ChartSense: Interactive Data Extraction from Chart Images

Daekyoung Jung[1] **Wonjae Kim**[1] **Hyunjoo Song**[1] **Jeong-in Hwang**[1]
Bongshin Lee[2] **Bohyoung Kim**[3] **Jinwook Seo**[1]
[1]Seoul National University, Seoul, Republic of Korea
[2]Microsoft Research, Redmond, WA, USA
[3]Hankuk University of Foreign Studies, Yongin-si, Republic of Korea
{rati, wjkim, hjsong, jihwang}@hcil.snu.ac.kr
bongshin@microsoft.com bkim@hufs.ac.kr jseo@snu.ac.kr

ABSTRACT
Charts are commonly used to present data in digital documents such as web pages, research papers, or presentation slides. When the underlying data is not available, it is necessary to extract the data from a chart image to utilize the data for further analysis or improve the chart for more accurate perception. In this paper, we present ChartSense, an interactive chart data extraction system. ChartSense first determines the chart type of a given chart image using a deep learning based classifier, and then extracts underlying data from the chart image using semi-automatic, interactive extraction algorithms optimized for each chart type. To evaluate chart type classification accuracy, we compared ChartSense with ReVision, a system with the state-of-the-art chart type classifier. We found that ChartSense was more accurate than ReVision. In addition, to evaluate data extraction performance, we conducted a user study, comparing ChartSense with WebPlotDigitizer, one of the most effective chart data extraction tools among publicly accessible ones. Our results showed that ChartSense was better than WebPlotDigitizer in terms of task completion time, error rate, and subjective preference.

Author Keywords
Chart recognition; Data extraction; Chart classification; Deep learning; Mixed-initiative interaction.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Due to their perceptual advantages over textual representations, charts are popular and preferable means to represent important numerical data in documents [19]. People use various types of charts in digital documents, many of which can be reused for several purposes such as further analyzing the data presented in the chart or improving the chart design. However, for most static charts available on the Web or in digital documents, people do not have access to the underlying data [8].

An automatic chart data extraction tool like ReVision could help people obtain data, but the extraction accuracy might be too low for practical use with general chart images because text region detection in images is challenging (i.e., often less than 80 percent of detection rates) even with the state-of-the-art algorithms [32]. A tool like WebPlotDigitizer [23] could help people obtain more accurate results with additional manual extraction capability, but it is tedious and time-consuming to handle multiple series data.

In this paper, to suggest a more practical solution to this chart data extraction problem, we present ChartSense, a semi-automatic interactive chart data extraction tool. ChartSense integrates algorithms to automatically detect marks (e.g., bars in a bar chart) and simple user interactions to support more accurate and efficient extraction of the underlying data from static chart images. For more effective extraction, ChartSense adopts the twofold pipeline proposed by Savva et al. [24]: ChartSense first identifies the type of chart by exploiting deep learning techniques, and ChartSense then applies an interactive data extraction algorithm most appropriate for the identified chart type. In addition, ChartSense provides a set of simple interactions to fine-tune the result, enabling more accurate data extraction.

We also evaluate the efficacy of ChartSense by comparing its classification accuracy with ReVision [24], the state-of-the-art chart type classification system, and its data extraction accuracy and task completion time with WebPlotDigitizer [23], the most effective one among the publicly available chart data extraction tools.

This paper makes the following contributions:

1. A chart type classification method using deep learning techniques, which performs better than ReVision [24].
2. A mixed-initiative interaction design for fast and accurate data extraction for six popular chart types.
3. The design and development of ChartSense, an interactive chart data extraction system equipped with

CHI 2017, May 06 - 11, 2017, Denver, CO, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4655-9/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3025453.3025957

the mixed-initiative interaction and the chart type classification method.

4. A controlled experiment showing the efficacy of ChartSense in comparison to WebPlotDigitizer and for three additional chart types.

RELATED WORK

Our work is built upon prior work in three related research areas; 1) mixed-initiative approaches and systems, 2) chart data extraction algorithms and tools, and 3) applications that use chart data extraction algorithms.

Mixed-initiative Approach and System

Mixed-initiative user interfaces enable people to collaborate effectively with intelligent agents [11]. Horvitz presented critical factors that should be considered when designing mixed-initiative interfaces that integrates automated services with direct manipulation interfaces. Many researchers tried similar approaches to solve challenging problems in various domains. Schwarz et al. [25] presented a framework for handling uncertainty in user inputs by providing feedback about input uncertainties. The feedback changes dynamically through user interactions depending on the probabilistic states of corresponding UI elements. Taking a similar approach, Gao et al. [7] presented a mixed-initiative system to resolve ambiguity in natural language interfaces for data visualization. Healey et al. [10] presented a semi-automatic visualization assistant system with which people can collaborate to improve the results of the system for more effective visualization of multi-dimensional datasets. Their approach is also based on a mixed-initiative strategy where people can integrate their strength with AI-based search algorithms. We also adopt a mixed-initiative approach in ChartSense to combine advantages from automatic chart mark extraction algorithms and people to improve efficiency and accuracy of data extraction from chart images.

Chart Data Extraction

There are automatic chart data extraction algorithms based on image processing and machine learning to extract data from chart images [6, 12, 13, 26]. They depend largely on edge detection or vectorization algorithms. Therefore, if the result of the edge detection or vectorization algorithms is noisy for an input chart image, the data extraction accuracy could degrade significantly. For example, when we apply the Canny edge detection algorithm [3] to line charts in our chart image corpus for line detection, we observe that too much noise is included in the result to accurately detect the target lines. We thus take a mixed initiative approach, i.e., involve users in the data extraction algorithms to improve the accuracy of the extraction results.

ReVision [24] is a system that automatically classifies chart types and extracts data from a chart image. It classifies a chart image as one of the ten chart types and automatically extracts data from bar charts and pie charts. Its classification accuracy is about 80% on average, and it extracts marks successfully from 71% of bar charts and 64% of pie charts. We build upon ReVision's twofold pipeline (type classification first and then data extraction) and improve accuracy for both the type classification and data extraction.

