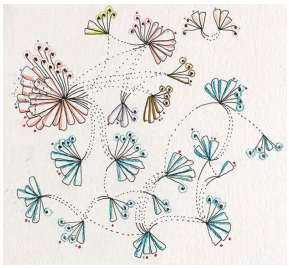
Figure 13: Although only 47 shape and color combinations were present in the original design (highlighted with dashed frames), we can use style transfer to synthesize the remaining 63 combinations. Note that the synthesis is more precise for shapes that are well-represented in the original data (such as shapes 2 and 4), compared to shape 10, which appeared only in black and white.

ues and explore their trends [2]. We have introduced a method to recover data from hand-drawn infographics, creating a bridge between expressive data representations and their digital forms.

Our approach leverages deep neural networks to detect glyphs and estimate the values they represent, despite significant variations in how they are drawn. We have also developed a user interface that helps identify and correct spurious errors in this estimation, allowing users to quickly extract accurate data from infographics containing up to several hundred glyphs.

Most of the errors we observed arise from ambiguities in the glyphs themselves, such as the use of similar colors and shapes to represent different data values, or the use of very small graphical elements. Such cases typically lead to high uncertainty in predictions (Figure 5). In this work, we refine the template design through a user-in-the-loop process guided by uncertainty and observe improvements in recognition accuracy. More formal investigations on template quality and glyph ambiguity could help inform designers about the readability of their glyphs. Other sources of error include the presence of graphical elements other than glyphs, such as background curves in the *Boyfriend* design (Figure 7, (1)). These issues could be addressed either by performing background removal before running glyph detection or by incorporating background distractors into the synthetic training dataset.

Our work also leaves challenging open problems for future work. We already discussed the possibility to extend our method to continuous parameter values by using regression instead of classification. Our implementation of glyph templates could also be extended to support additional geometric relationships between marks, for instance a radial arrangement of the glyphs shown in inset.[2] In addition, while we focused on recognizing individual glyphs, the layout of the glyphs can conveys additional information, such as a creating graph structure as in the inset. Existing work on inverse procedural modeling could aid in recovering distributional [55] or hierarchical [56] layouts. Going further, machine learning methods could be used not only to recover the parameter values of individual glyphs but also to construct the parametric glyph template that encapsulates all observed glyph instances. Recent work on neuro-symbolic programming has demonstrated automatic recovery of graphical structures from images and 3D shapes [57, 58], and their abstraction into parametric programs [59]. However, these methods are typically trained with far more examples than those available in a single hand-drawn visualization. Combining these methods with user interaction presents a promising direction to support glyph authoring from a small set of glyph instances.



References

[1] Lupi, G, Posavec, S. Dear Data. Princeton Architectural Press; 2016. https://www.dear-data.com/by-week.

[2] Lupi, G, Posavec, S. Observe, Collect, Draw!: A Visual Journal : Discover the Patterns in Your Everyday Life. Princeton Architectural Press; 2018.

[3] Quillin, K, Thomas, S. Drawing-to-learn: a framework for using drawings to promote model-based reasoning in biology. CBE—Life Sciences Education 2015;14(1).

[4] Khan, G. Drawing Data with Kids: Cultivating Data-Literacy: A Screen-Free Journey Through the Art of Visualization for Kids. Gulrez Khan; 2023.

[5] Fan, JE, Bainbridge, WA, Chamberlain, R, Wammes, JD. Drawing as a versatile cognitive tool. Nature Reviews Psychology 2023;2(9).

[6] Huang, D, Tory, M, Aseniero, BA, Bartram, L, Bateman, S, Carpendale, S, et al. Personal visualization and personal visual analytics. IEEE Transactions on Visualization and Computer Graphics 2015;21(3). doi:10.1109/TVCG.2014.2359887.

[7] Borgo, R, Kehrer, J, Chung, DHS, Maguire, EJ, Laramee, RS, Hauser, H, et al. Glyph-based visualization: Foundations, design guidelines, techniques and applications. In: Eurographics State of the Art Reports. 2013,doi:10.2312/conf/EG2013/stars/039-063.

