Visualizing for the Non-Visual: Enabling the Visually Impaired to Use Visualization

Jinho Choi, Sanghun Jung, Deok Gun Park, Jaegul Choo and Niklas Elmqvist

Korea University, Seoul, South Korea, University of Texas at Arlington, TX, USA, University of Maryland, College Park, MD, USA

Abstract

The majority of visualizations on the web are still stored as raster images, making them inaccessible to visually impaired users. We propose a deep-neural-network-based approach that automatically recognizes key elements in a visualization, including a visualization type, graphical elements, labels, legends, and most importantly, the original data conveyed in the visualization. We leverage such extracted information to provide visually impaired people with the reading of the extracted information. Based on interviews with visually impaired users, we built a Google Chrome extension designed to work with screen reader software to automatically decode charts on a webpage using our pipeline. We compared the performance of the back-end algorithm with existing methods and evaluated the utility using qualitative feedback from visually impaired users.

CCS Concepts

• Human-centered computing → Visual analytics; Visualization toolkits;

1. Introduction

Intrinsic to visualization is that it is visual, i.e., that it makes use of the human visual system to effectively convey data to the viewer. Obviously, this visual theme permeates the entire discipline—the scientific community is alive with visual design guidelines, graphical perception studies, and practical advice on color theory. As Card et al. [CMS99] note, much of human thinking is couched in visual terms, in that to understand something is called “seeing it,” in that we want to bring our thoughts into “focus,” and in that we strive to make our ideas “clear.” However, what if all you have are words and no pictures; that is, what if you are visually impaired? Is the power of visualization forever closed to you, or worse, are you actively barred from accessing important data about our world?

Such is certainly the case on today’s internet, where the web, for all its revolutionary impact on improving information access for the visually impaired [CP15], still holds hundreds and thousands of charts encoded in bitmap images where data are locked away for all but sighted users. (The correct term is really a pixel map, or a pixmap, to signify color rather than black-and-white images, but we will use the colloquial “bitmap” term throughout this paper.) The core assistive technology for the visually impaired, screen readers, which transforms a visual display to non-visual means such as text [HJ08], sound [Ifu17, MW94], or Braille [RSKS00, MW94],
does not work well for such data-rich images. Too often, webpages do not contain the raw data that generated these visuals, and while accessibility standards are on the rise [CP15], a vast collection of legacy charts exist on the web where no such data will ever be made available.

In this paper, we propose to bring visualization to the non-visual through an automated pipeline that analyzes the contents of bitmap images, detects the chart type, reads the labels, extracts the shapes into a vector graphics format, and decodes the data stored in them. To our best knowledge, our proposed deep-neural-network-based method is the first to extract the data from charts for visually impaired users in a fully automatic manner. The vector shapes, labels, and annotations can then be transformed into appropriate displays, such as a screen reader, an alternate visual representation more suitable for partially sighted users, or even a physical visualization. We also present a prototype implementation of a Google Chrome extension that can transparently detect static chart images on a website and translate them into non-visual means using our engine. The extension also provides information about charts such as the chart type and the number of data items and an alternative accessible interactive charts that are even useful for sighted users. We compared the performance of the back-end algorithm with ReVision [SKC∗11], which is a semi-automatic data extraction tool. We also conducted a qualitative evaluation with three visually impaired users. Their input has guided refinement of the prototype and provided insights about strengths and weaknesses of our work.

2. Related Work

Here we review relevant prior efforts to enable comprehension of visual charts for visually impaired users.

2.1. Assistive Technologies

While much of the public discourse around technology and people with disabilities tend to focus on problem areas, computing has in general had a revolutionary impact on inclusiveness and access for people with cognitive, developmental, intellectual, mental, physical, and sensory impairments [CP15]. The overall term for technologies that support disabled people in daily living is assistive technologies, and they include a range of devices from low-tech, mechanical ones, such as walkers and wheelchairs, to more technologically advanced ones, such as hearing aids and Cochlear implants for people with hearing impairments, and memory aids and conversation cues [WMMF15] to support cognitive impairment.

2.2. Assistive Technologies for Visual Impairment

Vision in particular is a critical sensory channel for effectively navigating the world. According to the World Health Organization [GLO] there are approximately 285 million visually impaired users in the world today, 39 million of them totally blind. In the United States, this number is approximately 10 million and 1.3 million, respectively (National Federation for the Blind (NFB), 2018: [Bli]). The approach taken by most assistive technologies is sensory substitution [CP15], where input from one sensory modality can be augmented with input from another sensory modality; for example, by converting written text into spoken language.

Sound is commonly used as a substitute because it is easy to generate without special hardware. Audification refers to the use of speech audio [MW94], whereas sonification refers to the use of non-speech audio [KWB∗10]. Screen readers, which transform text on a screen into voice, are examples of speech-based methods, and have quickly become integral for many visually impaired users when accessing the web.