WebPlotDigitizer [23] is a web-based tool that extracts data from four charts types (bar & line charts and polar & ternary diagrams). It has both automatic and manual modes. In the manual mode, people have to specify necessary information for data extraction. For example, they have to specify y positions of all bars' top for extracting data from a bar chart. In the automatic mode, it extracts marks automatically using a simple color detection technique. However, due to its low accuracy, people usually use the manual mode, which is faster and more accurate than the automatic mode. Ycasd [8] is a data extraction tool for line charts. It uses a manual technique similar to WebPlotDigitizer with which people have to specify all the data points in a line chart.

iVoLVER [21] is also a web-based versatile tool that extracts data from an image and reconstructs representations of the data. It relies on users' inputs in both specifying data types (e.g., text, colors, shapes, etc.) and sampling data points in the image. Thus, it requires a relatively large number of interactions to extract data accurately from a chart image. ChartSense adopts a mixed-initiative approach for faster and more accurate data extraction.

DataThief [30] is another tool that extracts data from line charts. It can extract one line at a time, so people have to run it multiple times for a chart with multiple lines. People also have to specify six points (origin point, end-points of x-y axis, start- and end-point of the target line, and a point on the line). DataThief extracts intermediate points between the start- and end-points of the line while tracing along the line. However, the tracing often fails and halts, so people have to manually specify the tracing direction at the halting point.

Applications using Chart Data Extraction

Chart data extraction results have been used in a number of applications, such as redesigning a chart, generating helpful overlay for a chart, mapping a text to a mark based on crowdsourcing, and aiding search and retrieval of chart images. ReVision [24] provides support for redesigning a chart for better perception. It extracts a relational data table from an input chart image and presents possible visualizations using the extracted data. ReVision allows people to select an alternative chart design from a list of charts ranked by MacKinley's effectiveness [20].

Harper and Agrawala [9] presented another technique for redesigning existing charts. Their technique extracts data from D3 [1] visualizations by analyzing the structure of documents generated by D3.js, and enables non-expert users to easily modify visual attributes of the target visualizations. Their system includes two tools: (1) *deconstructing* tool extracts marks, underlying data, and mapping between them from a visualization and (2) *restyling* tool helps users change the style of the visualization. If underlying data can be extracted from bitmap chart images, it becomes possible for their *restyling* tool to convert bitmap chart images to interactive D3 charts.

Graphical overlays [15] are a technique that automatically generates helpful overlays from chart images. It is based on mark extraction results by ReVision and DataThief. Graphical overlays are applicable to bar, pie, and line charts because ReVision and DataThief can extract data from those chart types. The graphical overlays technique can be used with other chart types if combined with mark extraction algorithms for more chart types.

Kong et al. [16] presented a useful application of chart mark and data extraction. They presented a crowdsourcing pipeline to generate mapping between text phrases in the main text and marks in the chart. They used ReVision to extract marks and corresponding data values from chart images in the pre-processing stage. This application can benefit from better chart mark and data extraction.

Choudhury and Giles [5] presented an architecture for extracting information from figures in a PDF file (i.e., academic papers). Their architecture consists of underlying data extractor, metadata extractor, natural language processor (to understand the semantics of figures), and search engine for figures and metadata. The architecture is like a digital library for figures, where users can search for figures of interest. Their data extractor module is limited to line charts. More accurate and general chart data extraction could make the architecture more widely applicable. Siegel et al. [27] parsed result-figures in research papers, and facilitated searching and retrieving of the figures. Chen et al. [4] proposed a search engine based on diagram component extraction, which is capable to search for similar diagrams with partially matched components.

CHARTSENSE SYSTEM

We designed the ChartSense system using the twofold pipeline of chart data extraction suggested in ReVision [24]: chart classification and data extraction. We implemented ChartSense as a web application: the server is responsible for type classification and data extraction while the client provides user interfaces for our interactive data extraction.

Chart Classification

Chart Image Corpus Construction

A corpus of chart images with correct tagging of chart types plays an important role in accurate chart type classification. Before introducing our chart type classification technique, we explain how we built our chart image corpus.

We started with the chart images corpus used in ReVison by Savva et al. [24]. To increase the corpus size, we collected additional chart images and manually tagged them with their chart types. We ran a script to collect chart images using Google image search and manually removed incorrect search results. To compare the accuracy of our classification with that of ReVision, we collected the same ten chart types as ReVision (area, bar, line, map, Pareto, pie, radar, scatter plot, table, and Venn diagram). We used the chart name as a search keyword (e.g., 'area chart,' 'bar chart') for each chart type and collected all images returned by Google image search. As a result, we obtained 737 ~ 901 images for each chart type (Table 1). Among these, we removed the following inappropriate images:

Chart Type	Collected	Filtered
Area chart	819	509
Bar chart	866	557
Line chart	885	619
Map	889	567
Pareto chart	737	391
Pie chart	874	568
Radar chart	822	465
Scatter plot	872	696
Table	901	594
Venn diagram	849	693

Table 1. Number of newly collected and filtered images

- False search results (e.g., bar charts or donut charts in pie chart image search)
- Abstract menu icons or symbols
- Images that include multiple types of charts (e.g., images with both bar and pie charts)
- Handmade sketches
- Box plots in bar chart image search
- Photos in map image search
- Forms in table image search

Table 1 shows the final number of images, collected and then filtered for each chart type. In this way, we constructed two image corpora for classifier evaluation: 1) small corpus (n = 2084): ReVision's chart images and 2) large corpus (n = 6997): merged corpus of ReVision's and newly obtained chart images (n = 5659 in Table 1).

Neural Network Model for Classification

We built our chart type classification model based on convolutional neural network (CNN), a type of feed-forward artificial neural network that was proven to show good accuracy for image classification among the variations of neural networks [17]. CNN consists of three layers: convolution layer, pooling layer, and fully connected layer. Since implementation details of each layer is not an issue for reproducing CNN, we explain each layer in a conceptual manner. Convolution layer tangles nearby pixels to abstract their meaning. Pooling layer extracts representative specimens from the result of the convolution layer to reduce computational time. Fully connected layer does conventional neural network learning with a back propagation method.

Among many of CNN variations, we chose GoogLeNet [28], an ensemble model based on CNN that showed the best performance in the ImageNet Large Scale Visual

Model	Classification Accuracy (%)		number of params
	Small corpus	Large corpus	
LeNet-1	41.8	44.2	0.4M
AlexNet	77.9	88.8	56.9M
GoogLeNet	76.7	91.3	6.0M

Table 2. Classification accuracy and size of models.

Recognition Competition (ILSVRC) 2014 competition. We also evaluated models with relatively shallower networks (i.e., LeNet-1 [18] and AlexNet [17]). We used our two constructed image corpus for training and validation. Training and validation sets were randomly divided into 80% and 20%, respectively, from the corpus.