[2] Dear Data Week 42 ©Giorgia Lupi

[8] Bressa, N, Louis, J, Willett, W, Huron, S. Input visualization: Collecting and modifying data with visual representations. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM; 2024,doi:10.1145/3613904.3642808.

[9] Offenwanger, A, Tsandilas, T, Chevalier, F. Datagarden: Formalizing personal sketches into structured visualization templates. IEEE Transactions on Visualization and Computer Graphics 2025;31(1):1268–1278. doi:10.1109/TVCG.2024.3456336.

[10] Poco, J, Heer, J. Reverse-engineering visualizations: Recovering visual encodings from chart images. Computer Graphics Forum 2017;36(3). doi:10.1111/cgf.13193.

[11] Song, S, Li, C, Sun, Y, Wang, C. Vividgraph: Learning to extract and redesign network graphs from visualization images. IEEE Transactions on Visualization and Computer Graphics 2023;29(7). doi:10.1109/TVCG.2022.3153514.

[12] Jiang, Q, Li, F, Zeng, Z, Ren, T, Liu, S, Zhang, L. T-rex2: Towards generic object detection via text-visual prompt synergy. In: European Conference on Computer vision. Springer; 2024,.

[13] Xia, H, Henry Riche, N, Chevalier, F, De Araujo, B, Wigdor, D. Dataink: Direct and creative data-oriented drawing. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM; 2018,doi:10.1145/3173574.3173797.

[14] Ren, D, Lee, B, Brehmer, M. Charticulator: Interactive construction of bespoke chart layouts. IEEE Transactions on Visualization and Computer Graphics 2019;25(1). doi:10.1109/TVCG.2018.2865158.

[15] Tsandilas, T. StructGraphics: Flexible Visualization Design through Data-Agnostic and Reusable Graphical Structures. IEEE Transactions on Visualization and Computer Graphics 2021;27(2):315–325. doi:10.1109/TVCG.2020.3030476.

[16] Brehmer, M, Kosara, R, Hull, C. Generative design inspiration for glyphs with diatoms. IEEE Transactions on Visualization and Computer Graphics 2022;28(01). doi:10.1109/TVCG.2021.3114792.

[17] Walny, J, Huron, S, Carpendale, S. An exploratory study of data sketching for visual representation. Computer Graphics Forum 2015;34(3). doi:https://doi.org/10.1111/cgf.12635.

[18] Wang, Z, Ritchie, J, Zhou, J, Chevalier, F, Bach, B. Data comics for reporting controlled user studies in human-computer interaction. IEEE Transactions on Visualization and Computer Graphics 2021;27(2). doi:10.1109/TVCG.2020.3030433.

[19] Wood, J, Isenberg, P, Isenberg, T, Dykes, J, Boukhelifa, N, Slingsby, A. Sketchy rendering for information visualization. IEEE Transactions on Visualization and Computer Graphics 2012;18(12). doi:10.1109/TVCG.2012.262.

[20] McNamara, A. Data visualization by hand: drawing data for your next story. Data Journalism 2021;URL: https://datajournalism.com/read/longreads/data-visualisation-by-hand.

[21] Lee, B, Kazi, RH, Smith, G. SketchStory: Telling more engaging stories with data through freeform sketching. IEEE Transactions on Visualization and Computer Graphics 2013;19(12). doi:10.1109/TVCG.2013.191.

[22] Lee, B, Smith, G, Riche, NH, Karlson, A, Carpendale, S. Sketchinsight: Natural data exploration on interactive whiteboards leveraging pen and touch interaction. In: Pacific Visualization Symposium. IEEE; 2015,doi:10.1109/PACIFICVIS.2015.7156378.

[23] Kim, NW, Im, H, Henry Riche, N, Wang, A, Gajos, K, Pfister, H. Dataselfie: Empowering people to design personalized visuals to represent their data. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM; 2019,doi:10.1145/3290605.3300309.

[24] Zhang, JE, Sultanum, N, Bezerianos, A, Chevalier, F. Dataquilt: Extracting visual elements from images to craft pictorial visualizations. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM; 2020,doi:10.1145/3313831.3376172.