Other than auditory sensors, tactile sensory input have been widely used; an example is a Braille-based book. Mynatte et al. [MW94] designed GUIB, which translates graphical web interfaces into textual displays. Refreshable Braille displays or Braille terminals can show a low-resolution image in addition to the traditional tactile characters [RSKS00]. Jayant et al. [JRWS07] developed a processing methods to translate figures into a form that is appropriate for Braille display. Kim and Lim proposed an assistive device, Handscope, which can translate a statistical graph (even a tag cloud) into tactile feedback [KL11]. Engel et al. [EW17] studied how design can improve readability of tactile charts and proposed improved base guidelines for chart design.

2.3. Visualization for the Non-Visual

Visualization leverages the human visual system to enable augmenting the user’s cognition by transforming symbolic data into geometric representations [CMS99]. For this reason, it has always been particularly alarming for the field of visualization to consider the “non-visual;” users who are visually impaired, and are thus potentially forever cut off from the benefits of visualization. While visualization research has considered color-blind users (e.g., people with color vision deficiencies), addressing full or even partial visual impairment has received relatively little attention in the community.

Sensory substitution using sound for visualization is either restricted to toolkits such as HighCharts, which provide native support for accessible charts [amC], or proper sonification efforts, where the data is converted to non-speech audio. For the latter, Zhao et al. [ZPSL08] investigated the use of nonverbal sound to explore georeference data using coordinated maps and tables. Goncu et al. [GMMM15] developed a web service that generates an accessible floor plan. Ferres et al. [FLST13] proposed a system that verbally describes line charts. Fact remains that if a chart is not constructed in an accessible way from the beginning, existing screen readers or sonification techniques will not work.

Touch can also be used to substitute for vision even for charts. Refreshable Braille displays can be used in lieu of a text-to-speech screen reader for the machine-readable portions of a chart [RSKS00], as well as more advanced method such as the Braille mouse [HIHP11], which combines some features of a normal mouse with a Braille display, and embossed touch maps [dAVT05], which convey data in a 2D area using shape. Initial work [XIP∗11] applies the TeslaTouch [BPH10] electrovibration mechanism to allow for sensing 2D data using touch. Fitzpatrick et al. used the statistical software R [FGS17], and Yu et al. combined tactile conversion with sound [YB02] to produce accessible statistical graphs for web. However, few of these efforts have been specifically targeted at visualization, and much work remains to be done in unlocking the thousands of raster charts available on the web today.
Perhaps the most promising approach to accessible data visualization is data physicalization [JDI15], where physical artifacts are used to convey data. While the original data physicalization manifesto does not consider accessibility as one of its corner stones, it certainly has potential for improving the inclusiveness of the visualization field. For example, Kane and Bigham work on supporting visually impaired and blind students through 3D-printed tangible maps [KB14], and Holloway et al. [HMB18] suggested replacing tactile maps with 3D models for the blind.

2.4. Extracting Data from Charts

The web has had a transcendental impact on information access for disabled people in general, and visually impaired people in particular [CP15]. The advent of accessibility guidelines for websites and screen readers that transform written text into non-visual means such as sound [Ifa17, MW94], touch [XIP11], or refreshable Braille displays [RSK00] have leveled the playing field somewhat. However, image data locked away in raster files remains a hurdle for visually impaired and blind users.

To overcome this limit, there have been attempts to extract data from charts in an image file using heuristics and expert rules. For non-raster images, Shao and Futrelle [SF05] extract simple graphical entities from SVG images in PDF for chart classification, while Battle et al. [BDM18] propose a web crawler that collects and classifies SVG images using image elements.

For raster images, Zhou and Tan used the Hough transform algorithm to extract information from bar charts, even hand-drawn ones [ZTOO]. Huang and Tan extracted information from additional chart types, including pie charts and line charts [HT07]. Similarly, Gao et al. [GZB12] extract data tables from raster images for accessibility use. SIGHT generates summary of a simple bar chart for visually impaired users [CESM12]. Chester et al. proposed a system that transforms grayscale charts into XML format [CE05]. However, none of these proposed approaches are able to handle general web-based rasterized charts, which have varying colors and shapes.

Because of the variations in the format of the charts, the performance of these systems can be improved with human guidance, as demonstrated by the ChartSense tool [JKS17]. Software packages exist that extract information from certain charts [Roh11, Tum06]. iVolYER [MNV16] extracts data from existing raster chart images and generates interactive animated visualization. Saava et al. developed a system called ReVision, to classify images of charts into five categories and extract data from bar charts and pie charts [SKC11]. They also allow users to build an alternative representation or change the color and the font of an input chart image. While these techniques require the interaction of users, such as annotating the x-label of a chart, our proposed system performs extraction without any intervention.

Poco and Heer proposed an automatic pipeline for extracting a visual encoding specification given a raster chart image [PH17]. Poco et al. focused on recovering the color mappings of charts that include a color legend [PMH18]. However, their approach mainly focuses on extracting specific components of a chart and does not extract raw data. Although there exist automatic systems to extract data from charts, they focus on only one specific chart type [AZG15, CMG15, CRM17]. Otherwise, we propose an automatic pipeline for three types of charts and utilize state-of-the-art modules based on deep neural networks.