Throughout the training process, we mainly used a deep learning framework Caffe [14]. We first normalized an image into a dimension of 256 × 256 × 3 (width × height × # of color channels) with a Python image library, and then constructed image database with Lightning Memory-Mapped Database (LMDB) to achieve higher I/O performance. We used all three networks described in the Caffe format from the Caffe Model Zoo. Among six Caffe solvers, we chose stochastic gradient descent (SGD) solver, which is widely used due to its simplicity and time efficiency [2]. The learning rate policy, which should be specified prior to running the solver, was initially set to 0.01 and was dropped by a factor of 10 at the 33% (0.001) and 66% (0.0001) of the iteration process. Each iteration does forward and backward propagations to obtain output and update weights. We used one Amazon Web Service g2.2xlarge (26 ECUs, 8 vCPUs, 2.6 GHz, Intel Xeon E5-2670, 15 GB memory) instance with Caffe-installed AMI (Amazon Machine Image) to train the models. It took approximately 2 hours to train GoogLeNet and AlexNet using the large corpus, and about 30 minutes for the small corpus. Training LeNet-1 took less than a minute for both corpora.

Chart Type Classification Accuracy

We first compared the accuracy of the three classification models (Table 2). Both AlexNet and GoogLeNet showed considerably higher accuracy compared to LeNet-1, yet the difference between the two was relatively small. However, the number of parameters of GoogLeNet model (approx. 5.9M) was ten times smaller than that of AlexNet model (56.9M). In terms of memory efficiency and its accuracy on large corpus, we adopted the model trained with GoogLeNet in ChartSense. Then, we compared the accuracy of our chart classification model with that of ReVision [24]. We calculated the accuracy using five-fold cross validation (five repetitions with 80% training and 20% validation sets). We trained our model using the two image corpora (i.e., small and large corpora).

Overall, ChartSense shows higher average accuracy than ReVision for both datasets. Figure 1 shows the chart classification accuracy results per chart type along with the overall average accuracy with all chart types combined. For individual chart types, ChartSense exhibits higher classification accuracy for all chart types than Revision when trained with the large corpus, but for six out of 10 types (bar chart, line chart, map, pie chart, scatterplot, and table) with the small corpus.

Data Extraction

In this section, we first summarize three main challenges in developing an automatic chart data extraction algorithm that has enough accuracy and effectiveness for practical use. We then describe a mixed-initiative approach we propose to overcome these challenges by utilizing both image processing techniques and user interactions.

Three Challenges in Chart Data Extraction

First, the diversity in chart style makes it difficult to apply a single extraction algorithm to all the charts of the same type: although two chart images belong to the same chart type, their detailed chart styles can be significantly different. For example, some line charts contain no tick marks on the *x*-axis or others may not have horizontal guidelines along the *y*-axis. If a data extraction algorithm works under the assumption that the chart image has tick marks on the *x*-axis or guidelines

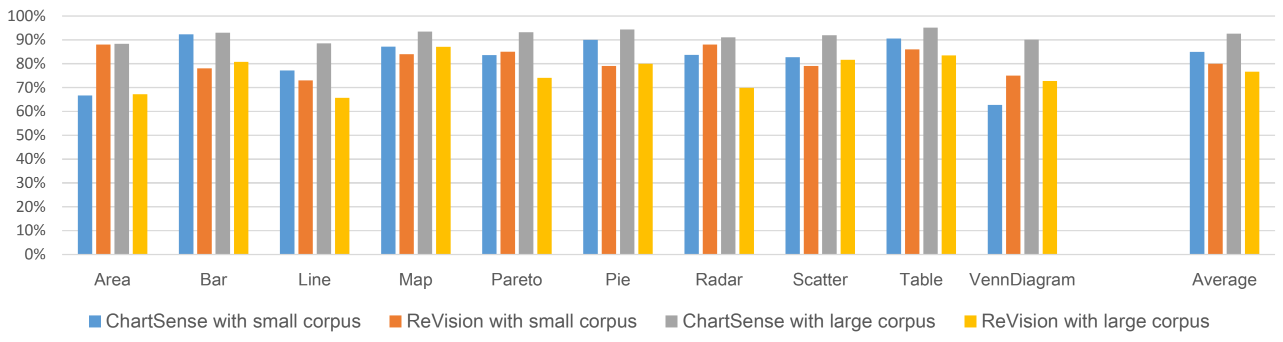


Figure 1. Accuracy of chart type classification of ChartSense and ReVision [24] for each chart type. Each classification model is trained with the small corpus and the large corpus.

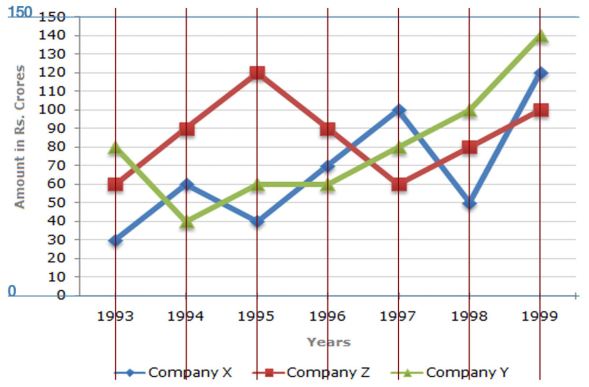


Figure 2. Users specify key features: two pairs of *y*-position/values (blue lines and texts) and interval of data points (red lines).

along the *y*-axis, it is not likely to work well for the charts that do not meet the assumption.

Second, Gestalt principles describe how effective we humans are in simplifying and deciphering even occluded visual components [31]. However, algorithms may have trouble in interpreting overlapped visual entities. For example, when two lines intersect each other in a line chart, humans can correctly perceive both series that intersect on a point. However, it could be challenging for line detection algorithms, which usually use pixel colors for identifying multiple lines, to correctly separate two series.

Third, to convert extracted marks into data values, it is necessary to read labels beside axes. Once a text region containing a label is identified, we can extract the label by using optical character recognition (OCR). Many OCR algorithms show reasonably high accuracy; however, to the best of our knowledge, although some previous works tried to solve similar problem for graphics or maps [22, 29], there has not been yet a text-region-detection algorithm for chart images with sufficient accuracy for practical usage (often less than 80 percent for detection and less than 60 percent for recognition) [32]. For example, chart data extraction tools such as ReVision [24] ask people to specify text regions in chart images manually.