[25] Offenwanger, A, Brehmer, M, Chevalier, F, Tsandilas, T. Timesplines: Sketch-based authoring of flexible and idiosyncratic timelines. IEEE Transactions on Visualization and Computer Graphics 2024;30(1). doi:10.1109/TVCG.2023.3326520.

[26] Savva, M, Kong, N, Chhajta, A, Fei-Fei, L, Agrawala, M, Heer, J. Revision: automated classification, analysis and redesign of chart images. In: Proc. Symposium on User Interface Software and Technology. ACM; 2011,doi:10.1145/2047196.2047247.

[27] Zhou, F, Zhao, Y, Chen, W, Tan, Y, Xu, Y, Chen, Y, et al. Reverse-engineering bar charts using neural networks. Journal of Visualization 2021;24(2). doi:10.1007/s12650-020-00702-6.

[28] Jung, D, Kim, W, Song, H, Hwang, Ji, Lee, B, Kim, B, et al. Chartsense: Interactive data extraction from chart images. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM; 2017,doi:10.1145/3025453.3025957.

[29] Song, S, Li, C, Li, D, Chen, J, Wang, C. Graphdecoder: Recovering diverse network graphs from visualization images via attention-aware learning. IEEE Transactions on Visualization and Computer Graphics 2024;30(7). doi:10.1109/TVCG.2022.3225554.

[30] Masson, D, Malacria, S, Vogel, D, Lank, E, Casiez, G. Chartdetective: Easy and accurate interactive data extraction from complex vector charts. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM. ISBN 9781450394215; 2023,doi:10.1145/3544548.3581113.

[31] Chen, C, Lee, B, Wang, Y, Chang, Y, Liu, Z. Mystique: Deconstructing svg charts for layout reuse. IEEE Transactions on Visualization and Computer Graphics 2024;30(1). doi:10.1109/TVCG.2023.3327354.

[32] Harper, J, Agrawala, M. Deconstructing and restyling d3 visualizations. In: Proc. Symposium on User Interface Software and Technology. ACM; 2014,doi:10.1145/2642918.2647411.

[33] Zhu-Tian, C, Wang, Y, Wang, Q, Wang, Y, Qu, H. Towards automated infographic design: Deep learning-based auto-extraction of extensible timeline. IEEE Transactions on Visualization and Computer Graphics 2020;26(1). doi:10.1109/TVCG.2019.2934810.

[34] Song, S, Zhang, Y, Lin, Y, Qu, H, Wang, C, Li, C. Gvvst: Image-driven style extraction from graph visualizations for visual style transfer. IEEE Transactions on Visualization and Computer Graphics 2024;doi:10.1109/TVCG.2024.3485701.

[35] Lu, M, Wang, C, Lanir, J, Zhao, N, Pfister, H, Cohen-Or, D, et al. Exploring visual information flows in infographics. In: Proc. CHI Conference on Human Factors in Computing Systems. ACM; 2020,doi:10.1145/3313831.3376263.

[36] Kalleli, S, Trigg, S, Albouy, S, Husson, M, Aubry, M. Historical astronomical diagrams decomposition in geometric primitives. In: Barney Smith, EH, Liwicki, M, Peng, L, editors. Document Analysis and Recognition - ICDAR. 2024,.

[37] Puhachov, I, Neveu, W, Chien, E, Bessmeltsev, M. Keypoint-driven line drawing vectorization via polyvector flow. ACM Transactions on Graphics 2021;40(6). doi:10.1145/3478513.3480529.

[38] Yan, C, Li, Y, Aneja, D, Fisher, M, Simo-Serra, E, Gingold, Y. Deep sketch vectorization via implicit surface extraction. ACM Transactions on Graphics 2024;43(4). doi:10.1145/3658197.

[39] Du, ZJ, Kang, LF, Tan, J, Gingold, Y, Xu, K. Image vectorization and editing via linear gradient layer decomposition. ACM Transactions on Graphics 2023;42(4). doi:10.1145/3592128.

[40] Manfredi, G, Capece, N, Erra, U, Gruosso, M. Treesketchnet: From sketch to 3d tree parameters generation. ACM Transactions on Intelligent Systems and Technology 2023;14(3). doi:10.1145/3579831.