3. Domain Characterization

The goal of our work is to extract data that is currently locked away in raster images containing charts in order to make the data accessible to visually impaired users. To do this, we need to understand how visually impaired users handle graphs and charts. For this purpose, we conducted a domain characterization on this topic.

3.1. Method

We conducted interview sessions to understand how visually impaired people currently access charts. We interviewed three visually impaired users using a structured interview format [LHF17]. Two are professional IT developers working in the web-accessibility domain (P1, P3), and the other is an undergraduate student (P2). P1 was born blind while P2 and P3 lost their vision about 10 and 20 years ago. All were male and completely blind. We conducted individual remote interviews with P1 and P3, and an on-site interview with P2. Each interview lasted approximately 30 minutes. Below we summarize our findings.

3.2. Findings

Visually impaired people typically browse the web using screen readers. P1 uses both a Braille display and speech, sometimes turning off the sound to use only the Braille display. P2 prefers the Braille display when the pronunciation of the screen reader is not clear. P3 uses only a screen reader. However, participants note that Braille displays are mostly limited to one line of characters, and that graphical Braille displays are still experimental.

For the Windows operating system, common screen readers are NVDA [NVA] and JAWS [JAW]. P1 and P2 use NVDA, because it is better integrated with the operating system, and is an open source project. P3 uses NVDA for web and JAWS for office productivity. Because NVDA is an open source project, we adopted NVDA as a development and test platform for our project. For mobile environments, Apple’s built-in VoiceOver [App] for iOS or Google’s TalkBack [Wha] for Android are commonly used. P1 uses Firefox because it works well with NVDA, and sometimes uses Google Chrome because it is fast. P2 uses Internet Explorer as a main browser because of its accessibility features and its JAWS integration. P3 uses Chrome because of specific accessibility features.

All screen readers announce whenever they detect an image. If there is an alternative text for it, that text is read aloud (or shown on the Braille display). If not, P1 and P3 use screen readers for its built-in OCR function to provide an idea of the image contents. P2 and P3 felt that the formatting on websites rarely follows accessibility standards, which makes it hard to read text or data tables. P2 often copied a webpage to a text editor to see only the textual information. For charts in SVG format, screen readers can use the alternative text for each element, or an accompanying data table if available. All participants rarely pay attention to images or charts. They rely on the textual content to understand the visual content.
However, there were some exceptions, such as when P2 was taking a statistics course. In that case, he needed to understand the course contents for the class, and had to rely on a human reader assigned by the university. According to P2, most people are not good as readers because they read what they think important, but not necessarily at the level of detail required by the visually impaired user. But when such readers were not available, he had to rely solely on textual contents. He sometimes used voice calling to get help from a sighted person. For P3, there were a few times when he wanted to access chart contents, such as when he had to read a consumer market research report that contained charts.

Data tables are an important aspect of accessing information that is otherwise expressed with graphs. Each screen reader provides its own mode and shortcut keys for table navigation. Some websites, such as government statistics sites, helpfully provide raw data tables with the charts. However, this can become tedious and difficult to overview for very large data tables. For P1, analyzing data table becomes challenging when there are more than 20 rows and 10 columns. P3 once analyzed a data table containing more than 100 rows and 50 columns; this may indicate that visually impaired users need superior memory capability instead of the external cognition [SR96] afforded by visual representations.

Technology can help visually impaired users in their computer usage. P1 had used sonification and was generally positive about its use for providing an overview of large data tables. He suggested that students who rely on data, or people who trade stocks, might be interested in using this technology. P3 felt that training is needed to use sonification effectively. P2 carries his own desktop keyboard in his bag wherever he goes, because he feels using varying physical keyboards is detrimental for his computer usage.

4. Extracting Data from Chart Images

We propose a fully automatic deep learning-based pipeline (Figure 2) that accepts a raster image as input and generates a structured representation of the chart in the image, including its original data. The system supports bar charts, pie charts, and line charts. Since charts in a webpage have various shapes, we make a few restrictions in chart styles as follow. First, input images should not have 3D effects and contain only one type of chart. Second, the system supports vertical bar charts, excluding horizontal bars and stacked bars. In the case of pie charts and line charts, legends should have distinct colors. We train deep neural networks for chart type classifications, text region detection and recognition, and data extraction. Below we present the components of our pipeline.

4.1. Chart Classification

Our system applies different methods to extract data from a given chart depending on its type. It first classifies input images into four classes: bar charts, line charts, pie charts, and others. Other types of charts not supported by our system as well as images that do not include charts are classified as “others.”