Mixed-Initiative Approach

The main goal of ChartSense interface and interaction design was to maximize the accuracy of data extraction results while minimizing the burden of manual user interactions. To achieve this design goal, we first reviewed available techniques for automatic data extraction and identified their strengths and weaknesses by trying them with many real chart images. Then, we designed basic interactions to overcome the weaknesses (1) by asking users to specify critical features (e.g., y-values in a line/area chart, base colors and center point in a radar chart) with which automatic extraction algorithms can generate more accurate results and (2) by presenting the automatic extraction results in a way to match unique characteristics of each chart type so that users can effectively fine-tune the results (e.g., adjusting the center point in a pie chart). Thus, reliability of chart type classifier was critical in our system and we improved the accuracy by utilizing a deep learning-based algorithm.

To support the fine-tuning of the results from data extraction algorithms, we tightly coupled a table view for extracted data values and a chart view where the reconstructed chart is overlaid on the original chart image; users can interactively verify all extracted values because any adjustments in either view are coordinated.

We here detail the data extraction algorithm for each chart type after explaining a few assumptions we made. We also report the portion of images covered by the assumptions in the newly obtained corpus. We denote user interaction and ChartSense reaction using '[US]' and '[CS],' respectively.

Line Chart

We make the following four assumptions regarding line charts: 1) line charts do not have 3D effects; 2) each series has a distinct color; 3) intervals are equal in size; and 4) each series has horizontal orientation (e.g., time axis for time series data is the horizontal axis). 84.17% of the line charts in our newly obtained corpus satisfy all the four assumptions. The percentages of line chart images that are excluded by not fulfilling the four assumptions are 2.26%, 5.33%, 5.65%, and 3.55%, respectively.

The data extraction process for line charts is as follows:

1. [US] Specify a bounding rectangle that encloses all the lines
2. [US] Specify two *y*-positions and the corresponding data values (Figure 2)
3. [US] Specify intervals of data points (Figure 2)
4. [CS] Detect lines in the rectangle specified in step 1
5. [US] Select correct lines among the detected ones
6. [CS] Convert the selected lines into values, reconstruct a line chart from the values, and overlay the line chart with the input image
7. [US] Adjust incorrectly recognized data points

Step 1 helps ChartSense crop the input image for more efficient processing in the following steps. In step 3, ChartSense incorporates user interactions to minimize user inputs required to define the interval. Users need to specify only three values: the horizontal positions of the leftmost and rightmost data points, and the number of horizontal sampling positions. ChartSense then calculates all other sampling horizontal positions between the two end positions. It shows the calculated *x*-positions with vertical red lines so that users can change the three values if necessary.

To detect lines in step 4, ChartSense starts from detecting the dominant colors in the image. Its dominant color detection algorithm converts the image into the HSV color space. ChartSense uses the H channel to acquire a histogram mapped to the number of pixels and then finds evident color ranges as in Figure 3 with a heuristically determined frequency threshold (i.e., 1% of total number of pixels). When there are more than one local maximum values within an evident color range (R3 in Figure 3), ChartSense divides the range into several parts using local minimum values as boundaries (e.g., R3, R4, and R5 in Figure 3). For each hue range, ChartSense can obtain a binary image where the value of each pixel that belongs to the hue range is white, and black otherwise. ChartSense uses the binary images to detect candidates for correct lines.

ChartSense derives least squared regression lines to enhance the accuracy of the originally extracted y-value of sampling point. To make a regression, ChartSense additionally extracts y-values from four evenly distributed locations between each pair of neighboring sampling points. Since there are two corresponding regression lines for all sampling points except for the two endpoints (i.e., the first and the last sampling points), ChartSense determines the final estimation of value by taking the average of the two derived y-values from the two regression lines. For the first and the last sampling points, ChartSense simply uses the estimated value from the only regression line. Finally, its algorithm connects all derived y-values to make a line.

Because ChartSense could not always detect lines correctly, it asks users to select real (correct) lines among the detected ones in step 5. ChartSense shows five detected line colors (each determined by taking the average of pixel values in each cluster) in a table, and users can check the corresponding detected lines one-by-one by hovering the mouse cursor over the table cells. After users select correctly detected lines, ChartSense extracts data and overlays the reconstructed line chart on the input image, and users can adjust incorrectly recognized data points by directly manipulating them.

Area Chart

Data extraction from area charts is almost identical to that of line charts. The only difference is that data extraction in line charts finds a point for each horizontal sampling position with a least square regression line while data extraction in area charts simply finds a range of y-values for each color at each sampling position.

Radar Chart

For radar charts, we make three assumptions: 1) each polygon's color is distinct; 2) all axes share the same scale; and 3) all angles between adjacent axes are same. 94.41% of radar chart images in our newly obtained corpus satisfies all the three assumptions. The percentages of chart images that are excluded by the three assumptions is 0.43%, 2.58%, and 2.58%. Color-filled polygons are particularly challenging to extract data from, because blending of overlapping polygons could generate a new polygon (see the inset figure). Thus, we utilize mixed-initiative approach in handling non-empty polygons.

The data extraction process for radar charts is as follows:

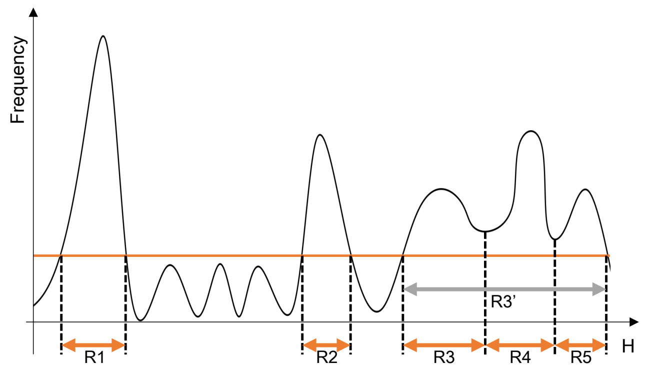
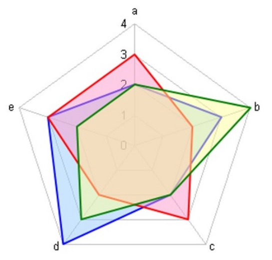


Figure 3. Detection of dominant colors from an image using a histogram built on the HSV color space. Candidate ranges are acquired using a heuristic frequency threshold (i.e., 1% of total number of pixels), which is denoted by orange line in the figure. A range with multiple local maximum values is divided again using local minimum values as boundaries.