[41] Huang, H, Kalogerakis, E, Yumer, E, Mech, R. Shape synthesis from sketches via procedural models and convolutional networks. IEEE Transactions on Visualization and Computer Graphics 2017;23(8). doi:10.1109/TVCG.2016.2597830.

[42] Nishida, G, Garcia-Dorado, I, Aliaga, DG, Benes, B, Bousseau, A. Interactive sketching of urban procedural models. ACM Transactions on Graphics 2016;35(4). doi:10.1145/2897824.2925951.

[43] Li, C, Pan, H, Bousseau, A, Mitra, NJ. Sketch2cad: Sequential cad modeling by sketching in context. ACM Transactions on Graphics 2020;39(6). doi:10.1145/3414685.3417807.

[44] Fuchs, J, Isenberg, P, Bezerianos, A, Keim, D. A systematic review of experimental studies on data glyphs. IEEE Transactions on Visualization and Computer Graphics 2017;23(7). doi:10.1109/TVCG.2016.2549018.

[45] Jocher, G, Chaurasia, A, Qiu, J. Ultralytics yolov8. 2023. URL: https://github.com/ultralytics/ultralytics.

[46] Lin, TY, Maire, M, Belongie, S, Hays, J, Perona, P, Ramanan, D, et al. Microsoft coco: Common objects in context. In: European Conference on Computer vision. Springer; 2014,.

[47] He, K, Zhang, X, Ren, S, Sun, J. Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition. IEEE; 2016, p. 770–778. doi:10.1109/CVPR.2016.90.

[48] Deng, J, Dong, W, Socher, R, Li, LJ, Li, K, Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In: Conference on Computer Vision and Pattern Recognition. IEEE; 2009,doi:10.1109/CVPR.2009.5206848.

[49] Lakshminarayanan, B, Pritzel, A, Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In: Guyon, I, Luxburg, UV, Bengio, S, Wallach, H, Fergus, R, Vishwanathan, S, et al., editors. Advances in Neural Information Processing Systems; vol. 30. 2017,.

[50] Haehn, D, Tompkin, J, Pfister, H. Evaluating 'graphical perception' with cnns. IEEE Transactions on Visualization and Computer Graphics 2018;25(1). doi:10.1109/TVCG.2018.2865138.

[51] Zhang, Y, Hare, J, Prügel-Bennett, A. Learning to count objects in natural images for visual question answering. In: International Conference on Learning Representations. 2018,.

[52] Zhu, JY, Park, T, Isola, P, Efros, AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: International Conference on Computer Vision. IEEE; 2017,doi:10.1109/ICCV.2017.244.

[53] Dodson, B. Keys to Drawing. Penguin Publishing Group; 2003.

[54] Huston, S. Figure Drawing for Artists: Making Every Mark Count. Rockport Publishers; 2016.

[55] Emilien, A, Vimont, U, Cani, MP, Poulin, P, Benes, B. Worldbrush: interactive example-based synthesis of procedural virtual worlds. ACM Transactions on Graphics 2015;34(4). doi:10.1145/2766975.

[56] Guo, J, Jiang, H, Benes, B, Deussen, O, Zhang, X, Lischinski, D, et al. Inverse procedural modeling of branching structures by inferring l-systems. ACM Transactions on Graphics 2020;39(5). doi:10.1145/3394105.

[57] Ellis, K, Ritchie, D, Solar-Lezama, A, Tenenbaum, JB. Learning to infer graphics programs from hand-drawn images. In: Proc. International Conference on Neural Information Processing Systems. 2018,.

[58] Jones, RK, Barton, T, Xu, X, Wang, K, Jiang, E, Guerrero, P, et al. Shapeassembly: learning to generate programs for 3d shape structure synthesis. ACM Transactions on Graphics 2020;39(6). doi:10.1145/3414685.3417812.

[59] Jones, RK, Guerrero, P, Mitra, NJ, Ritchie, D. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. ACM Transactions on Graphics 2023;42(4). doi:10.1145/3592416.