We adopt Convolutional Neural Networks (CNNs) [KSH12, HZRS16] as a classification model, which have shown impressive performance on image classification tasks. Whereas previous studies [PH17, JKS∗17] used AlexNet [KSH12] and GoogLeNet [SLJ∗15] for classification, we use residual networks [HZRS16] that yield state-of-the-art performance in most computer vision tasks. Specifically, we employ existing Resnet trained on the Imagenet dataset [RDS*15] and append a global average pooling layer before the last fully connected layer. We then fine-tune the model on our dataset using the Adam optimizer, where we set the learning rate as 0.0005. Each image was resized to 512 × 512 pixels.

For training and validation data, we crawled chart images for each chart type from Google image search. We then removed images that do not contain charts, 3D charts, as well as multi-layer charts that include multiple chart types, which account for 13 percent of the initial chart corpus. We finally obtained 938 bar charts, 627 pie charts, and 833 line charts. We used the Visual Object Classes (VOC) dataset to train the model in classifying images that do not include charts.

4.2. Text Extraction

Labels in a chart play an important role in conveying the underlying semantics of the data. Chart legends indicate the number of categories and how each category is encoded. The x and y axes indicate the actual value of the plots. The title, the x label, and the y label of a chart give detailed information about the data. Therefore, we must identify these text values and determine the role of the text to extract accurate data from a chart. Previous work [SKC∗11, JKS∗17] rely on human guidance to localize text regions and extract text.
values. However, this approach is not appropriate for visually impaired users. We therefore fully automated text extraction using deep learning. Text extraction consists of three sub-tasks: textual region detection, text recognition, also known as optical character recognition (OCR), and text role classification, e.g., whether a particular text string corresponds to legends, axis labels, tick labels, or titles (Fig. 4).

4.2.1. Textual Region Detection

For text localization, we employ the PixelLink [DLLC18] model that shows state-of-the-art performance in text detection tasks. This model applies VGG16 [SZ14] as a feature extractor, predicts text and link, and performs instance segmentation. The model takes an input image and predicts the bounding box, which informs coordinates in the image for each label. To enhance text localization performance, we first train it on the SynthText dataset [GVZ16] for 400K iterations. We then fine-tune the model using the FigureQA dataset [KMA∗17], which consists of 100,000 images with ground truth bounding boxes for texts in bar, pie, and line charts (Fig. 3). Each chart image in this dataset varies in terms of font and size of text strings, tick, and span.

4.2.2. Text Recognition

Text recognition from an image, also known as optical character recognition (OCR), is still a challenging task. We first crop text in a chart image using bounding boxes predicted by the text localization model, and run the OCR model. For our OCR model, we tried two models, Tesseract, a publicly available OCR library, and convolutional recurrent neural networks (CRNNs) [SBY15]. CRNN is composed of convolutional layers that extract feature maps and produces sequence of feature vectors, recurrent layers that capture contextual features within a sequence, and transcription layers that convert predictions into label sequence [SBY15]. We compared these two models using the FigureQA dataset and found that CRNNs recognize y-tick numerical data much better than Tesseract, which is critical to extract exact data from a chart. The model sometimes makes mistakes on certain numbers in y-tick labels. To enhance OCR performance, we apply a few heuristics for y-tick labels such as replacing “o” to “0”, “c” to “0” and “i” to “1”.

4.2.3. Text Role Classification

For text role classification, we adopt the method introduced by Poco and Heer [PH17]. They classify the text of charts into title, y-tick, y-label, x-tick, y-label, legend label, legend title, and text label using support vector machines (SVMs). The feature vector of each text element in the image is defined by bounding boxes and their geometric information. We initially applied their method on the FigureQA dataset, which offers bounding boxes for text, with some success. However, the trained model performed poorly on chart images parsed from webpages, which have various font sizes, label orientations, and label lengths.

To increase the prediction accuracy, we modify the dimension of the feature vector to 8 from its original 14. The features brought from prior work are normalized center coordinates of the text, angle with respect to the image center, vertical score and horizontal score, and normalized width and height of the text. The vertical and horizontal score are defined by the number of boxes that intersect the current box vertically and horizontally, respectively, over the total number of boxes. We add the chart type predicted by the classifier to the feature vector. For our classifier, we train a gradient boosting tree with feature vectors, where we set the number of individual learners as 1,000, the learning rate as 1, and the maximum tree depth as 20. The FigureQA dataset does not contain bounding boxes of the legend title and the text labels, and thus we exclude them and train the model to classify texts into 6 classes. Texts that are not part of these six classes, such as the watermark of a chart or other redundant labels, are classified as “others.”

4.3. Data Extraction

We extract data from three types of charts, each using varying visual encodings. This meant that we had to design extraction algorithms for each chart type. Here we demonstrate how object detection and text extraction are utilized. We then describe the method used for each chart type.