1. [US] Specify the center point of a radar chart
2. [US] Specify two consecutive axes end points of the radar chart
3. [US] Specify background color and non-mixed foreground colors for polygons
4. [US] Specify the value for axis end points
5. [CS] Detect polygons
6. [CS] Convert the selected polygons into values, create a radar chart, and overlay the radar chart on the input image
7. [US] Adjust incorrectly recognized data points



Radar charts use a similar visual encoding with line charts; both charts use position as a visual variable to encode data. The difference is that radar charts use a polar coordinate system while line charts use the Cartesian coordinates system. While a polyline in a line chart connects data points that are on distinct vertical axes, a polygon in a radar chart connects data points on distinct radial axes. Therefore, the center point and axes angles are necessary in radar charts instead of the baseline and data points' interval in bar charts. Note that the center point and axes detections for radar charts are more challenging because of the circular arrangement. Thus, ChartSense asks users to specify the center point and the end-points of each axis.

Regarding color-filled radar charts, ChartSense additionally asks users to specify foreground color information (by clicking on regions with a non-mixed color. Since overlapped regions have blended colors, clustering based on hue is not applicable for color-filled cases. Thus, ChartSense prepares a list of blended colors from every possible combination of the non-mixed foreground colors. It clusters pixels by finding the most similar color from the list and builds binary images for each cluster. In each binary image, ChartSense samples 20 outermost points between each neighboring axes pair. Then, it builds lines with all possible pair of points. For each line, it compares an area of a triangle constructed by the line and two axes and selects the line that yields the smallest area and contains all 20 sampled points inside the triangle. Since there are two neighboring axes, final data points are determined as the average of the two estimations. When the algorithm fails to find more than one outermost points, it returns a random value on the axis to leave opportunity for manual adjustment.

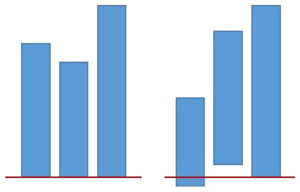
Bar Chart

We make two assumptions: 1) bar charts have no 3D effect; and 2) bar charts do not include stacked bar charts. Among our 557 bar chart images, 68.58% satisfies all the two assumptions. The percentages of chart images that are excluded by the two assumptions are 9.34%, and 24.24%, respectively.

ChartSense detects bars in a bar chart image as follows. It detects a list of color ranges using an H channel histogram as it does with line charts. For each color range, it makes a binary image by setting white for pixels that have color within the range and black for the others. To remove axis or assistance grid lines, ChartSense applies an open morphological transform (kernel size: 13) to the binary image. ChartSense then finds all connected components by 8-way connectivity, and define a rectangular bounding box for each connected component, which becomes a bar in the result. The detection algorithm for a single bar is similar to prior work [24], but if there are multiple bars in one binary images (e.g., grouped bar chart with multiple series), they are considered as the bars from the same series.

Using the bar detection algorithm, ChartSense extracts data from bar charts as follows.

1. [CS] Detects the baseline (*x*-axis) and overlay the detected baseline on the input image
2. [US] Adjust the baseline if needed
3. [US] Specify two *y*-positions and the corresponding data values
4. [CS] Detect bars, convert them into values, create a bar chart from the extracted values, and overlay the bar chart on the input image
5. [US] Adjust incorrectly recognized bars if needed



ChartSense detects bars in the input image by the bar detection algorithm. Then, it infers chart orientation from the number of bars that shares vertical or horizontal baselines (i.e., the orientation is vertical when there are more bars with horizontally aligned bottom lines, and vice versa). If the extracted bars' bottom lines are aligned on a line, ChartSense regards the line as the baseline (left inset figure). Otherwise, ChartSense calculates the average of the bottom lines' *y* values and regards the horizontal line at the average y value as baseline (right inset figure).

Users can fine-tune the baseline position correctly in step 2. ChartSense overlays the detected baseline on the input image, and users can move the baseline using mouse or keyboard until the extracted baseline matches the original baseline. After users confirm the baseline, ChartSense crops the image above the baseline and uses the cropped image as an input to the next bar detection algorithm in step 4. Cropping removes unnecessary outer regions for more accurate detection of bars.

ChartSense assumes that the *y*-axis scale is linear, and asked users to specify any two *y*-positions along with the corresponding data values in step 3. Guidelines and associated text values appear as users specify the *y*-positions and data values as blue lines and text.

After extracting data values, ChartSense reconstructs a bar chart from the extracted data and overlays the bar chart with the input image. In the new bar chart, ChartSense represents bars as transparent rectangles. When users move mouse cursor over the rectangles, ChartSense highlights them with translucent color. Users can easily identify the disagreement between the underlying data and the extracted data by comparing the two. The reconstructed bar chart supports direct manipulation. Users can adjust bar heights, add missing bars, or remove misrecognized bars directly in the reconstructed bar chart until the reconstructed bar chart matches the input bar chart. After the initial data extraction (step 4), the data table shows the extracted data. ChartSense updates the table dynamically upon any user modifications to the reconstructed bar chart, and users can select and highlight a bar by clicking on or hovering mouse cursor over the corresponding cell in the table.

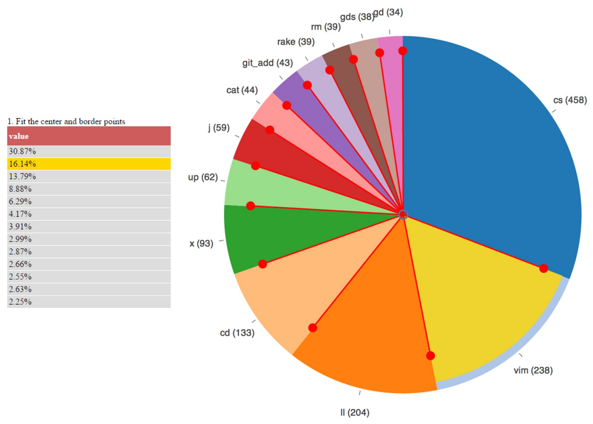
The data extraction process for bar charts shows the general flow of data extraction in ChartSense, which applies to other chart types in general. Therefore, for the remaining chart types, we explain only the unique processes for each chart type while skipping the common parts.

Pie Chart

We make three assumptions: 1) pie charts have no 3D effect; 2) each wedge has a distinct color from its adjacent wedges; and 3) no wedges protrude over the circle's border. 53.16% of pie charts in our corpus satisfies all the three assumptions. The percentages of chart images that are excluded by these assumptions are 37.89%, 6.49%, and 8.42%, respectively.

The data extraction for pie charts consists of only two steps:

1. [CS] Detect the center point and all borders
2. [US] Adjust the center point correctly and add, remove, or modify incorrectly recognized border points


1. Fit the center and border points

value
30.87%
16.14%
13.79%
8.88%
6.29%
4.17%
3.91%
2.99%
2.87%
2.66%
2.55%
2.63%
2.25%

Figure 4. Data extraction from pie charts. The blue circle represents the detected center point and red points represent border points. If users modify the position of the blue circle or red points, the data table is updated immediately.

We focus on explaining center and border points detection algorithm in detail because it is the essential part in the pie chart data extraction. First, ChartSense makes a binary image from a given input image by the same method used for bar charts. Second, it identifies the center and the radius of the pie circle by applying Open morphological transform, Canny edge detection algorithm [3], and Hough transform in turn. After finding the center C and the radius R, ChartSense defines border points by sampling the 1,000 pixels uniformly along the circle whose center is C and radius is one of 0.5R, 0.6R, 0.7R, 0.8R, and 0.9R. While traversing the sampling points sequentially, ChartSense detects a sudden change of pixel values between two consecutive sampling points, and regards the center of the transition points as a border point. This method can fail when there are letters or symbols in the wedges. Therefore, ChartSense runs the border search procedure five times with different radii (0.5R, 0.6R, 0.7R, 0.8R, 0.9R) and use the result that returns the smallest number of border points. This approach is similar to prior work [24] that uses a circular Hough transformation with multiple radii for robustness. After detecting the center and border points, ChartSense overlays them on the input chart. Users fine-tune the center point and the border points, and the data table is updated accordingly (Figure 4).

EVALUATION: CONTROLLED EXPERIMENT

We conducted a controlled experiment to evaluate the effectiveness of ChartSense for data extraction from chart images. The experiment consisted of two parts (Figure 5). In Part 1, we compared ChartSense (CS) to WebPlotDigitizer (WPD) for bar and line charts. In Part 2, we examined the effectiveness of CS only for three additional charts—area, pie, and radar charts—because WPD does not support data extraction from these charts.

Participants and Dataset

We recruited 16 participants (9 males and 7 females) from a university campus recruiting website. Average age of the participants was 22.6, ranging from 20 to 26. They received about $20 for their participation.

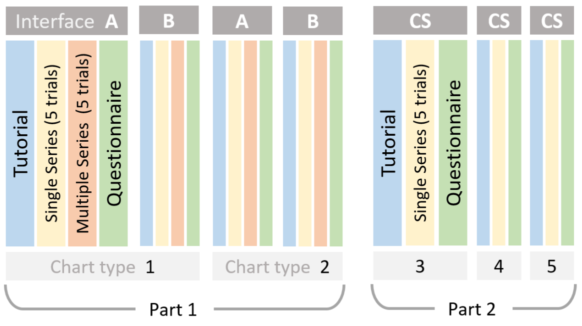


Figure 5. The study consists of two parts. In the first part, we compared ChartSense to WebPlotDigitizer for bar and line charts. In the second part, we evaluated ChartSense for area, radar, and pie charts.

Chart images used for the study were selected from the image corpus we built using Google image search for chart type classification. To calculate the data extraction accuracy, we selected images that had ground truths (i.e., all marks in an image have text labels representing the data values of the marks). For Part 1, we selected 10 bar chart images and 10 line chart images. The images were divided into two categories: single series (with nine marks) and multiple series (two series with nine marks for each). To control the number of marks to be consistent, we trimmed some marks from the original images when necessary. For Part 2, we selected five images per chart type for area, pie, and radar charts. The images contained different number of marks (6 to 16 for area charts, 6 to 13 for pie charts, and 4 to 24 for radar charts). We erased the ground truth before the user study.

Study Design, Procedure, and Setup

We ran the comparison part of the study as a 2 (Interface: CS vs. WPD) × 2 (Chart Type: Bar vs. Line) × 2 (Series: single vs. multiple) within-subject design (see Part 1 in Figure 5). Each participant performed the data extraction tasks from both bar and line charts, for both single and multiple series charts, using both CS and WPD. To avoid the learning effect, we counterbalanced the order of interfaces and chart types.

Each session began with a brief introduction of the study procedure. We then asked the participants to fill out a simple pre-study questionnaire to collect their age, gender and major.

In Part 1, prior to beginning the timed tasks with each block (i.e., a combination of Chart Type and Interface), we demonstrated participants how to extract data from charts using the interfaces for about two minutes. We then asked them to practice extracting data from sample charts until they could get familiar with the data extraction using the interfaces without time constraints. For the timed tasks, participants extracted data five times for each number of series, and saved the result in a local folder. We asked participants to extract data for chart images as accurately and quickly as possible. After they completed each of the four

Question	Bar		Line	
	CS	WPD	CS	WPD
Q1. **This interface was easy to learn.***	6.4	5.0	6.0	4.8
Q2. **This interface was easy to use.***	6.4	4.6	5.9	4.9
Q3. This interface was easy to understand.	5.3	4.0	5.6	4.5
Q4. **This interface was accurate.***	6.3	4.7	6.3	4.8
Q5. **I would like to use this interface again.***	5.7	4.0	5.8	4.1

Table 3. Average Likert scale ratings for CS and WPD using the scale of 1=Strongly disagree and 7=Strongly agree. The ratings of CS were significantly higher than those of WPD for every questions except for Q3 with line chart (gray background).

blocks, we asked participants to fill out a questionnaire. The experiment for Part I took about 40 minutes on average.

Participants went through a similar process in Part 2 but only with CS to extract data from area, pie, and radar charts. After the tutorial, participants extracted data five times for each chart type. After they completed each of the three chart types, we asked participants to fill out a questionnaire. Part 2 took about 50 minutes on average.

Each participant worked on a PC with a client web application running on a Google Chrome web browser. All results (extracted data values) were saved in the local disk. For each condition, we showed the folder that contains five chart image files. We asked participants to extract data from the image files in the order they appear in the folder. For each trial, we timed with a stopwatch, starting when participants opened a file and ending when they saved the result.

Results

We measured and analyzed two dependent variables from the experiment: (1) task completion time and (2) error rate that is the ratio of difference between an extracted value and its ground truth to the ground truth. We also performed a statistical analysis of questionnaire responses.

Part 1: Task Completion Time

We analyzed the task completion time with a 2 (Interface) × 2 (Chart Type) × 2 (Series) repeated-measures analysis of variance (RM-ANOVA).