4.3.1. Decoding Visual Encodings of Charts

Different types of charts have numerous variations of their forms and shapes. For example, some charts have 3-dimensional shapes as well as multiple categories. Moreover, the color encoding scheme and the aspect ratio of a chart vary across images. Even so, most of charts share common basic graphical elements such as lines, bars, or circles. From this perspective, a crucial part of data extraction is to detect these graphical elements in a reliable manner. To this end, we adopt an integrated approach that carefully combines the object detection model that extracts circles and rectangles from a given raster image, as well as those results obtained from the text extraction module and data extraction algorithms.

4.3.2. Decoding Bar Charts

To extract data from a bar chart, we first detect bars with the Yolo2 [RF16] object detection model. We train this model using the FigureQA dataset, which contains a ground truth bounding box for each bar. The object detection model identifies bars of a bar chart from left to right and predicts the top left and bottom right positions of bounding boxes as highlighted in red dots in Fig. 4(a). The height of each bar can be simply computed by subtracting the minimum y value from the maximum y value of the predicted bounding box.
The base of the detected bars should be aligned because they start from an identical baseline. However, the detected bounding boxes of bars may not always have the same starting position. Therefore, we set the baseline as the mean value of the y position for the base of each bounding box (seen in Fig. 4(a) as a red dotted line).

To convert bar heights into actual data values, we must calculate the scale of the chart as the ratio of the number of pixels with respect to the tick span of the y-axis. These two values are driven by the results of the text extraction module. We first denote a set \( \{ b_i, o_i \} \) as the bounding box information and \( \{ o_i \in \mathbb{R} \} \) as the OCR results for every \( i \)-th text classified as the ‘y-tick’. The set is sorted by the minimum y value of \( b_i \). We compute the tick span value of the y-axis \( (s_i) \) by subtracting the tick value of the current index \( o_i \) from the tick value of the next index \( o_{i+1} \).

We repeat this process for all adjacent element pairs in the set. Tick spans of two adjacent y ticks are highlighted in red in our example (Fig. 4(b)). The OCR module often fails to correctly recognize the tick value, and the text detection module occasionally misses the y-tick (Fig. 4(b)). An example of this is seen in Fig. 4(b), where it misses the “30” y tick, therefore producing an incorrect tick span value of 20. To handle this type of error, the tick span is determined as the most frequent values among \( s_i \). In the case where y-ticks are not explicit numbers (e.g., 1M, 1B, 1K), we separate characters from the y-tick and notify users about them as the unit of the chart. The number of pixels between ticks is similarly determined by computing the distance of two adjacent texts and finding the most frequent value. The tick value and the number of pixels between two spans are determined as 10 and 100, respectively, and therefore the scale is computed as 10 in our example (Fig. 4(b)). Finally, by multiplying the scale with the height of each bar, the actual values are obtained.

Due to failures in the object detection or text detection models, the number of detected bars and the number of detected x-ticks may sometimes be different. To solve this problem, we assign the extracted value to the label that has minimum distance between the bounding box of the bar and the label.

4.3.3. Decoding Pie Charts

For a pie chart, the text extraction model first predicts labels, which indicate the name of the element, with their text and bounding boxes (Fig. 5(a)). If a pie chart includes legends (57% of pie charts), we crop the input image to small patches next to legend labels to identify the colors in legends, (Fig. 5(b)). Otherwise, we extract patches from a middle point of the center of the circle and the center of bounding boxes. We then convert the patch color into a gray scale. We find the most frequent color value of a preview patch and set this value as the color indicator of the corresponding legend label. For example, the color value of “Oxygen” is 91 and “Hydrogen” is 112, which are encoded as red and yellow, respectively. If colors of the sectors have the same luminance, we convert the color space as HSV (hue, saturation, and value).

The next step is to identify the proportions of each color. We crop the circle that includes all the arcs of the pie chart, with the bounding box predicted by the object detection model. Since the bounding box of a circle does not always include all region of the pie chart, we add some padding to the bounding box when cropping (Fig. 5(c)). We then convert the colors of the cropped image into gray scale, as we did in the preview patches, and compute the histogram of pixel colors in the cropped image. As shown in Fig. 5(d), the histogram presents the number of pixels for every color value of gray scales. We obtain the data value of each legend label by calculating the ratio of pixel counts of the color indicator with respect to the total number of pixels.

4.3.4. Decoding Line Charts

Our system interprets line charts by decoding the corresponding y values for every x-tick. The data extraction process starts by determining the tick value and the number of pixels between two spans. We apply the same method we used for bar charts (Fig. 6(a)).

We then determine the y coordinate of the baseline (Fig. 6(b)). The y coordinate of the baseline can be simply obtained by the bounding box of the x-tick. However, some charts do not start the...
Figure 6: Line chart data extraction process. We count pixels with respect to tick value. Next, we find the baseline of the chart. We then convert pixels into HSV and find the most frequent color values. We calculate the distance of the x-tick and a line.

The next step of data extraction is to compute the distance between x-ticks and the line. Starting from the baseline we calculated, we determine the height of a line by moving the point vertically until we find pixels with an identical hue as that of the color indicator (Fig. 6(d)). We repeat this procedure for every detected x-tick. We finally convert heights into actual values, as we did for bar charts.

4.4. Browser Plugin for Visually Impaired Users

We implemented a Google Chrome extension that detects images in a webpage, sends the URL to a web service implementing our model, and then dynamically appends the data table and accessible interactive charts onto the original webpage (Fig. 7). The extension first parses image URLs from a browser whenever a new page is loaded. Classifying all images on a webpage using the full model requires considerable computation. We therefore apply a rule-based heuristic to simplify the process. The classifier omits those images whose width or height is smaller than a particular threshold (200 pixels); such images are mostly icons or logos. The system proceeds with the remaining steps for data extraction if the image is classified as a pie chart, a bar chart, or a line chart. We then convert the extracted data into HTML table and attach it to the webpage. Visually impaired users can access the data table of charts with screen readers. We used an iterative design process for the Chrome extension using formative interviews with visually impaired users which is described in Section 6.

Furthermore, the Chrome extension also adds a more accessible interactive version of the charts (Fig. 8). Visual impairment is a diverse spectrum. For minor visual impairment, the extracted information can be used to help in these scenarios. Users can change the type of the charts according to their preferences. The text or image segment can be augmented with an interactive tooltip overlaid over the static raster image, such that when the user hovers their mouse over specific segment of bar charts, the label and data point can be
verbally spoken. Such an interactive version of charts will improve accessibility to people with poor vision, such as the elderly.

5. Quantitative Evaluation

This section presents experimental results for our model.

5.1. Evaluation Dataset

To evaluate our system, we sample 100 images for each type of chart from a validation set from the FigureQA dataset. The images found on the web are far more varied than those included in FigureQA. We therefore randomly sample 100 images from the web image corpus for each chart type and use them as additional test set. Since most images collected from the web do not contain raw data, two authors manually annotated chart types, bounding boxes, and textual contents as well as numerical value for each x-label.

5.2. Recognition Accuracy

We evaluate the text and data extraction modules of our system using the two datasets. We also compare our method with existing systems: ReVision [SKC+11] and ChartSense [JKS+17].

5.2.1. Chart Type Classification

We remove the evaluation set from the collected chart corpus and use the remaining images as the training set. The trained model perfectly classifies all charts in the evaluation set of FigureQA and the web-collected dataset. To compare our model with previous work, we also validate our classification model on the ReVision [SKC+11] dataset, which includes 2,084 images in 10 classes. Following the test protocol used by Poco and Heer [PH17], we split the ReVision dataset into 75% and 25% for training and evaluation sets. Our classifier also shows state-of-the-art performance in classifying 10 chart types in the ReVision dataset as shown in Table 1.

5.2.2. Text Extraction

We first evaluate the text detection model using the Intersection-Over-Union (IOU) metric. This metric finds the best matching bounding boxes between the ground truth and the predicted ones, computes the area of the intersecting region divided by the area of their union region for each match, and regards it as successful prediction if the IOU measure is higher than the threshold, e.g., 0.5. As shown in Table 2 (left), our model detects most labels for the FigureQA dataset. For the web-collected dataset, our model achieves a high precision but a relatively low recall value (Table 2(right)). A main reason for the low recall is due to low-resolution images in the web-collected dataset, as shown in Fig. 9(d). The lower the resolution of a chart image is, the more failure of text detection occurs. The model also fails to detect long sequences of words as a single text, as shown in Fig. 9(c). If the model detects it as several text regions of words, the IOU values will be below the threshold, and this metric will consider them as failures.

We then evaluate the text role classification model. As shown in Table 3, the text classifier achieves over 90% F1 scores on the FigureQA dataset, except the x-label and the legend label. If legend labels are located near the x-label, as shown in Fig. 9(b), the model

Table 1: Chart classification results compared to Poco & Heer [PH17], ChartSense [JKS+17], and Revision [SKC+11] using 10 chart types in Revision dataset.

<table>
<thead>
<tr>
<th>Chart Type</th>
<th>Ours</th>
<th>Poco &amp; Heer</th>
<th>ChartSense</th>
<th>ReVision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>96%</td>
<td>95%</td>
<td>67%</td>
<td>88%</td>
</tr>
<tr>
<td>Bar</td>
<td>98%</td>
<td>97%</td>
<td>93%</td>
<td>78%</td>
</tr>
<tr>
<td>Line</td>
<td>99%</td>
<td>94%</td>
<td>78%</td>
<td>73%</td>
</tr>
<tr>
<td>Map</td>
<td>97%</td>
<td>96%</td>
<td>88%</td>
<td>84%</td>
</tr>
<tr>
<td>Pareto</td>
<td>100%</td>
<td>89%</td>
<td>85%</td>
<td>85%</td>
</tr>
<tr>
<td>Pie</td>
<td>96%</td>
<td>98%</td>
<td>92%</td>
<td>79%</td>
</tr>
<tr>
<td>Radar</td>
<td>94%</td>
<td>93%</td>
<td>86%</td>
<td>88%</td>
</tr>
<tr>
<td>Scatter</td>
<td>98%</td>
<td>92%</td>
<td>86%</td>
<td>79%</td>
</tr>
<tr>
<td>Table</td>
<td>92%</td>
<td>98%</td>
<td>94%</td>
<td>86%</td>
</tr>
<tr>
<td>VennDiagram</td>
<td>97%</td>
<td>91%</td>
<td>67%</td>
<td>75%</td>
</tr>
<tr>
<td>Avg</td>
<td>97%</td>
<td>94%</td>
<td>90%</td>
<td>80%</td>
</tr>
</tbody>
</table>