We found a significant main effect of Interface ($F_{1,15}$=70.57, p<.001) (Figure 6a). We also found significant main effects of Chart Type ($F_{1,15}$=11.09, p<.001) and Series ($F_{1,15}$=69.35, p<.001) (Figure 6b, 6c). It was expected that line charts take more time than bar charts because more interactions are required for line charts regardless of interface being used. Also, it is not surprising that charts with a larger number of series would take more time to complete the task.

For both chart types, CS helped participants finish tasks faster than WPD (Figure 7). We found a significant interaction between Interface and Series ($F_{1,15}$=84.711, p<.001). For both "single" and "multiple" series charts, CS took less time than WPD. More interestingly, for charts that include more marks, the task time of WPD drastically increased; however, CS exhibited a relatively moderate increase in the task completion time.

Part 1: Error Rate

We saved the extracted data and calculated error rate for each extracted data value as follow.

$$\text{error rate} = \frac{|ground\ truth - extracted\ value|}{ground\ truth}$$

For each trial, we took an average of error rates for all extracted values (Figure 8). We analyzed the error rate with a 2 (Interface) × 2 (Chart Type) × 2 (Series) RM-ANOVA.

We found a significant main effect of Interface ($F_{1,15}$=12.785, p=.003). We also found a significant main effect of Chart Type ($F_{1,15}$=478.964, p<.001) and Series ($F_{1,15}$=37.101, p<.001). We found a significant interaction between Interface and Series ($F_{1,15}$=11.449, p=.004), and Chart Type and Series ($F_{1,15}$=9.090, p=.009)

Part 1: Subjective Preference

We ran Friedman Chi-Square tests for the ratings and found a significant difference for all questions (p<.001), indicating that participants felt that CS was easier to use than WPD. The ratings for CS tended to be higher than WPD for other three

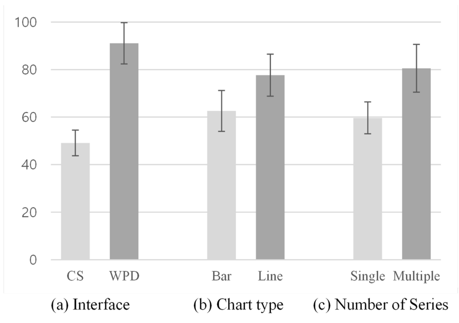


Figure 6. Task completion time (sec) by main effects. Main effects were significant for all three factors.

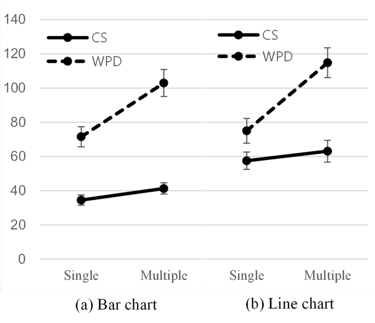


Figure 7. Significant interaction effect between Interface and Number of series on completion time (sec).

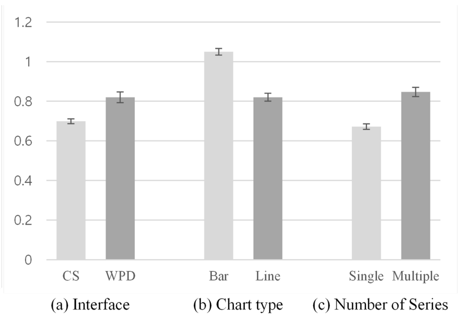


Figure 8. Error rate (%) by main effects. Main effects were significant for all three factors.

questions. The subjective evaluation result is summarized in Table 3.

Part 2: Task Completion Time, Error Rate, and Subjective Preference

The second part of our study only involved CS. The average task completion time and error rate are summarized in Table 4. The subjective evaluation results show that CS interfaces for area, pie and radar charts are easy to learn, easy to use, and easy to understand (ratings are 6/6.56/5.6 for area, 5.94/6.38/5.44 for pie, and 5.75/6.06/5.44 for radar charts).

DISCUSSION AND FUTURE WORK

Chart Type Classification Accuracy

Our quantitative comparison on chart type classification accuracy of ChartSense and ReVision showed that our classification model is more accurate than ReVision's model. When trained with the large image corpus (i.e., union of previously used dataset in ReVision and newly collected dataset), our classification model showed higher accuracy than ReVision for all chart types, but when trained with the small image corpus, ReVision's classification model showed higher accuracy in five chart types—area, Pareto, radar, scatter plot, and Venn diagram. This can be attributable to the fact that the number of images for certain chart types is not sufficient to build an effective CNN model, thus shared features do not properly represent the chart type throughout the convolution layer and pooling layer in the network.

Mixed-initiative Approach

Our controlled user study results supported our hypothesis that ChartSense outperforms WebPlotDigitizer in terms of task completion time, error rate, and subjective satisfaction. We attribute this result to our design decision to take a mixed-initiative approach that harmoniously integrates the state-of-the-art automatic mark extraction techniques with simple yet effective user interactions. We believe our prototype is designed to take advantages from both automatic and manual approaches while mitigating their disadvantages.

Because we could not find any available tools that can extract data from area, pie, and radar charts, we could not conduct a comparative study for these chart types. However, given that error rates are around 3% (Table 4), we believe ChartSense could be practically useful.

In statistical analysis, we excluded data points from repetitive unintentional mistakes with WPD interface and outliers identified with interquartile range (i.e., Tukey's test). While using WPD for line chart, some participants clicked on the first data points rather than a point on the *x*-axis, which is supposed to be the reference value for data extraction. Since it led to huge errors, we excluded such data points before applying Tukey's test.

It was surprising that line chart data extraction showed significantly lower error rate than bar chart data extraction, as we expected that it would be easier for participants to fine-tune bar tops than they fine-tune data points in line charts. It might be because of more user interactions in line charts did more good (more information) than harm (potential error) in our mixed-initiative approach.

Chart Type	Area	Pie	Radar
Time (s)	60.40	32.51	61.63
Error rate (%)	1.89	2.21	3.19

Table 4. Average task completion time and error rate for area, pie, and radar charts by ChartSense.

Generalizability

We made a few assumptions for input chart images that ChartSense can handle. Although our data extraction algorithms cover a significant number of real-world charts even with the assumptions, more work is needed to make our approach more generally applicable. It is especially important if users are interested in redesigning existing charts using perceptually more effective encodings because many poorly designed charts are excluded by the assumptions. An interesting future research direction would be to integrate ChartSense with chart restyling techniques proposed by Savva et al. [24] or Harper and Agrawala [9].