Table 2: Text detection results for both datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>FigureQA</th>
<th>Web-collected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Rec.</td>
</tr>
<tr>
<td>Bar</td>
<td>93.5%</td>
<td>94.0%</td>
</tr>
<tr>
<td>Lin</td>
<td>99.9%</td>
<td>98.7%</td>
</tr>
<tr>
<td>Pie</td>
<td>100%</td>
<td>99.0%</td>
</tr>
<tr>
<td>Avg</td>
<td>97.8%</td>
<td>97.2%</td>
</tr>
</tbody>
</table>

© 2019 The Author(s)

Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.
Table 3: Text role classification results (both).

<table>
<thead>
<tr>
<th></th>
<th>FigureQA</th>
<th>Web-collected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prec.</td>
<td>Rec.</td>
</tr>
<tr>
<td>title</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>y-label</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>x-label</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>x-tick</td>
<td>100%</td>
<td>70%</td>
</tr>
<tr>
<td>legend</td>
<td>93%</td>
<td>86%</td>
</tr>
</tbody>
</table>

Table 4: Charts extracted for our system compared to ReVision.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>FQA</th>
<th>web data</th>
<th>FQA</th>
<th>web data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>99%</td>
<td>89%</td>
<td>49%</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>Pie</td>
<td>92%</td>
<td>86%</td>
<td>100%</td>
<td>70%</td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>72%</td>
<td>88%</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>87.67%</td>
<td>87.67%</td>
<td>74.5%</td>
<td>47.5%</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Mean error rate for our model vs. ReVision.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>FQA</th>
<th>web data</th>
<th>FQA</th>
<th>web data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>0.33</td>
<td>0.45</td>
<td>0.5</td>
<td>2.23</td>
<td></td>
</tr>
<tr>
<td>Pie</td>
<td>1.01</td>
<td>0.81</td>
<td>0.12</td>
<td>0.57</td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>2.58</td>
<td>2.07</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>

Object detection errors, text classification errors, and text detection errors account for 21.6%, 49.7%, and 29.7% of total number of errors. Most errors occur in low resolution images.

Figure 9: Common examples of data extraction failure. Object detection errors, text classification errors, and text detection errors account for 21.6%, 49.7%, and 29.7% of total number of errors. Most errors occur in low resolution images.

5.2.3. Data Extraction

To the best of our knowledge, despite many efforts to extract data from chart images [JKS∗17,Roh11], a fully automated system does not so far exist. Our work is also one of the first system that aids visually impaired to access visualizations on the web. Thus, we compare our model to a semi-automatic approach, ReVision [SKC∗11], using both the FigureQA dataset and the dataset that we collected from the web. As ReVision cannot detect text automatically, we manually annotated bounding boxes in the evaluation datasets with the interface that ReVision provides to crop text regions for OCR.

We evaluate systems based on success rate, which indicates the proportion of the number of charts extracted without failure with respect to the total number of charts. The extracted results are treated as successful only if the model predicts all data elements without missing out and the error rate [JKS∗17] is lower than the threshold of 5. We also use the mean error rate, defined as

$$\text{mer} = \frac{1}{n} \sum_{i=1}^{n} \frac{|g_i - p_i|}{g_i},$$

where $g_i$ is the ground truth value of the $i$-th element, $p_i$ is a predicted value of the $i$-th element, and $n$ is the number of charts extracted successfully. We exclude charts whose error rate is over the threshold when computing mean error rate because these outliers might make the comparison less meaningful.

Table 4 presents success rates of our system and ReVision. Our system performs superior to ReVision on both the FigureQA dataset and the web-collected dataset. The difference between mean error rates of our model and ReVision becomes larger for the web-collected dataset (Table 5). Only in the case of pie charts does ReVision perform slightly better than our system for the FigureQA dataset, whose charts are simple and clear. However, the success rate of ReVision is lower than ours while the mean error rate is almost the same for the web-collected dataset, indicating the superiority of our system to ReVision for various charts. Since ReVision does not support a line chart, we only evaluate this in our system. Line charts in both dataset consist of multiple lines with multiple categories, which leads to the mean error rate to be increased.

Object detection errors, which account for 21.6% of the total failure, occur when the bar has grid effects or color gradients (Fig. 9(a)). Because the charts we collected have a number of variations in their texts, 49.7% of failure was due to errors on text classification (Fig. 9(b)). The system also often fails to localize labels if texts have large space between words or are not aligned at a single horizontal line (Fig. 9(c), with 29.7% failure rate).

6. Qualitative Evaluation

In addition to the quantitative evaluation comparing the performance of our model, we conducted an qualitative evaluation to collect feedback from visually impaired users.
6.1. Method
Evaluating with visually impaired users can be difficult due to the relatively low number of appropriate participants. Using proxy users, e.g., individuals without disabilities simulating disabilities is not desirable because visually impaired people typically develop other skills over time to overcome their disability, while proxy users lack them [LFH17]. Elmqvist and Yi [EY15] suggested using qualitative expert reviews for such situations. Hence, we engaged three visually impaired people (P1, P2, and P3) in such an expert review.

We asked our participants to view web-based charts and use the NVDA screen reader to hear the output from data tables extracted using our Google Chrome extension. Then the participants were asked a few questions about the data. For example, we asked the participants to find the maximum and the minimum value of bar and pie charts, and to describe the shape of line charts. After the session, we asked the participants for their general feedback. Each session was conducted individually and lasted half an hour.

6.2. Findings
Our experts found several usability issues that we corrected during iterative development. The first issue was the format of the generated HTML table. P1 suggested the use of the accessible table format that supports keyboard shortcuts (for accessible tables, refer to [Web]). P2 pointed out web accessibility errors in the generated table. While the table is acceptable for sighted people, non-compliance to accessibility standards such as the use of scope attribute in the table row headers slowed down the table comprehension. We fixed this concern and confirmed the update with P3.

Our experts also found that the descriptive caption was originally too long, slowing their navigation when read by the screen reader. We moved captions to a text paragraph prior to the generated table, which looked identical to sighted users, while it was a significant accessibility improvement for our participants. P1 suggested to add a subtle sound cue when a new table is generated from a chart.

We also found that the presentation of numbers can also affect user perception. For example, when the values from the pie charts are presented, it was given as a long floating-point number. This number format was detrimental because the users had to memorize the entire table to build an overview. Changing it to a simple integer value helped this mental process by reducing the memory footprint.

In addition to specific usability feedback, our participants also provided general feedback about the system. P1 stated that our tool is innovative in that it can extract data even for charts that do not provide any web-accessibility features. Previously, he manually used an OCR tool to extract text from an image and used it to guess chart type and collect some text labels from the image. P1’s primary concern was about how the system is robust to errors. Error types include incorrect, missing, or misaligned data values as well as incorrect OCR results. Misaligned error means mismatching the data values and data labels, which can result in off by one error for all table rows. P1 felt that incorrect data values and incorrect OCR results would not be a critical issue, because users can easily fix these kinds of errors. Furthermore, they are accustomed to these kinds of errors and can try to verify with other available OCR tools. However, misaligned and missing data values are potentially problematic because it can lead to a completely different understanding of the data. Nevertheless, P1 felt that the gain from our approach significantly exceeds the potential risk, given that the overall recognition performance improves over time. P3 suggested to use sound according to the confidence of the algorithm. For example, when data is uncertain, the tool can use a suggestive musical chime.

An interesting point about the screen reader is that it works as an environment for multiple plugins. For example, P1 used a strategy of running an OCR plug-in on a chart image when our tool failed. P1 noted that our tool could fit into this plugin ecosystem by providing the unique role of extracting data tables from raster images. Ideally the OCR programs should have the ability to transform the images to the description and the charts into the data tables.

P1 and P3 suggested that our tool would be particularly useful in educational settings for visually impaired. After learning about various statistical charts, users would be able to build mental images of the charts using the data table as well as visual shape of charts such as bar, line, and pie. This is helpful because each chart form has a unique characteristic and usage area, and being able to build a mental visualization can be better than having only the data table.

7. Conclusion and Future Work
We have presented a method to automatically reconstruct visualizations stored in raster images into their original geometric shapes, and then use this information to extract the underlying data. Our application for this data is to support the visually impaired in understanding chart images. Our method interfaces with current screen readers, thus making previously locked-away data accessible to a huge population. We present a Google Chrome extension that automatically detects, extracts, and presents any raster charts as a thumbnail, a multi-level textual summary, and as a raw data table.

In our future work, we will continue studying the use of visualization for visually impaired users. In particular, we believe that we can do better than just presenting the raw data to the user, but instead automatically derive insights about the data. Furthermore, methods such as ours are fascinating because they essentially mimic our own perceptual system. We will refine our model so that it can eventually recognize, decode, and extract data from general visual representations, not just familiar ones.

Acknowledgments. This work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (No. NRF2016R1C1B2015924). Jaegul Choo is the corresponding author.

References

© 2019 The Author(s)


[CE05] Chester D., Elzer S.: Getting computers to see information graphics so users do not have to. In International Symposium on Methodologies for Intelligent Systems (2005), Springer, pp. 660-668.