Text-region-finding detection algorithm with high accuracy can improve the performance of our interactive data extraction algorithm because the performance of OCR is significantly affected by text-region-finding results. Text labels in charts play an important role in chart data extraction algorithms since they serve as reference points to convert marks into values. Since there was no practical text-region-finding algorithms available, users have to manually specify *y*-positions and type in corresponding data values in current chart data extraction tools. An actionable future research direction would be to developing a text-region-finding algorithm for charts. It might be less challenging to detect text regions in chart images than in general images because the text labels often are located near axes in charts.

CONCLUSION

We presented ChartSense: a system that classifies chart type and extracts underlying data from chart images by an interactive data extraction algorithm. Our quantitative evaluation showed that ChartSense achieves higher classification accuracy than the previous classification system, ReVision. We conducted a controlled experiment to investigate whether it could improve users' performance to extract data for two chart types: bar and line charts. We found that participants could extract data faster and more accurately with ChartSense than WebPlotDigitizer. Furthermore, ChartSense is practically useful for three additional chart types—area, pie, and radar charts.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIP) (No. NRF-2014R1A2A2A03006998 and NRF-2016R1A2B2007153). The ICT at Seoul National University provided research facilities for this study. Jinwook Seo is the corresponding author.

REFERENCES

1. Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE Trans. Vis. Comput. Graphics* 17, 12: 2301-2309.
2. Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, Grégoire Montavon, Geneviève, Orr, Klaus-Robert Müller (eds.). Springer Berlin Heidelberg, 421-436.
3. John Canny. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell* PAMI-8, 6: 679-698.
4. Zhe Chen, Michael Cafarella, and Eytan Adar. 2015. DiagramFlyer: A search engine for data-driven diagrams. In *Proceedings of the 24th International Conference on World Wide Web Companion* (WWW '15), 183-186.
5. Sagnik Ray Choudhury and Clyde Lee Giles. 2015. An architecture for information extraction from figures in digital libraries. In *Proceedings of the 24th International Conference on World Wide Web Companion* (WWW '15), 667-672.
6. Jinglun Gao, Yin Zhou, and Kenneth E. Barner. 2012. View: Visual information extraction widget for improving chart images accessibility. In *Proceedings of the 19th IEEE International Conference on Image Processing* (ICIP '12), 2865-2868
7. Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing ambiguity in natural language interfaces for data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology* (UIST '15), 489-500.
8. Arnd Gross, Sibylle Schirm, and Markus Scholz. 2014. Ycasd–a tool for capturing and scaling data from graphical representations. *BMC bioinformatics* 15, 1: 219.
9. Jonathan Harper and Maneesh Agrawala. 2014. Deconstructing and restyling D3 visualizations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (UIST '14), 253-262.
10. Christopher G. Healey, Sarat Kocherlakota, Vivek Rao, Reshma Mehta, and Renee St. Amant. 2008. Visual perception and mixed-initiative interaction for assisted visualization design. *IEEE Trans. Vis. Comput. Graphics* 14, 2: 396-411.
11. Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (CHI '99), 159-166.
12. Weihua Huang, Chew Lim Tan, and Wee Kheng Leow. 2004. Model-based chart image recognition. In *Graphics Recognition. Recent Advances and Perspectives,* Josep Lladós and Young-Bin Kwon (eds.). Springer Berlin Heidelberg, 87-99.
13. Weihua Huang, Ruizhe Liu, and Chew Lim Tan. 2007. Extraction of vectorized graphical information from scientific chart images. In *Proceedings of the 9th International Conference on Document Analysis and Recognition* (ICDAR '07)*,* 521-525.
14. Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia* (MM '14), 675-678.
15. Nicholas Kong and Maneesh Agrawala. 2012. Graphical overlays: Using layered elements to aid chart reading. *IEEE Trans. Vis. Comput. Graphics* 18, 12: 2631-2638.
16. Nicholas Kong, Marti A. Hearst, and Maneesh Agrawala. 2014. Extracting references between text and charts via crowdsourcing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '14), 31-40.
17. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th Neural Information Processing Systems* (NIPS '12), 1106-1114.
18. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11: 2278-2324.
19. Yan Liu, Xiaoqing Lu, Yeyang Qin, Zhi Tang, and Jianbo Xu. 2013. Review of chart recognition in document images. In *IS&T/SPIE Electronic Imaging*, 865410-865410.
20. Jock Mackinlay. 1986. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.* 5, 2: 110-141.
21. Gonzalo Gabriel Méndez, Miguel A. Nacenta, and Sebastien Vandenheste. 2016. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '16), 4073-4085
22. Aria Pezeshk and Richard L. Tutwiler. 2011. Automatic feature extraction and text recognition from scanned topographic maps. *IEEE Trans. Geosci. Remote Sens* 49, 12: 5047-5063.
23. Ankit Rohatgi. 2015. WebPlotDigitizer, Version 3.8. Retrieved September 22, 2015 from http://arohatgi.info/WebPlotDigitizer
24. Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. 2011.

ReVision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (UIST '11), 393-402.

25. Julia Schwarz, Scott Hudson, Jennifer Mankoff, and Andrew D. Wilson. 2010. A framework for robust and flexible handling of inputs with uncertainty. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology* (UIST '10), 47-56.
26. Mingyan Shao and Robert P. Futrelle. 2006. Recognition and classification of figures in PDF documents. In *Graphics Recognition. Ten Years Review and Future Perspectives,* Wenyin Liu and Josep Lladós (eds.). Springer Berlin Heidelberg, 231-242.
27. Noah Siegel, Zachary Horvitz, Roie Levin, Santosh Divvala, and Ali Farhadi. 2016. FigureSeer: Parsing result-figures in research papers. In *Proceedings of the 14th European Conference on Computer Vision* (ECCV '16), 664-680.
28. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR '15), 1-9.
29. Karl Tombre, Salvatore Tabbone, Loïc Pélissier, Bart Lamiroy, and Philippe Dosch. 2002. Text/graphics separation revisited. In *Document Analysis Systems V,* Daniel Lopresti, Jianying Hu, and Ramanujan Kashi (eds.). Springer Berlin Heidelberg, 200-211.
30. Bas Tummers. 2015. DataTheif III. Retrieved September 22, 2015 from http://www.datathief.org/
31. Colin Ware, 2012. Information Visualization: perception for design (3rd. ed.). Elsevier.
32. Qixiang Ye and David Doermann. 2015. Text detection and recognition in imagery: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 7: 1480-1500.